МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
БОРИСОГЛЕБСКИЙ ФИЛИАЛ
(БФ ФГБОУ ВО «ВГУ»)

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО УЧЕБНОЙ ДИСЦИПЛИНЕ
Архитектура компьютера

Методические указания для обучающихся по освоению дисциплины

Приступая к изучению учебной дисциплины, прежде всего обучающиеся должны ознакомиться с учебной программой дисциплины. Электронный вариант рабочей программы размещён на сайте БФ ВГУ.

Обучающиеся должны иметь четкое представление о:

- перечне и содержании компетенций, на формирование которых направлена дисциплина;
- основных целях и задачах дисциплины;
- планируемых результатах, представленных в виде знаний, умений и навыков, которые должны быть сформированы в процессе изучения дисциплины;
- количестве часов, предусмотренных учебным планом на изучение дисциплины, форму промежуточной аттестации;
- количестве часов, отведенных на контактную и самостоятельную работу;
- формах контактной и самостоятельной работы;
- структуре дисциплины, основных разделах и темах;
- системе оценивания ваших учебных достижений;
- учебно-методическом и информационном обеспечении дисциплины.

Знание основных положений, отраженных в рабочей программе дисциплины, поможет обучающимся ориентироваться в изучаемом курсе, осознавать место и роль изучаемой дисциплины, строить свою работу в соответствии с требованиями, заложенными в программе.

Основными формами контактной работы по дисциплине являются лекции и лабораторные работы, посещение которых обязательно для всех студентов.

В ходе лекционных занятий следует не только слушать излагаемый материал и кратко его конспектировать, но очень важно участвовать в анализе примеров, предлагаемых преподавателем, в рассмотрении и решении проблемных вопросов, выносимых на обсуждение. Необходимо критически осмысливать предлагаемый материал, задавать вопросы как уточняющего характера, помогающие уяснить отдельные излагаемые положения, так и вопросы продуктивного типа, направленные на расширение и углубление сведений по изучаемой теме, на выявление недостаточно освещенных вопросов, слабых мест в аргументации и т.п.

В ходе выполнения лабораторных работ студент выполняет задания, содержащиеся в методическом пособии дисциплины в соответствии с имеющимися указаниями. Далее студент самостоятельно выполняет индивидуальное задание.

Обязательно следует познакомиться с критериями оценивания каждой формы контроля – это поможет избежать недочетов, снижающих оценку за работу.

При подготовке к промежуточной аттестации необходимо повторить пройденный материал в соответствии с учебной программой, примерным перечнем вопросов, выносящихся на зачет. Рекомендуется использовать конспекты лекций и источники, перечисленные в списке литературы в рабочей программе дисциплины, а также ресурсы электронно-библиотечных систем. Необходимо обратить особое внимание на темы учебных занятий, пропущенных по разным причинам. При необходимости можно обратиться за консультацией и методической помощью к преподавателю.

Методические материалы для обучающихся по освоению теоретических вопросов дисциплины

№	Тема лекции	Рассматриваемые вопросы
1	Создание и эволюция ЭВМ	Технические предпосылки и практические потребности создания ЭВМ. Эволюция ЭВМ. Основные классы современных ЭВМ. Портативные компьютеры.
2	Программное управление	Автоматизация вычислительного процесса. Программирование на языке ассемблер.
3	Основные блоки ЭВМ,	Микропроцессоры. Запоминающие устройства ПК. Системные

	их назначение и функциональные характеристики	платы и чипсет. Интерфейсная система ПК.
4	Внешние устройства ПК	Внешние запоминающие устройства. Устройства ввода. Устройства вывода.

Методические материалы для обучающихся по подготовке к практическим/лабораторным занятиям

№	Тема занятия	Рассматриваемые вопросы
1	Создание и эволюция ЭВМ	Команда MOV и арифметические команды. Команды логических операций, сдвигов, выделение битовых полей.
2	Программное управление	Ветвления. Команды передачи управления. Команда LOOP. Обработка данных в цикле без ветвлений. Обработка данных в цикле с использованием ветвлений. Работа с видеопамятью (виртуальный дисплей). Разные задачи. Задачи на встроенном ассемблере системы Turbo Pascal.

Тематика рефератов/докладов/эссе, методические рекомендации по выполнению контрольных и курсовых работ, иные материалы

Примерный перечень вопросов к зачету с оценкой по дисциплине «Архитектура компьютера»

1. История развития вычислительной техники. Поколения ЭВМ. Классификация ЭВМ.
2. Архитектура ЭВМ. Принципы фон-Неймана.
3. Персональный компьютер. Компоненты ПК. Магистрально-модульный принцип.
4. Функциональная структура микропроцессора. Устройство управления, арифметико-логическое устройство, интерфейсная часть микропроцессора.
5. Микропроцессор. Характеристики, функции и виды процессоров.
6. Микропроцессорная память. Регистры (базовый набор x86). Общая структурная схема микропроцессора.
7. Режимы работы процессора. Адресация памяти.
8. Физические компоненты микропроцессора. Конвейеризация. Адресация в реальном и защищенном режиме.
9. Кэш-память. Кэширование.
10. Материнская плата. Ее компоненты. Чипсет. Базовая система ввода/вывода (BIOS). Ее основные функции. Конфигурирование компьютера.
11. Внутренние интерфейсы (системная шина, AGP, доступ к памяти, Шина HyperTransport, ata(ide), SATA, SCSI).
12. Внешние интерфейсы.
13. Память компьютера. Виды памяти.
14. Устройство и принцип работы жёсткого диска.
15. Устройства ввода информации. Клавиатура и мышь.
16. Устройства вывода информации.
17. Оптические приводы.
18. Графические технологии.
19. Технологии трехмерной графики (LOD, mip mapping, композитные текстуры, трехмерные текстуры, методы фильтрации текстур).
20. Современные тенденции развития архитектуры ЭВМ.
21. Магистрально-модульный принцип. Принцип открытой архитектуры. Структурная схема персональной ЭВМ.
22. Режимы работы компьютеров. Однопрограммный режим. Многопрограммный режим.
23. Система прерываний программ в ПК. Пользовательские, системные и справочные прерывания.

24. Язык ассемблера. Основные компоненты языка ассемблер. Алфавит, идентификаторы, константы, команды.
25. Арифметические команды. Особенности выполнения команд сложения и вычитания. Умножение и деление. Команды INC, DEC.
26. Команда сравнения. Команды безусловного и условного перехода. Циклы.

Примерный перечень тем курсовых работ по дисциплине «Архитектура компьютера»

1. БЭСМ.
2. Принтеры.
3. Плоттеры.
4. Клавиатуры.
5. Мыши.
6. Джойстики и трекболы.
7. Сканеры.
8. Звуковые карты.
9. Сетевые карты.
10. Элементная база первых поколений компьютеров.
11. Технологические процессы производства современных СБИС.
12. Мониторы.
13. Проекторы.
14. Процессоры AMD.
15. Суперкомпьютеры.

Примерный перечень тем докладов по дисциплине «Архитектура компьютера»

1. Открытая архитектура ЭВМ.
2. Основные различия операционных систем.
3. Кэш-память: виды, принцип работы.
4. Аппаратная платформа Макинтош
5. Видеоадаптер EGA, VGA, SVGA.
6. Виды твердотельных накопителей.
7. Сравнительная характеристика серверов.
8. Сервера фирмы Apple.
9. Сервера фирмы HP.
10. Нестандартные устройства ввода информации.
11. Коммуникаторы.
12. Современные ноутбуки.
13. Профессиональные графические планшеты.
14. Перспективы развития мультимедийных технологий.
15. Домашний сервер.
16. Анализ файловых систем.
17. Технология записи, чтения и хранения информации на жестком диске.
18. Планшетные ЭВМ.
19. Терминальные учебные классы.
20. Сетвые хранилища данных.

Методические материалы для выполнения лабораторных работ по дисциплине «Архитектура компьютера»

Представление данных в компьютерах

Для того чтобы освоить программирование на ассемблере, неизбежно приходится знакомиться с двоичными и шестнадцатеричными числами. В некоторых случаях в

тексте программы можно обойтись и обычными десятичными числами, но без понимания того, как на самом деле хранятся данные в памяти компьютера, очень трудно использовать логические и битовые операции и многое другое.

Двоичная система счисления

Практически все существующие сейчас компьютерные системы, включая Intel, используют для всех вычислений двоичную систему счисления. В их электрических цепях напряжение может принимать два значения, и эти значения назвали нулем и единицей. Двоичная система счисления как раз и использует только эти две цифры, а вместо степеней десяти, как в обычной десятичной системе, здесь используют степени двойки.

Чтобы перевести двоичное число в десятичное, надо сложить двойки в степенях, соответствующих позициям, где в двоичном стоят единицы. Например:

$10010110b = 1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = 128+16+4+2 = 150$

Чтобы перевести десятичное число в двоичное, можно, например, просто делить его на 2, записывая 0 каждый раз, когда число делится на два, и 1, когда не делится.

	Остаток	**Разряд**
150/2 = 75	0	0
75/2 = 37	1	1
37/2 = 18	1	2
18/2 = 9	0	3
9/2 = 4	1	4
4/2 = 2	0	5
2/2 = 1	0	6
1/2 = 0	1	7
Результат: 10010110b		

Чтобы отличать двоичные числа от десятичных, в ассемблерных программах в конце каждого двоичного числа ставят букву «b» (двоичные числа в модели не реализованы и в коде не распознаются).

Биты, байты и слова

Минимальная единица информации называется битом. Бит может принимать только два значения — обычно 0 и 1. На самом деле эти значения совершенно необязательны — один бит может принимать значения «да» и «нет», показывать присутствие и отсутствие жесткого диска, и т.п. — важно лишь то, что бит имеет только два значения. Но далеко не все величины принимают только два значения, а значит, для их описания нельзя обойтись одним битом.

Единица информации размером восемь бит называется байтом. Байт — это минимальный объем данных, который реально может использовать компьютерная программа. Даже чтобы изменить значение одного бита в памяти, надо сначала считать байт, содержащий его. Биты в байте нумеруют справа налево, от нуля до семи, нулевой бит часто называют младшим битом, а седьмой — старшим.

Так как всего в байте восемь бит, байт может принимать до $2^8 = 256$ разных значений. Байт используют для представления целых чисел от 0 до 255, целых чисел со знаком от -128 до +127, набора символов ASCII или переменных, принимающих менее 256 значений, например для представления десятичных чисел от 0 до 99.

Следующий по размеру базовый тип данных — слово. Размер одного слова в процессорах Intel — два байта.

Биты с 0 по 7 составляют младший байт слова, а биты с 8 по 15 — старший. В слове содержится 16 бит, а значит, оно может принимать до $2^{16} = 65536$ разных значений. Слова используют для представления целых чисел без знака со значениями 0 — 65535, целых чисел со знаком со значениями от -32768 до +32767, адресов сегментов и

смещений при 16-битной адресации. Два слова подряд образуют двойное слово, состоящее из 32 бит. Байты и слова — основные типы данных, с которыми производится работа.

ВАЖНО! В компьютерах, использующих процессоры Intel, все данные хранятся так, что младший байт находится по младшему адресу, так что слова записываются задом наперед, то есть сначала (по младшему адресу) записывают последний (младший) байт, а потом (по старшему адресу) записывают первый (старший) байт. Если из программы всегда обращаться к слову как к слову, это не оказывает никакого влияния. Но если вы хотите прочитать первый (старший) байт из слова в памяти, придется увеличить адрес на 1.

Шестнадцатеричная система счисления

Главное неудобство двоичной системы счисления — это размеры чисел, с которыми приходится обращаться. На практике с двоичными числами работают, только если необходимо следить за значениями отдельных бит, а когда размеры переменных превышают хотя бы четыре бита, используется шестнадцатеричная система. Эта система хороша тем, что она гораздо более компактна, компактнее десятичной, и тем, что перевод в двоичную систему и обратно происходит очень легко. В шестнадцатеричной системе используется 16 «цифр»: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F; и номер позиции цифры в числе соответствует степени, в которую надо возвести число 16, так что:

$96h = 9 * 16^1 + 6 * 16^0 = 150$

Перевод в двоичную систему и обратно осуществляется крайне просто — вместо каждой шестнадцатеричной цифры подставляют соответствующее четырехзначное двоичное число:

9h = 1001b, 6h = 0110b, 96h = 10010110b

В ассемблерных программах при записи чисел, начинающихся с A,B,C,D,E,F, в начале приписывается цифра 0, чтобы нельзя было спутать такое число с именем метки или другим идентификатором. После шестнадцатеричных чисел ставится буква «h». Например, число 0Ah (десятичное 10) без предшествующего нуля выглядит точно так же, как имя регистра AH.

Десятичное	Двоичное	Шестнадцатиричное
0	0000b	00h
1	0001b	01h
2	0010b	02h
3	0011b	03h
4	0100b	04h
5	0101b	05h
6	0110b	06h
7	0111b	07h
8	1000b	08h
9	1001b	09h
10	1010b	0Ah
11	1011b	0Bh
12	1100b	0Ch
13	1101b	0Dh
14	1110b	0Eh
15	1111b	0Fh

16	10000b	10h

Числа со знаком

Легко использовать байты или слова для представления целых положительных чисел — от 0 до 255 или 65535 соответственно. Чтобы использовать те же самые байты или слова для представления отрицательных чисел, существует специальная операция, известная как дополнение до двух. Для изменения знака числа выполняют инверсию, то есть заменяют в двоичном представлении числа все единицы нулями и нули единицами, а затем прибавляют 1. Например, пусть используются переменные типа слова:

150 = 0096h = 0000 0000 1001 0110b

инверсия дает:

1111 1111 0110 1001b+1 = 1111 1111 0110 1010b = 0FF6Ah

Проверим, что полученное число на самом деле -150: сумма с +150 должна, быть равна нулю:

+150 + (-150) = 0096h + FF6Ah = 10000h

Единица в 16-м разряде не помещается в слово, и значит, мы действительно получили 0. В этом формате старший (7-й или 15-й для байта или слова соответственно) бит всегда соответствует знаку числа: 0 — для положительных и 1 — для отрицательных. Таким образом, схема с использованием дополнения до двух выделяет для положительных и отрицательных чисел равные диапазоны: -128 — +127 — для байта, -32768 — +32767 — для слов.

Логические операции

Один из широко распространенных вариантов значений, которые может принимать один бит, — это значения «правда» и «ложь», используемые в логике, откуда происходят так называемые «логические операции» над битами. Так, если объединить «правду» и «правду» — получится «правда», а если объединить «правду» и «ложь» — «правды» не получится (разве что «полуправда», но такое значение отсутствует). В ассемблере нам встретятся четыре основные операции — И (AND), ИЛИ (OR), исключающее ИЛИ (XOR) и отрицание (NOT), действие которых приводится в таблице.

И	ИЛИ	Исключающее ИЛИ	НЕ
0 AND 0 = 0	0 OR 0 = 0	0 XOR 0 = 0	NOT 0 = 1
0 AND 1 = 0	0 OR 1 = 1	0 XOR 1 = 1	NOT 1 = 0
1 AND 0 = 0	1 OR 0 = 1	1 XOR 0 = 1	
1 AND 1 = 1	1 OR 1 = 1	1 XOR 1 = 0	

Все эти операции побитовые, поэтому, чтобы выполнить логическую операцию над числом, надо перевести его в двоичный формат и выполнить операцию над каждым битом, например:

96h AND 0Fh = 10010110b AND 00001111b = 00000110b = 06h

Организация памяти

Память с точки зрения процессора представляет собой последовательность байт, каждому из которых присвоен уникальный адрес. Он может принимать значения от 0 до 2^{32}-1 (4 гигабайта) в современных компьютерах, в то время как в данной программе объем памяти составляет от 0 до 2^{18}-1 (256 килобайт). Программы же могут работать с памятью как с одним непрерывным массивом или как с несколькими массивами (сегментированные модели памяти). Во втором случае для задания адреса любого байта требуется два числа — адрес начала массива и адрес искомого байта внутри массива. Помимо основной памяти программы могут использовать регистры — специальные ячейки памяти, расположенные физически внутри процессора, доступ к которым осуществляется не по адресам, а по именам.

Регистры процессора

Регистры общего назначения

16-битные регистры AX, BX, CX, DX могут использоваться без ограничений для любых целей. В процессорах 8086 – 80286 все регистры имели размер 16 бит и назывались именно так, а 32-битные EAX, EBX, ECX и EDX появились с введением 32-битной архитектуры в 80386. Также, отдельные байты в 16-битных регистрах AX – DX тоже имеют свои имена и могут использоваться как 8-битные регистры. Старшие байты этих регистров называются AH, BH, CH, DH, а младшие — AL, BL, CL, DL. Буквы H и L в их именах происходят от слов HIGH и LOW - больший(старший) и меньший(младший) соответственно.

Другие четыре регистра общего назначения — SI, DI, BP и SP — имеют более конкретное назначение и могут применяться для хранения всевозможных временных данных, только когда они не используются по назначению. Так же, как и с регистрами AX – DX, регистры SI, DI, BP и SP являются младшими половинами регистров ESI, EDI, EBP и ESP соответственно, которые, как уже и говорилось, появились с введением 32-битной архитектуры в 80386.

Сегментные регистры

При использовании сегментированной модели памяти для формирования любого адреса применяются два числа — адрес начала сегмента и смещение искомого байта относительно этого начала. В реальности программа обращается к сегментам, используя вместо настоящего адреса начала сегмента 16-битное число, называемое селектором. В процессорах Intel 8086 предусмотрено четыре шестнадцатибитных регистра — CS, DS, ES, SS, используемых для хранения селекторов. Это не значит, что программа не может одновременно работать с большим количеством сегментов памяти, — в любой момент времени можно изменить значения, записанные в этих регистрах (В программе это не реализовано, то есть из сегментных регистров значения можно только считывать, изначально все четыре регистра указывают на разные сегменты).

В отличие от регистров DS, ES, которые называются регистрами сегментов данных, регистры CS и SS отвечают за сегменты двух особенных типов — сегмент кода и сегмент стека. Сегмент кода содержит программу, исполняющуюся в данный момент, так что запись нового селектора в этот регистр приводит к тому, что далее будет исполнена не следующая по тексту программы команда, а команда из кода, находящегося в другом сегменте, с тем же смещением. Смещение следующей выполняемой команды всегда хранится в специальном регистре — IP (указатель инструкции, тридцатидвухтибитная форма EIP), запись в который также приведет к тому, что следующей будет исполнена какая-нибудь другая команда. На самом деле все команды передачи управления — перехода, условного перехода, цикла и т.п. — и осуществляют эту самую запись в CS и IP.

Стек

Стек — это специальным образом организованный участок памяти, используемый для временного хранения данных, для передачи параметров вызываемым подпрограммам и для сохранения адреса возврата при вызове процедур и прерываний. Легче всего представить стек в виде стопки листов бумаги (это одно из значений слова «stack» в английском языке) — вы можете класть и забирать листы бумаги только с вершины стопки. Таким образом, если записать в стек числа 1, 2, 3, то при чтении они будут получаться в обратном порядке — 3, 2, 1. Стек располагается в сегменте памяти, описываемом регистром SS, а текущее смещение вершины стека записано в регистре SP, причем при записи в стек значение этого смещения уменьшается, то есть стек растет вниз от максимально возможного адреса.

При вызове подпрограммы параметры в большинстве случаев помещают в стек, а в BP записывают текущее значение SP. Тогда, если подпрограмма использует стек для временного хранения своих данных, SP изменится, но BP можно будет использовать

для того, чтобы считывать значения параметров напрямую из стека (их смещения будут записываться как BP + номер параметра).

Регистр флагов

Еще один важный регистр, использующийся при выполнении большинства команд, — регистр флагов FLAGS. В этом регистре каждый бит является флагом, то есть устанавливается в 1 при определенных условиях или установка его в 1 изменяет поведение процессора.

- CF — флаг переноса. Устанавливается в 1, если результат предыдущей операции не уместился в приемнике и произошел перенос из старшего бита или если требуется заем (при вычитании), иначе устанавливается в 0. Например, после сложения слова 0FFFFh и 1, если регистр, в который надо поместить результат, — слово, в него будет записано 0000h и флаг CF = 1.
- PF — флаг четности. Устанавливается в 1, если младший байт результата предыдущей команды содержит четное число бит, равных 1; устанавливается в 0, если число единичных бит нечетное. (Это не то же самое, что делимость на два. Число делится на два без остатка, если его самый младший бит равен нулю, и не делится, если он равен 1.)
- AF — флаг полупереноса или вспомогательного переноса. Устанавливается в 1, если в результате предыдущей операции произошел перенос (или заем) из третьего бита в четвертый.
- ZF — флаг нуля. Устанавливается в 1, если результат предыдущей команды — ноль.
- SF — флаг знака. Этот флаг всегда равен старшему биту результата.
- TF — флаг ловушки. Этот флаг был предусмотрен для работы отладчиков, не использующих защищенный режим. Установка его в 1 приводит к тому, что после выполнения каждой команды программы управление временно передается отладчику.
- IF — флаг прерываний (не используется). Установка этого флага в 1 приводит к тому, что процессор перестает обрабатывать прерывания от внешних устройств. Обычно его устанавливают на короткое время для выполнения критических участков кода.
- DF — флаг направления. Этот флаг контроллирует поведение команд обработки строк — когда он сброшен в 0, строки обрабатываются в сторону увеличения адресов, а когда DF = 1 — наоборот.
- OF — флаг переполнения. Этот флаг устанавливается в 1, если результат предыдущей арифметической операции над числами со знаком выходит за допустимые для них пределы. Например, если при сложении двух положительных чисел получается число со старшим битом, равным единице (то есть отрицательное) и наоборот.

Флаги IOPL (уровень привелегий ввода-вывода) и NT (вложенная задача) применяются в защищенном режиме.

Способы адресации

Регистровая адресация

Операнды могут располагаться в любых регистрах общего назначения и сегментных регистрах. В этом случае в тексте программы указывается название соответствующего регистра, например команда, копирующуя в регистр AX содержимое регистра BX, записывается как:

```
mov    ax,bx
```

Непосредственная адресация

Некоторые команды (все арифметические команды, кроме деления) позволяют указывать один из операндов непосредственно в тексте программы, например команда

```
mov    ax,2
```

помещает в регистр AX число 2.

Прямая адресация

Если известен адрес операнда, располагающегося в памяти, можно использовать этот адрес. Если операнд — слово, находящееся в сегменте, на который указывает ES, со смещением от начала сегмента 0001, то команда

```
mov   ax,es:[0001]
```

поместит это слово в регистр AX. Если селектор сегмента данных находится в DS, имя сегментного регистра при прямой адресации можно не указывать, DS используется по умолчанию. Прямая адресация иногда называется адресацией по смещению.

Косвенная адресация

По аналогии с регистровыми и непосредственными операндами адрес операнда в памяти также можно не указывать непосредственно, а хранить в любом регистре. До 80386 для этого можно было использовать только BX, SI, DI и BP. Например, следующая команда помещает в регистр AX слово из ячейки памяти, селектор сегмента которой находится в DS, а смещение — в BX:

```
mov   ax,[bx]
```

Как и в случае прямой адресации, DS используется по умолчанию, но не во всех случаях: если смещение берут из регистра BP, то в качестве сегментного регистра используется SS (в программе это не реализовано, т.е. DS всегда используется по умолчанию, если не указан сегмент).

Адресация по базе со сдвигом

Теперь скомбинируем два предыдущих метода адресации: следующая команда

```
mov   ax,[bx]+2
```

помещает в регистр AX слово, находящееся в сегменте, указанном в DS, со смещением на 2 большим, чем число, находящееся в BX. Так как слово занимает ровно два байта, эта команда поместила в AX слово, непосредственно следующее за тем, которое есть в предыдущем примере. Такая форма адресации используется в тех случаях, когда в регистре находится адрес начала структуры данных, а доступ надо осуществить к какому-нибудь элементу этой структуры. Другое важное применение адресации по базе со сдвигом — доступ из подпрограммы к параметрам, данным в стеке, используя регистр BP в качестве базы и номер параметра в качестве смещения. Другие допустимые формы записи этого способа адресации:

```
mov   ax,[bx+2]
mov   ax,2[bx]
```

Все эти формы записи означают одно и то же действие, но в программе реализован только первый способ, когда сдвиг в виде числа с сомволом + (плюс) записывается после закрывающейся квадратной скобки.

До 80386 в качестве базового регистра можно было использовать только BX, BP, SI или DI и сдвиг мог быть только байтом или словом (со знаком). С помощью этого метода можно организовывать доступ к одномерным массивам байт: смещение соответствует адресу начала массива, а число в регистре — индексу элемента массива, который надо считать. Очевидно, что, если массив состоит не из байт, а из слов, придется умножать базовый регистр на два (умножение при адресации не реализовано).

Адресация по базе с индексированием

В этом методе адресации смещение операнда в памяти вычисляется как сумма чисел, содержащихся в двух регистрах, и смещения, если оно указано. Все следующие команды — это разные формы записи одного и того же действия:

```
mov   ax,[bx+si]+2    ; реализован только такой вариант
mov   ax,[bx+si+2]
mov   ax,[bx][si]+2
mov   ax,[bx+2][si]
mov   ax,[bx][si+2]
mov   ax,2[bx][si]
```

В регистр AX помещается слово из ячейки памяти со смещением, равным сумме чисел, содержащихся в BX и SI, и числа 2. Из шестнадцатибитных регистров так можно складывать только BX+SI, BX+DI, BP+SI и BP+DI. Так можно прочитать, например, число из двумерного массива: если задана таблица 10x10 байт, 2 — смещение ее начала от начала сегмента данных, BX = 20, а SI = 7, приведенные команды прочитают слово, состоящее из седьмого и восьмого байт третьей строки.

ВАЖНО!

В некоторых случаях при использовании операнда в памяти возникает неопределенность с размером данных, которые должен описывать данный операнд. Например в команде

```
mov [bx],5
```

не понятно, что нужно записать в память: байт или слово; и при попытке откомпилировать такой код компилятор выдаст ошибку. В таком случае необходимо указать тип перед аргументом в памяти с помощью директивы PTR: если нужно описать байт, то пишут BYTE PTR, а если слово, то WORD PTR. Таким образом команда примет вид

```
mov byte ptr [bx],5
```

если нужен байт, или

```
mov word ptr [bx],5
```

если нужно слово.

Команды

Внимание! Команды, которые описаны в данном разделе, могут отличаться от таких же команд, используемых реальными процессорами.

Пересылка данных

MOV приемник, источник

Пересылка данных

Базовая команда пересылки данных. Копирует содержимое источника в приемник, источник не изменяется. Команда MOV действует аналогично операторам присваивания из языков высокого уровня, то есть команда

```
mov ax,bx
```

эквивалентна выражению

```
ax := bx;
```

языка Паскаль или

```
ax = bx;
```

языка C, за исключением того, что команда ассемблера позволяет работать не только с переменными в памяти, но и со всеми регистрами процессора.

В качестве источника для MOV могут использоваться: число (непосредственный операнд), регистр общего назначения, сегментный регистр или операнд, находящийся в памяти (то есть адрес памяти). В качестве приемника — регистр общего назначения или память. Оба операнда должны быть одного и того же размера — байт или слово.

Нельзя выполнять пересылку данных с помощью MOV из одного адреса памяти в другой — эту операцию выполняют двумя командами MOV (из памяти в регистр и уже из него в другой адрес памяти) или парой команд PUSH/POP.

XCHG операнд1, операнд2

Обмен операндов между собой

Содержимое операнда 2 копируется в операнд 1, а старое содержимое операнда 1 — в операнд 2. XCHG можно выполнять над двумя регистрами или над регистром и памятью.

```
xchg ax,bx   ; то же, что три команды на языке Pascal:
             ; temp := ax; ax := bx; bx := temp;
xchg al,al   ; а эта команда не делает ничего
```

PUSH источник

Поместить данные в стек

Помещает содержимое источника в стек. Источником может быть регистр, сегментный регистр или память. Фактически эта команда копирует содержимое источника в память по адресу SS:[SP] и уменьшает SP на размер источника в байтах (2). Команда PUSH практически всегда используется в паре с POP (считать данные из стека). Так, например, чтобы скопировать содержимое из одного адреса памяти в другой (что нельзя выполнить одной командой MOV), можно использовать такую последовательность команд:

```
push    ds:[1234h]
pop     es:[4321h]
; теперь содержимое памяти по адресу ES:[4321h]
; такое же, как и в DS:[1234h]
```

Другое частое применение команд PUSH/POP — временное хранение переменных, например:

```
push    ax  ; сохраняет текущее значение AX
; здесь располагаются какие-нибудь команды,
; которые используют AX, например ADD, CMP
pop     ax  ; восстанавливает старое значение AX
```

Команда PUSH SP на 8086 помещает в стек значение SP уже уменьшенным на два, в то время как начиная с 80286 SP помещается в стек до того, как эта же команда его уменьшит.

POP приемник

Считать данные из стека

Помещает в приемник слово, находящееся в вершине стека, увеличивая SP на 2. POP выполняет действие, полностью обратное PUSH. Приемником может быть регистр общего назначения или память.

XLAT

Трансляция в соответствии с таблицей

Помещает в AL байт из таблицы в памяти по адресу DS:BX со смещением относительно начала таблицы, равным AL. В качестве примера использования XLAT можно написать следующий вариант преобразования шестнадцатеричного числа в ASCII-код соответствующего ему символа:

```
mov     al,0Ch
mov     bx,1230h
xlat
```

если в памяти по адресу DS:BX были записаны символы "0123456789ABCDEF", то теперь AL содержит не число 0Ch, а ASCII-код буквы «C».

LEA приемник, источник

Вычисление эффективного адреса

Вычисляет эффективный адрес источника (переменная) и помещает его в приемник (регистр). С помощью LEA можно вычислить адрес переменной, которая описана сложным методом адресации, например по базе с индексированием.

Команду LEA часто используют для быстрых арифметических вычислений, например сложения:

```
lea     bx,[bx]+12  ; BX = BX + 12
```

или трехоперандного сложения:

```
lea     ax,[bx+si]  ; AX = BX + SI
```

IN приемник, источник

Считать данные из порта

В приемник (регистр - только AL или AX) считывает байт или слово из порта, номер которого находится в источнике (непосредственный операнд от 0 до 255). В качестве номера порта может быть указано только одно из чисел: 0 (порт LPT 378h), 1 (порт LPT 379h), 2, 3 или 4 (текстовые поля, располагающиеся на вкладке Ввод-Вывод). Команда

для LPT порта не работает в системах семейства NT в связи с тем, что в них прямое обращение к портам запрещено.
Допустимы только следующие варианты:

```
in    al,0  ; байт из LPT 378h
in    al,1  ; байт из LPT 379h
in    al,2  ; байт из поля "Порт #2"
in    al,3  ; байт из поля "Порт #3"
in    al,4  ; байт из поля "Порт #4"
in    ax,2  ; слово из поля "Порт #2"
in    ax,3  ; слово из поля "Порт #3"
in    ax,4  ; слово из поля "Порт #4"
```

OUT приемник, источник
Записать данные в порт
Записывает байт или слово из источника (регистр AL или AX соответственно) в порт, номер которого находится в приемнике (непосредственный операнд от 0 до 255). В качестве номера порта может быть указано только одно из чисел: 0 (порт LPT 378h), 2, 3 или 4 (текстовые поля, располагающиеся на вкладке Ввод-Вывод). (Порт LPT 379h существует только для чтения). Команда для LPT порта не работает в системах семейства NT в связи с тем, что в них прямое обращение к портам запрещено. Для Порта #4 учитываются только два младших бита (см. раздел Ввод-Вывод).
Допустимы только следующие варианты:

```
out   0,al  ; байт в LPT 378h
out   2,al  ; байт в поле "Порт #2"
out   3,al  ; байт в поле "Порт #3"
out   4,al  ; байт в поле "Порт #4"
out   2,ax  ; слово в поле "Порт #2"
out   3,ax  ; слово в поле "Порт #3"
out   4,ax  ; слово в поле "Порт #4"
```

Двоичная арифметика
Все команды из этого раздела, кроме команд деления и умножения, изменяют флаги OF, SF, ZF, AF, CF, PF в соответствии с назначением каждого из этих флагов.
ADD приемник, источник
Сложение
Команда выполняет арифметическое сложение приемника и источника, помещает сумму в приемник, не изменяя содержимое источника. Приемник может быть регистром или памятью, источник может быть числом, регистром или переменной, но нельзя использовать переменную одновременно и для источника, и для приемника. Команда ADD никак не различает числа со знаком и без знака, но, употребляя значения флагов CF (перенос при сложении чисел без знака), OF (перенос при сложении чисел со знаком) и SF (знак результата), можно использовать ее и для тех, и для других.
Команда ADD аналогична команде на языке Pascal:

```
приемник := приемник + источник;
```

ADC приемник, источник
Сложение с переносом
Эта команда во всем аналогична ADD, кроме того, что она выполняет арифметическое сложение приемника, источника и флага CF. Пара команд ADD/ADC используется для сложения чисел повышенной точности. Сложим, например, два 32-битных целых числа: пусть одно из них находится в паре регистров DX:AX (младшее слово (биты 0–15) — в AX и старшее (биты 16 – 31) — в DX), а другое — в паре регистров BX:CX:

```
add    ax,cx
adc    dx,bx
```

Если при сложении младших слов произошел перенос из старшего разряда (флаг CF=1), то он будет учтен следующей командой ADC.

SUB приемник, источник

Вычитание

Вычитает источник из приемника и помещает разность в приемник. Приемник может быть регистром или памятью, источник может быть числом, регистром или памятью, но нельзя использовать память одновременно и для источника, и для приемника. Точно так же, как и команда ADD, SUB не делает различий между числами со знаком и без знака, но флаги позволяют использовать ее как для тех, так и для других. Команда SUB аналогична команде на языке Pascal:

```
приемник := приемник - источник;
```

SBB приемник, источник

Вычитание с займом

Эта команда во всем аналогична SUB, кроме того, что она вычитает из приемника значение источника и дополнительно вычитает значение флага CF. Так, можно использовать эту команду для вычитания 32-битных чисел в DX:AX и BX:CX аналогично ADD/ADC:

```
sub     ax,cx
sbb     dx,bx
```

Если при вычитании младших слов произошел заем, то он будет учтен при вычитании старших.

IMUL источник

Умножение чисел со знаком

Источник (регистр или память) умножается на AL или AX (в зависимости от размера операнда), и результат располагается в AX или DX:AX соответственно. Считается, что результат может занимать в два раза больше места, чем размер источника. Приемник автоматически оказывается достаточно большим. Флаги OF и CF будут равны нулю, если результат умножения поместился целиком в младшую половину приемника, и единице в противном случае. Значения флагов SF, ZF, AF и PF после команды IMUL не определены.

MUL источник

Умножение чисел без знака

Выполняет умножение содержимого источника (регистр или память) и регистра AL, AX (зависимости от размера источника) и помещает результат в AX, DX:AX соответственно. Если старшая половина результата (AH, DX) содержит только нули (результат целиком поместился в младшую половину), флаги CF и OF устанавливаются в 0, иначе — в 1. Значение остальных флагов (SF, ZF, AF и PF) не определено.

IDIV источник

Целочисленное деление со знаком

Выполняет целочисленное деление со знаком AX (если источник — байт) или DX:AX (если источник — слово) на источник (регистр или память) и помещает результат в AL или AX, а остаток — в AH или DX соответственно. Результат всегда округляется в сторону нуля, знак остатка всегда совпадает со знаком делимого (источника), абсолютное значение остатка всегда меньше абсолютного значения делителя (источника). Значения флагов CF, OF, SF, ZF, AF и PF после этой команды не определены. Переполнение или деление на ноль вызывает ошибку выполнения (ошибка при делении).

DIV источник

Целочисленное деление без знака

Выполняет целочисленное деление без знака AX (если источник — байт) или DX:AX (если источник — слово) на источник (регистр или память) и помещает результат в AL или AX, а остаток — в AH или DX соответственно. Результат всегда округляется в сторону нуля, абсолютное значение остатка всегда меньше абсолютного значения делителя. Значения флагов CF, OF, SF, ZF, AF и PF после этой команды не

определены. Переполнение или деление на ноль вызывает ошибку выполнения (ошибка при делении).

INC приемник

Инкремент

Увеличивает приемник (регистр или память) на 1. Единственное отличие этой команды от ADD приемник,1 состоит в том, что флаг CF не затрагивается. Остальные арифметические флаги (OF, SF, ZF, AF, PF) устанавливаются в соответствии с результатом сложения.

DEC приемник

Декремент

Уменьшает приемник (регистр или память) на 1. Единственное отличие этой команды от SUB приемник,1 состоит в том, что флаг CF не затрагивается. Остальные арифметические флаги (OF, SF, ZF, AF, PF) устанавливаются в соответствии с результатом вычитания.

NEG приемник

Изменение знака

Выполняет над числом, содержащимся в приемнике (регистр или память), операцию дополнения до двух. Эта операция эквивалентна обращению знака операнда, если рассматривать его как число со знаком. Если приемник равен нулю, флаг CF устанавливается в 0, иначе — в 1. Остальные флаги (OF, SF, ZF, AF, PF) устанавливаются в соответствии с результатом операции.

CMP приемник, источник

Сравнение

Сравнивает приемник и источник и устанавливает флаги. Сравнение осуществляется путем вычитания источника (число, регистр или память) из приемника (регистр или память; приемник и источник не могут быть памятью одновременно), причем результат вычитания никуда не записывается, единственным результатом работы этой команды оказывается изменение флагов CF, OF, SF, ZF, AF и PF. Обычно команду CMP используют вместе с командами условного перехода (Jcc), которые позволяют использовать результат сравнения, не обращая внимания на детальное значение каждого флага. Так, команда JE выполнят переход, если значения операндов предшествующей команды CMP были равны.

Логические операции

AND приемник, источник

Логическое И

Команда выполняет побитовое «логическое И» над приемником (регистр или память) и источником (число, регистр или память; источник и приемник не могут быть памятью одновременно) и помещает результат в приемник. Любой бит результата равен 1, только если соответствующие биты обоих операндов были равны 1, и равен 0 в остальных случаях. Наиболее часто AND применяют для выборочного обнуления отдельных бит, например, команда

```
and  al,0Fh ;00001111b
```

обнулит старшие четыре бита регистра AL, сохранив неизменными четыре младших. Флаги OF и CF обнуляются, SF, ZF и PF устанавливаются в соответствии с результатом, AF не определен.

OR приемник, источник

Логическое ИЛИ

Выполняет побитовое «логическое ИЛИ» над приемником (регистр или память) и источником (число, регистр или память; источник и приемник не могут быть памятью одновременно) и помещает результат в приемник. Любой бит результата равен 0, только если соответствующие биты обоих операндов были равны 0, и равен 1 в остальных случаях. Команду OR чаще всего используют для выборочной установки отдельных бит, например, команда

```
or    al,0Fh  ;00001111b
```
приведет к тому, что младшие четыре бита регистра AL будут установлены в 1. При выполнении команды OR флаги OF и CF обнуляются, SF, ZF и PF устанавливаются в соответствии с результатом, AF не определен.

XOR приемник, источник

Логическое исключающее ИЛИ

Выполняет побитовое «логическое исключающее ИЛИ» над приемником (регистр или память) и источником (число, регистр или память; источник и приемник не могут быть памятью одновременно) и помещает результат в приемник. Любой бит результата равен 1, если соответствующие биты операндов различны, и нулю, если одинаковы. XOR используется для самых разных операций, например:

```
xor    ax,ax   ; обнуление регистра AX
```
или
```
xor    ax,bx
xor    bx,ax
xor    ax,bx   ; меняет местами содержимое AX и BX
```
Оба этих примера могут выполняться быстрее, чем соответствующие очевидные команды
```
mov    ax,0
```
или
```
xchg   ax,bx
```
NOT приемник

Логическое НЕ

Каждый бит приемника (регистр или память), равный нулю, устанавливается в 1, и каждый бит, равный 1, сбрасывается в 0. Флаги не затрагиваются.

TEST приемник, источник

Логическое сравнение

Вычисляет результат действия побитового «логического И» над приемником (регистр или память) и источником (число, регистр или памятью; источник и приемник не могут быть памятью одновременно) и устанавливает флаги SF, ZF и PF в соответствии с полученным результатом, не сохраняя результат (флаги OF и CF обнуляются, значение AF не определено). TEST, так же как и CMP, используется в основном в сочетании с командами условного перехода (Jcc).

Сдвиговые операции

SHL приемник, счетчик

Логический сдвиг влево

SHR приемник, счетчик

Логический сдвиг вправо

Эти две команды выполняют двоичный сдвиг приемника (регистр или переменная) вправо (в сторону младшего бита) или влево (в сторону старшего бита) на значение счетчика (число или регистр CL (в данной программе можно указывать любой восьмибитный регистр, но при компиляции все равно подставляется CL), из которого учитываются только младшие три или четыре бит для байта и слова соответственно, которые могут принимать значения от 0 до 7 или до 15 соответственно). Операция сдвига на 1 эквивалентна умножению (сдвиг влево) или делению (сдвиг вправо) на 2. Так, число 0010b (2) после сдвига на 1 влево превращается в 0100b (4). Команда SHL на каждый шаг сдвига старший бит заносится в CF, все биты сдвигаются влево на одну позицию, и младший бит обнуляется. Команда SHR выполняет прямо противоположную операцию: младший бит заносится в CF, все биты сдвигаются на 1 вправо, старший бит обнуляется. Эта команда эквивалентна беззнаковому целочисленному делению на 2. Сдвиги больше чем на 1 эквивалентны соответствующим сдвигам на 1, выполненным последовательно.

Сдвиги на 1 изменяют значение флага OF: SHL устанавливают его в 1, если после сдвига старший бит изменился (то есть старшие два бита исходного числа не были одинаковыми), и в 0, если старший бит остался тем же. SHR устанавливает OF в значение старшего бита исходного числа. Для сдвигов на несколько бит значение OF не определено. Флаги SF, ZF, PF устанавливаются всеми сдвигами в соответствии с результатом, значение AF не определено.
В процессорах 8086 непосредственно можно было задавать в качестве второго операнда только число 1 и при использовании CL учитывать все биты, а не только младшие 5, но уже начиная с 80186 эти команды приняли свой окончательный вид.

Команды передачи управления

JMP метка

Безусловный переход

Передает управление в другую точку программы, не сохраняя какой-либо информации для возврата. Операндом может быть непосредственный адрес для перехода (в программах используют имя метки, установленной перед командой, на которую выполняется переход). Переход может быто только типа near (ближний переход) – если адрес перехода находится в пределах от -32768 до +32767 байт от команды JMP.
При выполнении перехода типа short и near команды перехода фактически изменяют значение регистра IP, изменяя тем самым смещение следующей исполняемой команды относительно начала сегмента кода. Если операнд для них – непосредственно указанное число (в программах имя метка), то его значение суммируется с содержимым IP, приводя к относительному переходу. В ассемблерных программах в качестве операнда указывают имена меток, но на уровне исполнимого кода ассемблер вычисляет и записывает именно относительные смещения.

Jcc метка

Условный переход

Это набор команд, каждая из которых выполняет переход (типа short), если удовлетворяется соответствующее условие. Условием в каждом случае реально является состояние тех или иных флагов, но, если команда из набора Jcc используется сразу после CMP, условия приобретают формулировки, соответствующие отношениям между операндами CMP (см. таблицу). Например, если операнды CMP были равны, то команда JE, выполненная сразу после этого CMP, осуществит переход. Операнд для всех команд из набора Jcc — 8-битное смешение относительно текущей команды.
Слова «выше» и «ниже» в таблице относятся к сравнению чисел без знака, а слова «больше» и «меньше» учитывают знак.

Команда	Реальное условие	Условие для CMP
JA JNBE	CF = 0 и ZF = 0	если выше если не ниже или равно
JAE JNB JNC	CF = 0	если выше или равно если не ниже если нет переноса
JB JNAE JC	CF = 1	если ниже если не выше или равно если перенос
JBE JNA	CF = 1 или ZF = 1	если ниже или равно если не выше
JE JZ	ZF = 1	если равно если ноль
JG JNLE	ZF = 0 и SF = OF	если больше если не меньше или равно
JGE	SF = OF	если больше или равно

JNL		если не меньше
JL JNGE	SF <> OF	если меньше если не больше или равно
JLE JNG	ZF = 1 или SF <> OF	если меньше или равно если не больше
JNE JNZ	ZF = 0	если не равно если не ноль
JNO	OF = 0	если нет переполнения
JO	OF = 1	если есть переполнение
JNP JPO	PF = 0	если нет четности если нечетное
JP JPE	PF = 1	если есть четность если четное
JNS	SF = 0	если нет знака
JS	SF = 1	если есть знак

LOOP метка
Цикл
Уменьшает регистр CX на 1 и выполняет переход типа short на метку (которая не может быть дальше, чем на расстоянии от -128 до +127 байт от команды LOOP), если CX не равен нулю. Эта команда используется для организации циклов, в которых регистр CX играет роль счетчика. Так, в следующем фрагменте команда ADD выполнится 10 раз:

```
        mov     cx,0Ah
loop_start: add     ax,cx
        loop    loop_start
```

Команда LOOP полностью эквивалентна паре команд

```
        dec     cx
        jnz     метка
```

Но LOOP короче этих двух команд на один байт и не изменяет значения флагов.
LOOPE метка
Цикл, пока равно
LOOPZ метка
Цикл, пока ноль
LOOPNE метка
Цикл, пока не равно
LOOPNZ метка
Цикл, пока не ноль
Все эти команды уменьшают регистр CX на один, после чего выполняют переход типа short, если CX не равен нулю и если выполняется условие. Для команд LOOPE и LOOPZ условием является равенство единице флага ZF, для команд LOOPNE и LOOPNZ — равенство флага ZF нулю. Сами команды LOOPcc не изменяют значений флагов, так что ZF должен быть установлен (или сброшен) предшествующей командой.
CALL метка
Вызов процедуры
Сохраняет текущий адрес в стеке и передает управление по адресу, указанному в операнде. Операндом может быть непосредственное значение адреса (метка в ассемблерных программах). При этом, так же как и в случае с JMP, выполняется ближний вызов процедуры. Процессор помещает значение регистра IP, соответствующее следующей за CALL команде, в стек и загружает в IP новое значение,

осуществляя тем самым передачу управления. Если операнд – метка в программе, то ассемблер указывает ее относительное смещение.

RET число

Возврат из процедуры

RET (RETN для ближнего перехода) считывает из стека слово и загружает его в IP, выполняя тем самым действия, обратные ближнему вызову процедуры командой, CALL. Операнд для RET необязателен, но, если он присутствует, после считывания адреса возврата из стека будет удалено указанное количество байт — это бывает нужно, если при вызове процедуры ей передавались параметры через стек. В реальных ассемблерных программах часто используют команду RET для завершения работы программы и возврата с DOS. Ассемблер же при компиляции подставляет вместо нее RETF, что соответствует возврату из дальней процедуры – из стека загружается не только старое (то, которое было до запуска программы) значение IP, а еще и старое значение CS. Поэтому здесь не следует пытаться завершить программу командой RET — она просто загрузит из стека значение для возврата. Чтобы остановить выполнение программы, используйте команду HLT.

NOP

Отсутствие операции

NOP – однобайтная команда (код 90h), которая не выполняет ничего, только занимает место и время. Можно многие команды записать так, что они не будут приводить ни к каким действиям, например:

```
mov    ax,ax   ; 2 байта
xchg   ax,ax   ; 2 байта
```

HLT

Остановить процессор

Используется для прекращения выполнения программы. Если эта команда отсутствует, то процессор будет выполнять программу, а после ее окончания в памяти будет выполнять поледующие инструкции (могут быть абсолютно любые данные, например, от предыдущих программ), что может привести к непредсказуемому результату.

Строковые операции

Все команды для работы со строками считают, что строка-источник находится по адресу DS:SI (то есть в сегменте памяти, указанном в DS со смещением в SI), а строка-приемник — в ES:DI. Кроме того, все строковые команды работают только с одним элементом строки (байтом или словом) за один раз. Для того чтобы команда выполнялась над всей строкой, необходим один из префиксов повторения операций.

REP

Повторять

REPE

Повторять, пока равно

REPZ

Повторять, пока ноль

REPNE

Повторять, пока не равно

REPNZ

Повторять, пока не ноль

Все эти команды — префиксы для операций над строками. Любой из префиксов выполняет следующую за ним команду строковой обработки столько раз, сколько указано в регистре CX, уменьшая его при каждом выполнении команды на 1. Кроме того, префиксы REPZ и REPE прекращают повторения команды, если флаг ZF сброшен в 0, и префиксы REPNZ и REPNE прекращают повторения, если флаг ZF установлен в 1. Префикс REP обычно используется с командами MOVS, LODS и STOS, а префиксы REPE, REPNE, REPZ и REPNZ — с командами CMPS и SCAS. Поведение префиксов не с командами строковой обработки не определено. На самом деле префиксы REP,

REPE и REPZ имеют один и тот же код F3h, а префиксы REPNE и REPNZ — F2h. Разные команды используют их тем или иным способом. Команды копирования строк MOVS, LODS и STOS не могут повторяться с префиксами REPNE и REPNZ, а префиксы REPE и REPZ ведут себя как REP, т.е повторяют команду число раз, указанное в CX. Команды сравнения строк CMPS и SCAS с префиксом REP выполняются так же, как и с префиксами REPE и REPZ — пока не встретятся различные значения. В данной программе эти префиксы необходимо записывать перед нужной командой на отдельной строке, как самостоятельную команду.

```
mov    al,20h     ; символ пробела в AL
mov    cx,8       ; восемь раз
rep               ;  повторять
stosb             ;  копирование AL в ES:DI
```

MOVSB

Копирование строки байт

MOVSW

Копирование строки слов

Копирует один байт (MOVSB) или слово (MOVSW) из памяти по адресу DS:SI в память по адресу ES:DI. После выполнения команды регистры SI и DI увеличиваются на 1 или 2 (если копируются байты или слова), если флаг DF = 0, и уменьшаются, если DF = 1. При использовании с префиксом REP команда MOVS выполняет копирование строки длиной в CX байт или слов.

CMPSB

Сравнение строк байт

CMPSW

Сравнение строк слов

Сравнивает один байт (CMPSB) или слово (CMPSW) из памяти по адресу DS:SI с байтом или словом по адресу ES:DI и устанавливает флаги аналогично команде CMP. После выполнения команды регистры SI и DI увеличиваются на 1 или 2 (если сравниваются байты или слова), если флаг DF = 0, и уменьшаются, если DF = 1. При использовании с префиксами REPNE/REPNZ или REPE/REPZ команда CMPS выполняет сканирование строки длиной в CX байт или слов, в первом случае до первого совпадения в сравниваемых строках, а во втором — до первого несовпадения.

SCASB

Сканирование строки байт

SCASW

Сканирование строки слов

Сравнивает содержимое регистра AL (SCASB) или AX (SCASW) с байтом или словом из памяти по адресу ES:DI и устанавливает флаги аналогично команде CMP. После выполнения команды регистр DI увеличивается на 1 или 2 (если сканируются байты или слова), если флаг DF = 0, и уменьшается, если DF = 1. При использовании с префиксами REPNE/REPNZ или REPE/REPZ команда SCAS выполняет сканирование строки длиной в CX байт или слов, в первом случае до первого совпадения в сравниваемой строке, а во втором — до первого несовпадения.

LODSB

Чтение байта из строки

LODSW

Чтение слова из строки

Копирует один байт (LODSB) или слово (LODSW) из памяти по адресу DS:SI в регистр AL или AX. После выполнения команды регистр SI увеличивается на 1 или 2 (если считывается байт или слово), если флаг DF = 0, и уменьшается, если DF = 1. При использовании с префиксом REP команда LODS выполнит копирование строки длиной в CX байт или слов, что приведет к тому, что в аккумуляторе окажется последний элемент строки. На самом деле эту команду используют без префиксов, часто внутри

цикла в паре с командой STOS, так что LODS считывает число, другие команды выполняют над ним какие-нибудь действия, а затем STOS записывает измененное число в память.

STOSB

Запись байта в строку

STOSW

Запись слова в строку

Копирует регистр AL (STOSB) или AX (STOSW) в память по адресу ES:DI. После выполнения команды регистр DI увеличивается на 1 или 2 (если копируется байт или слово), если флаг DF = 0, и уменьшается, если DF = 1. При использовании с префиксом REP команда STOS заполнит строку длиной в CX числом, находящимся в аккумуляторе.

Управление флагами

STC

Установить флаг переноса

Устанавливает флаг CF в 1.

CLC

Сбросить флаг переноса

Сбрасывает флаг CF в 0.

CMC

Инвертировать флаг переноса

Инвертирует флаг CF.

SALC

Установить AL в соответствии с CF

Устанавливает AL в 0FFh, если флаг CF = 1, и сбрасывает в 00h, если CF = 0.

STD

Установить флаг направления

Устанавливает флаг DF в 1.

CLD

Сбросить флаг направления

Сбрасывает флаг DF в 0.