МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО
СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ
РЕСПУБЛИКИ УЗБЕКИСТАН

М.М. АЛИЕВ

ЦИФРОВАЯ ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И МИКРОПРОЦЕССОРЫ

Утверждено Министерством высшего и среднего специального образования Республики Узбекистан в качестве учебного пособия для студентов высших технических учебных заведений

Ташкент – 2009

Алиев М.М. Цифровая вычислительная техника и микропроцессоры. Т., Изд-во «Fan va texnologiya», 2009. 160 стр.

Рассмотрены принципы построения цифровой вычислительной техники, архитектурные особенности микропроцессоров, программные ресурсы, система команд, а также вопросы программирования на языке ассемблера. Приведены многочисленные примеры выполнения команд и программ задач.

Для студентов технических вузов и инженерно-технических работников в области вычислительной техники, автоматики и связи, занятых разработкой и применением микропроцессорных систем.

Рецензенты: **И.АЛИМОВ** - д.т.н., профессор, ведущий научный сотрудник института «Информатика»;
Б.САЛИХОВ - начальник Центра статистики и вычислительной техники ГАЖК «Ўзбекистон темир йўллари»

ISBN 978-9943-10-206-4

ВВЕДЕНИЕ

В основных направлениях экономического и социального развития Узбекистана на период до 2010 года предусмотрено «Последовательно повышать организационную и технологическую гибкость производства. Внедрять автоматизированные системы в различные сферы хозяйственной деятельности, и в первую очередь в проектирование, управление оборудованием и технологическими процессорами. Поднять уровень автоматизации производства примерно в 2 раза. Создавать комплексно-автоматизированные производства, которые можно быстро и экономично перестраивать».

Все эти задачи будут решаться применением в отраслях народного хозяйства вычислительной техники и в первую очередь микропроцессорной техники, построенной на больших интегральных схемах.

Микропроцессоры (МП) и микро-ЭВМ благодаря реализованной в них возможности программного управления обладают свойствами универсальных устройств и позволяют применять средства и методы цифровой обработки данных и цифрового управления в таких областях техники и народного хозяйства, в которых ранее их использование было экономически неоправданным.

Постоянно возрастающий объем производства МП и микро-ЭВМ, улучшение их технических характеристик и снижение стоимости дают возможность утверждать, что в ближайшие годы МП и микро-ЭВМ будут очень широко использоваться в устройствах и системах автоматики, телемеханики и связи. Опыт применения МП и микро-ЭВМ в локальных устройствах автоматики связи и системах управления объектами и технологическими процессорами свидетельствует о том, что использование МП дает возможность создать системы автоматики с качественно новыми интеллектуальными свойствами, что обеспечивает достижение исключительно высоких технико-экономических показателей и расширение функциональных возможностей этих устройств и систем.

В 1976 г. на вопрос анкеты «Каково Ваше личное мнение о микропроцессоре» предложенный американским журналом «Инструментейшн технолоджи» своим читателям - пользователя и разработчикам средств промышленной автоматики, ответы распределились следующим образом «53% - «Невероятно, сказочно!; 27% - Микропроцессоры найдут применение в качестве элементов управления и 20% - Мнение не оформилось окончательно, но первое время работать с ними будет трудно».

В настоящее время, если бы был задан такой же вопрос все 100% ответили бы утвердительно, что не только целесообразно, но и необходимо применять МП. То есть внедрение миниатюрных электронных управляющих машин, микропроцессоров и промышленных роботов открывает безграничные возможности в системах железнодорожной автоматики, телемеханики и связи.

Еще в первой половине XIX в. английский математик Чарльз Бэббидж попытался построить универсально вычислительное устройство, то есть компьютер, (Бэббидж называл его Аналитической машиной). Именно Бэббидж впервые додумался до того, что компьютер должен содержать память и управляться с помощью программы. Бэббидж хотел построить свой компьютер как механическое устройство, а программ собирался задавать посредством, перфокарт — карт из плотной бумаги с формацией, наносимой с помощью отверстий (они в то время уже широко употреблялись в ткацких станках). Однако довести до конца эту работу Бэббидж не смог — она оказалась слишком сложной для техники того времени.

Первые компьютеры. В 40-ходах XX в. сразу несколько групп исследователей повторили попытку Бэббиджа на основе техники XX в. — электромеханических реле. Некоторые из этих исследователей ничего не знали о работах Бэббиджа и переоткрыли его идеи заново. Первым из них был немецкий инженер Конрад Цузе, который в 1941 г. построил небольшой компьютер на основе нескольких электромеханических реле. Но из-за войны работы Цузе не были опубликованы. А в США в 1943 г. на одном из предприятий фирмы IBM американец Говард Эйкен создал более мощный компьютер под названием «Марк-1». Он уже позволял проводить вычисления в сотни раз быстрее, чем вручную (с помощью арифмометра), и реально использовался для военных расчетов.

Однако электромеханические реле работают весьма медленно и недостаточно надежно. Поэтому начиная с 1943 г. в США группа специалистов под руководством Джона Мочли и Преспера Экерта начала конструировать компьютер ENIAC на основе электронных ламп. Созданный ими компьютер работал в тысячу раз быстрее, чем Марк-1. Однако обнаружилось, что большую часть времени этот компьютер простаивал — ведь для задания метода расчетов (программы) в этом компьютере приходилось в течение нескольких часов или даже нескольких дней подсоединять нужным образом провода. А сам расчет после этого мог занять всего лишь несколько минут или даже секунд.

Компьютеры с хранимой в памяти программой. Чтобы упростить и ускорить процесс задания программ, Мочли и Экерт стали конструировать новый компьютер, который мог бы *хранить программу в своей памяти.* В 1945 г. к работе был привлечен знаменитый математик Джон фон Нейман, который подготовил доклад об этом компьютере. Доклад был разослан многим ученым и получил широкую известность, поскольку в нем фон Нейман ясно и просто сформулировал общие принципы функционирования компьютеров, т.е. универсальных вычислительных устройств. И до сих пор подавляющее большинство компьютеров сделано в соответствии с теми принципами, которые изложил в своем докладе в 1945 г. Джон фон Нейман. Первый компьютер, в котором были воплощены принципы фон Неймана, был построен в 1949 г. английским исследователем Морисом Уилксом.

Развитие элементной базы компьютеров. В 40-х и 50-х годах компьютеры создавались на основе электронных ламп. Поэтому компьютеры были очень большими, и (занимали огромные залы), дорогими и ненадежными — ведь электронные лампы, как и обычные лампочки, часто перегорают. Но в 1948 г. были изобретены *транзисторы* — миниатюрные и недорогие электронные приборы, которые смогли заменить электронные лампы. Это привело к уменьшению размеров компьютеров в сотни раз и повышению их надежности. Первые компьютеры на основе транзисторов появились в конце 50-х годов, а к середине 60-х годов был созданы и значительно более компактные внешние устройства для компьютеров, что позволило фирме Digital Equipment выпустить в 1965 г. первый мини-компьютер PDP-8 размером с холодильник и

стоимостью всего 20 тыс. дол. (компьютеры 40-х и 50-х годов, обычно стоили миллионы дол.).

После появления транзисторов наиболее трудоемкой операцией при производстве компьютеров было соединение и спайка транзисторов для создания электронных схем. Но в 1959 г. Роберт Нойс (будущий основатель фирмы Intel) изобрел способ, позволяющий создавать на одной пластине кремния транзисторы и все необходимые соединения между ними. Полученные электронные схемы стали называться *интегральными схемами*, или *чипами*. В 1968 г. фирма Burroughs выпустила первый компьютер на интегральных схемах, а в 1970 г. фирма Intel начала продавать интегральные схемы памяти. В дальнейшем количество транзисторов, которое удавалось разместить на единице площади интегральной схемы, увеличивалось приблизительно вдвое каждый и год, что и обеспечивает постоянное уменьшение стоимости компьютеров и повышение быстродействия.

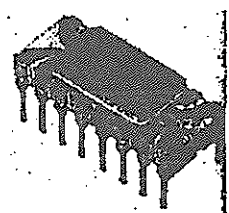


Рис. 1.

Микропроцессоры. В 1970 г. был сделан еще один важный шаг на пути к персональному компьютеру — Маршиан Эдвард Хофф из фирмы Intel сконструировал интегральную схему, аналогичную по своим функциям центральному процессору большого компьютера. Так появился первый *микропроцессор* Intel-4004 (см. рис. 1), который был выпущен, в продажу в 1971 г. Это был настоящий прорыв, ибо микропроцессор Intel-4004 размером менее 3 см был производительнее гигантской машины ENIAC. Правда, возможности Intel-4004 были куда скромнее, чем у центрального процессора больших компьютеров того времени, — он работал гораздо медленнее и мог обрабатывать одновременно только 4 бита информации, (процессоры больших компьютеров обрабатывали 16 или 32 бита одновременно), но и стоил он в десятки тысяч раз

дешевле. Но рост производительности микропроцессоров не заставил себя ждать. В 1973 г. фирма Intel выпустила 8-битовый микропроцессор Intel-8008, а в 1974 г. – его усовершенствованную версию Intel-8080, которая до конца 70-х годов стала стандартом для микрокомпьютерной индустрии.

Появление персональных компьютеров. Вначале микропроцессоры использовались в различных специализированных устройствах, например, в калькуляторах. Но в 1974 г. несколько фирм объявили о создании на основе микропроцессора Intel-8080 *персонального компьютера,* т.е. устройства, выполняющего те же функции, что и большой компьютер, но рассчитанного для иного пользователя. В начале 1975 г. появился первый коммерчески распространяемый персональный компьютер Альтаир-8800 на основе микропроцессора Intel-8080. .Этот компьютер продавался по цене около 500 дол. И хотя возможности его были весьма ограничены (оперативная память, составляла всего 256 байт, клавиатура и экран отсутствовали), его появление было встречено с большим энтузиазмом: в первые же месяцы было продано несколько тысяч комплектов машины. Покупатели снабжали этот компьютер дополнительными устройствами: монитором для вывода информации, клавиатурой, блоками расширения памяти и т.д. Вскоре эти устройства стали выпускаться другими фирмами. В конце 1975 г. Пол Аллен и Билл Гейтс (будущие основатели фирмы Microsoft) создали для компьютера «Альтаир» интерпретатор языка Basic, что позволило пользователям достаточно просто общаться с компьютером и легко писать для него программы. Это также способствовало популярности персональных компьютеров.

Успех Альтаир-8800 заставил многие фирмы также заняться производством персональных компьютеров. Персональные компьютеры стали продаваться уже в полной комплектации, с клавиатурой и монитором, спрос на них составил десятки, а затем и сотни тысяч штук в год. Появилось несколько журналов, посвященных персональным компьютерам. Росту объема продаж весьма способствовали многочисленные полезные программы, разработанных для деловых применений. Появились и коммерчески распространяемые программы, например, программа для редактирования текстов WordStar и табличный процессор VisiCalc (соответственно 1978 и 1979 гг.). Эти (и многие другие) программы сделали

покупку персональных компьютеров весьма выгодным для бизнеса: с их помощью стало возможно выполнять бухгалтерские расчеты, составлять документы и т.д. Использование же больших компьютеров для этих целей было слишком дорого.

Появление IBM PC. В конце 70-х годов распространение персональных компьютеров даже привело к некоторому снижению спроса на большие компьютеры и мини-компьютеры (мини-ЭВМ). Это стало предметом серьезного беспокойства фирмы IBM (International Business Machines Corporation) — ведущей компании по производству больших компьютеров, и в 1979 г. фирма IBM решила попробовать свои силы на рынке персональных компьютеров. Однако руководство фирмы недооценило будущую важность этого рынка и рассматривало создание персонального компьютера всего лишь как мелкий эксперимент — что-то вроде одной из десятков проводившихся в фирме работ по созданию нового оборудования. Чтобы не тратить на этот эксперимент слишком много денег, руководство фирмы предоставило подразделению, ответственному за данный проект, невиданную в фирме свободу. В частности, ему было разрешено не конструировать персональный компьютер «с нуля»; а использовать блоки, изготовленные другими фирмами. И это подразделение сполна использовало предоставленный шанс.

Прежде всего, в качестве основного микропроцессора компьютера был выбран новейший тогда 16-разрядный микропроцессор Intel-8088. Его использование позволило значительно увеличить потенциальные возможности компьютера, так как новый микропроцессор позволял работать с 1 Мбайтом памяти, а все имевшиеся тогда компьютеры были ограничены 64 Кбайтами. В компьютере были использованы и другие комплектующие различных фирм, а его программное обеспечение было поручено разработать небольшой фирме Microsoft.

В августе 1981 г. новый компьютер под названием IBM PC (читается — Ай-Би-Эм Пи-Си) был официально представлен публике и вскоре после этого он приобрел большую популярность у пользователей. Через один-два года компьютер IBM PC занял ведущее место на рынке, вытеснив модели 8-битовых компьютеров.

В настоящее время IBM PC-совместимые компьютеры превратились в мощные высокопроизводительные устройства.

По всем основным показателям (быстродействие, емкость оперативной и дисковой памяти и др.) они в сотни раз превосходят первоначальную модель IBM PC, а стоят обычно даже дешевле. ЕСЛИ бы такими же темпами развивалось, скажем, автомобилестроение, то сейчас за несколько тысяч долларов предлагались бы автомобили, передвигающиеся со скоростью космических ракет и вмещающие сотни человек. В мире ежегодно производится несколько десятков миллионов IBM PC совместимых компьютеров, это более чем 90% всех производимых в мире компьютеров.

Производством компонент и устройств для IBM PC-совместимых компьютеров занимаются тысячи фирм; в числе которых — сотни гигантских международных корпораций, в том числе таких известных, как Intel, Toshiba, Fujitsu, Siemens, Hitachi, Hewlett-Packat, Phillips, Samsung, Goldstar и др. Конкуренция в этой области острейшая, результате цены на комплектующие и внешние устройства постоянно падают, а характеристики их — улучшаются. Иногда изменения на рынке бывают просто невероятными, так, начиная с лета 1995 за год цены на микросхемы оперативной памяти подешевели более чем в пять раз!

Особую роль среди производителей компонент играет фирма Intel — лидер в области разработки и производства микропроцессоров и материнских плат для IBM PC-совместимых компьютеров. Именно разработки фирмы Intel в значительной степени определяют прогресс компьютерной индустрии. Но и Intel не является абсолютным монополистом — с ним конкурируют фирмы AMD, Cyrix, IBM и др.

ГЛАВА 1. МИКРОПРОЦЕССОРЫ И МИКРО-ЭВМ. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

1.1. Классификация

Электронно-вычислительная машина (ЭВМ) представляет собой устройство, предназначенное для выполнения вычислительных операций и основным устройством ЭВМ является процессор (вычислитель).

Последним достижением электронно-вычислительной техники являются БИС и построенные на их основе микропроцессоры – устройства, которые представляют собой одни или несколько крохотных кусочков кремния с функциональной плотностью 35 млн. элементов на кристалл и обладающие способностью выполнять под программным управлением обработку информации, включая ввод и вывод информации принятие решений, арифметические и логические операции.

В общем случае в состав МП входят: арифметическо-логическое устройство (АЛУ); схема управления и синхронизация; регистр – аккумулятор; сверхоперативное запоминающее устройство (СОЗУ), программный счетчик; адресный стек; регистр команд и дешифратор кода операции; параллельные шины данных и ввода-вывода; схема управления памятью и вводом-выводом.

Микро-ЭВМ – это вычислительная или управляющая система выполнения на основе МП, в состав которой могут также входить: программная память (обычно-постоянное запоминающее устройство (ПЗУ)); память данных (обычно-оперативное запоминающее устройство (ОЗУ)); устройства ввода-вывода (УВВ); генератор тактовых сигналов, а также другие устройства, выполненные с использованием БИС или элементов с меньшей степенью интеграции.

Если первый МП появился в 1971 г. (Intel 4004), то в настоящее время уже их насчитывают сотнями различных типов (табл.1).

Таблица №1.

Технические характеристики микропроцессоров
Процессоры 70-х годов

	4004	8008	8080	8086	8088
Объявлено о выпуске	15/11/71	1/4/72	1/4/74	8/6/78	1/6/79
Тактовая частота	108 КГц	108 КГц	2 МГц	5 МГц, 8 МГц, 10 МГц	5 МГц, 8 МГц
Разрядность шины	4 бит	8 бит	8 бит	16 бит	8 бит
Количество транзисторов	2 300 (10 микрон)	3 500	6 000 (6 микрон)	29 000 (3 микрона)	29 000 (3 микрона)
Адресуемая память	640 байт	16 Кбайт	64 Кбайт	1 Мб	1 Мб
Виртуальная память	--	--	--	--	--
Краткая характеристика	Первая микросхема, выполняющая арифметичес-кие вычисления	Обработка цифровых и текстовых данных	10-ратный рост производительности по сравнению с процессором 8008	10-кратный рост производительности по сравнению с процессором 8080	Аналог процессора 8086, но с 8-разрядной внешней шиной

Процессоры 80-х годов

	80286	Микропроцессор Intel386™ DX	Микропроцессор Intel386™ SX	Центральный процессор Intel486™ DX
Объявлено о выпуске	1/2/82	17/10/85	16/6/88	10/4/89
Тактовая частота	6 МГц, 8 МГц, 10 МГц, 12.5 МГц	16 МГц, 20 МГц, 25 МГц, 33 МГц	16 МГц, 20 МГц, 25МГц, 33 МГц	25 МГц, 33 МГц, 50 МГц
Разрядность шины	16 бит	32 бит	16 бит	32 бит
Количество транзисторов	134 000 (1,5 микрон)	275 000 (1 микрон)	275 000 (1 микрон)	1,2 миллиона (1 микрон) (0,8 микрона на частоте 50 МГц)
Адресуемая память	16 мегабайт	4 гигабайт	4 гигабайт	4 гигабайт
Виртуальная память	1 гигабайт	64 терабайт	64 терабайт	64 терабайт
Краткая характеристика	Рост производительности в 3-6 раз по сравнению с процессором 8086	Первая микросхема архитектуры X86, способная обрабатывать 32-разрядные наборы данных	Недорогое устройство с возможностью 32-разрядной обработки данных благодаря 16-битной адресной шине	Встроенная кэш-память 1-го уровня

Процессоры 90-х годов

	Микропроцессор Intel486™ SX	Процессор Pentium®	Процессор Pentium® Pro	Процессор Pentium® II
Объявлено о выпуске	22/4/91	22/3/93	01/11/95	07/5/97
Тактовая частота	16 МГц, 20 МГц, 25 МГц, 33 МГц	60 МГц, 66 МГц, 75 МГц, 90 МГц, 100 МГц, 120 МГц, 133 МГц, 150 МГц, 166 МГц	150 МГц, 166 МГц, 180 МГц, 200 МГц	200 МГц, 233 МГц, 266 МГц, 300 МГц
Разрядность шины	32 бит	32 бит	64 бит	64 бит
Количество транзисторов	1,185 миллиона (1 микрон)	3,1 миллиона (0,8 микрон)	5,5 миллионов (0,6 микрон)	7,5 миллионов
Адресуемая память	4 гигабайт	4 гигабайт	64 гигабайт	64 гигабайт
Виртуальная память	64 терабайт	64 терабайт	64 терабайт	64 терабайт
Краткая характеристика	Конструктивный аналог Intel486™ DX, но без математиче-	Пятикратный рост производительности по сравнению с про-	Высокопроизводительный процессор с примене-	Двойная независимая шина, динамическое исполнение,

	ского сопроцессора	цессором Intel486™ DX 33-МГц благодаря применению суперскаляр-ной архитектуры	нием архитектуры динамического исполнения	технология Intel MMX™

Процессоры конца 90-х годов

	Процессор Intel® Celeron™ (400, 366 МГц)	Процессор Pentium ® II Xeon ™ (450 МГц)	Процессор Intel® Celeron™	Процессор Pentium® III	Процессор Pentium® III Xeon™
Объявлено о выпуске	04/1/99	05/1/99	25/1/99	26/2/99	17/3/99
Тактовая частота	400, 366 МГц;	450 МГц	266 и 300 МГц;	450, 500 МГц	500, 550 МГц
Разрядность шины	64 бит			64 бит	64 бит
Количество транзис-торов	19 млн (0.25-мкм)	7.5 млн	18.9 млн (0.25-мкм)	9.5 млн (0.25-мкм)	9.5 млн (0.25-мкм)
Адресуемая память	4 гигабайт	64 гигабайт		64 гигабайт	64 гигабайт
Виртуальная память		64 терабайт			

Процессоры 2000-х годов

	INTEL Pentium III	INTEL Pentium III	INTEL Pentium IV	INTEL Xeon
Объявлено о выпуске	18/01/2000 19/03/2001 21/05/2001 30/07/2001 13/11/2001	21/03/2001 24/08/2001	27/08/2001	25/09/2001
Тактовая частота	500МГц-1.13ГГц	700-933МГц	1,4-2ГГц	2;1.7;1.5и1, 4ГГц
Разрядность шины	64 бит	64 бит	64 бит	64 бит
Количество транзисторов	0,13 Микр.тех	0,18 Микр.тех	0,18 Микр.тех	0,18 Микр.тех
Адресуемая память	64 Гб	64 Гб	64 Гб	64 Гб
Виртуальная память	---	--	--	--
Краткая характеристика	Рабочие станции, мобильные ПК, офисные и пользовательские ПК	Офисные и домашние ПК. Одновосми процесссорные рабочие станции	Офисные и домашние ПК Одно-двух процессорные серверы	Офисные и домашние ПК. Двух процессорные серверы и рабочие станции

1.2. Архитектура ЭВМ

Под архитектурой ЭВМ понимается абстрактное представление машины в терминах основных функциональных

моделей, языка ЭВМ, структура данных. Архитектура не определяет особенностей реализации аппаратной части ЭВМ, времени выполнения команд, степени параллелизма при выполнении программы, ширины шин и других аналогичных характеристик. Архитектура отображает аспекты структуры ЭВМ, которые являются видимыми для пользователя: системы команд, режимы адресации, форматы и длину данных, набор регистров ЭВМ, доступных пользователю. Одним словом термин «архитектура» используется для описания возможностей, предоставляемых ЭВМ, а термин «организация» определяет, как эти возможности реализованы.

Все ЭВМ содержат следующие функциональные блоки (рис.1.1), имеющие свою микро архитектуру: процессор, состоящий из арифметико-логического устройства, аккумулятора, регистров, счетчиков команд, устройства управления, память, устройства ввода и вывода информации.

Объединение функциональных блоков в ЭВМ осуществляется посредством следующей системы шин: шины данных, обращение к различным устройствам ЭВМ, и шины управления для передачи управляющих сигналов. Для связи пользователя с ЭВМ предусмотрен пульт управления, который позволяет выполнять такие действия, как пуск ЭВМ, останов, под действием которого прекращается поступление сигналов с генератора тактирующих сигналов и процессор переходит в состояние ожидания. Загрузка начального адреса программы в программный счетчик, индикация содержимого ячеек памяти и регистров процессора, пошаговое выполнение команд программы при ее отладке. Для ввода данных в ЭВМ могут быть использованы обычные периферийные устройства: НГД, телетайп, клавиатура, перфоратор. Данные от процесса или объекта управления могут быть введены непосредственно. Если данные имеют аналоговую форму, например, напряжение или ток, то они должны быть вначале преобразованы в цифровую форму, так как ЭВМ может работать только по двоичной системе. Эти преобразования осуществляются с помощью аналого-цифрового преобразователя (АЦП). Аналогично данные для управления процессом, полученные на выходе ЭВМ, могут быть преобразованы в аналоговую форму с помощью цифро-аналогового преобразователя (ЦАП).

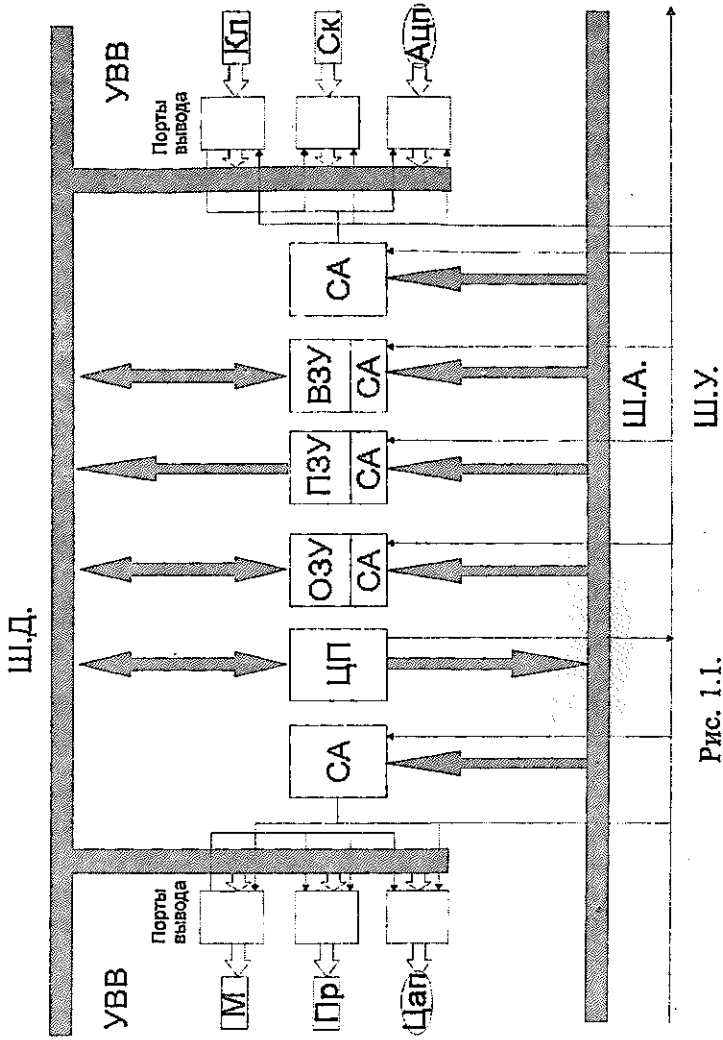


Рис. 1.1.

Данные, поступающие с устройства ввода, передаются на шины данных в виде 8-разрядных параллельных или последовательных кодовых сигналов через порт ввода. Селектор адреса определяет порт ввода, который передает данные на шину данных в некоторый момент времени.

1.2.1. Система шин микро-ЭВМ

Для соединения различных блоков ЭВМ используется так называемая «шинная структура». Шина – это многожильный кабель. В микро-ЭВМ имеется три различные шины: адреса, данных и управления.

Шина адреса. Число линий в шине адреса определяется числом разрядов в адресе памяти. Большинство микро-ЭВМ имеют 16,20,32 и 36 разрядные адреса. (8080 - 16 разрядные, 8086 – 20 разрядные,80486 – 32 разрядные).

Шина данных. Число линий равно длине слова микро-ЭВМ, и составляло на первых процессорах 8-разрядов на последних моделях 64 разряда.

Шина управления. Содержит линии управления, число которых зависит от типа микро-ЭВМ. Сигналы, включающие различные блоки в работу, передаются по шине управления.

Когда говорят о шинах в микро-ЭВМ, то имеют в виду внутренние шины. Микро-ЭВМ имеет разветвленную структуру внутренних шин, которая показана на рис.1.1 пунктиром стрелками показано принимает и (или) передает сигналы данный блок. Если сигналы и принимаются и передаются, то это соединение шин называются двунаправленными.

Примечание 1. Шинная структура позволяет прямо подсоединять к микро-ЭВМ новые блоки. Это имеет большое значение.

Примечание 2. Шина данных является двунаправленной. Это естественно, не относится к соединению между шиной данных и ПЗУ.

1.2.2. Система соединения блоков микро-ЭВМ

Информация может поступать на данную шину только от одного блока микро-ЭВМ. Например, информация может поступить на шину данных только от одного из следующих блоков: ЦП, ОЗУ, ПЗУ, одного из портов ввода. Для достижения этого в каждую выходную линию шины каждого порта включена буферная схема.

Сигналы, которые переключают буферные схемы в требуемое состояние, поступает от устройства управления УУ. Таким образом, УУ выбирает блок ЭВМ, который передает данные в шину.

1.2.3. Обмен данными в микро-ЭВМ

От ЦП к памяти (рис 1.2). Операция записи в память выполняется для передачи данных из ЦП в выбранную ячейку памяти. Происходит следующим образом:

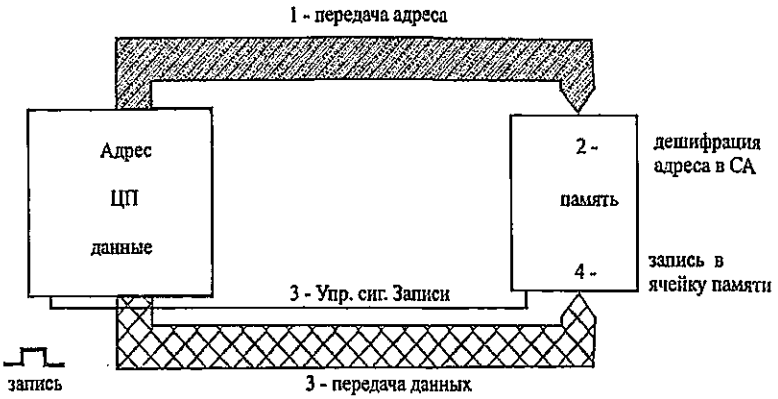


Рис. 1.2.

1. Адрес ячейки памяти, в которую нужно записать данные, поступают из ЦП в память;
2. Адрес дешифруется селектором адреса СА;
3. ЦП передает в память данные и одновременно управляющий сигнал записи;
4. Данные заносятся в ячейку памяти по заданному адресу.

От памяти к ЦП (рис1.3). Операция считывания (чтения) из памяти выполняется для передачи данных, имеющихся в выбранной ячейке памяти, в ЦП.

1. Адрес ячейки памяти, содержимое которой должно быть передано в ЦП, поступает из ЦП в память;
2. Адрес дешифруется селектором адреса;
3. ЦП направляет в память сигнал считывания;
4. Содержимое ячейки памяти поступает в ЦП.

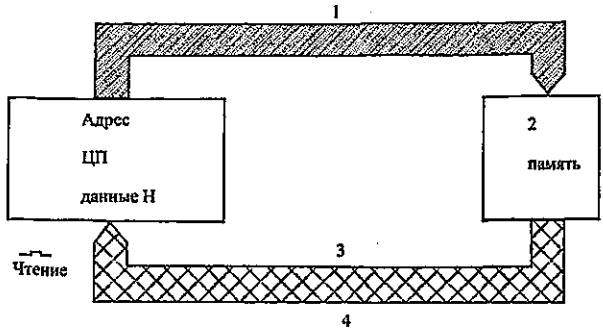


Рис. 1.3.

Из устройства ввода в ЦП (рис 1.4).

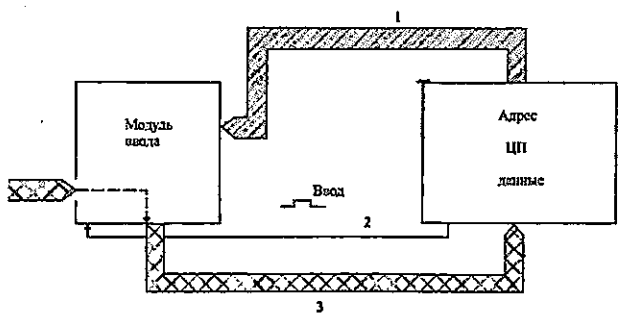


Рис.1.4.

1. ЦП направляет адрес выборки устройства ввода-вывода (УВВ) на модуль ввода-вывода через адресную шину и тем самым определяет, с какого порта должны быть получены данные;
2. Чтобы показать, что требуется операция ввода, ЦП посылает сигнал ввода модулю ВВ через шину управления;
3. Данные, представленные в выбранном порте ввода, переписываются в ЦП.

От ЦП на устройство вывода (рис. 1.5).

1 ЦП направляет адрес выборки устройства ввода-вывода (УВВ) на модуль ввода-вывода через адресную шину и тем самым определяет, на какой порт должны быть выведены данные;

2. Чтобы показать, что требуется операция вывода, ЦП посылает сигнал вывода модулю ВВ через шину управления;
3. Данные, находящиеся в ЦП переписываются в порт вывода.

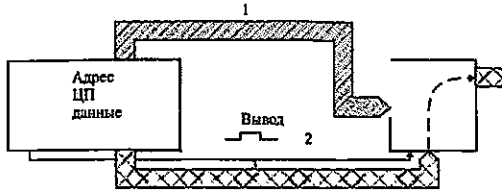


Рис.1.5.

1.2.4. Типы памяти

Основная память системы ЭВМ состоит из памяти программ и памяти данных.

Когда программа помещается в память постоянно, она обычно располагается в устройстве, называемом *постоянным запоминающим устройством* (ПЗУ). Постоянное запоминающее устройство является обычно кристаллом программируемой интегральной микросхемой с неизменяемой программой. Память изменяющихся данных, называемая *оперативным запоминающим устройством* (ОЗУ) или памятью с произвольным доступом (первое название используется более часто), составляется также обычно из ИС.

Информация (чаще всего программа) постоянно хранится в ПЗУ. Ее можно только считывать и нельзя менять или обновлять. Имеются три типа ПЗУ:

1. ПЗУ, запрограммированное изготовителем микроЭВМ;

2. Программированное ПЗУ(ППЗУ). Пользователь может запрограммировать ППЗУ с помощью программирующего устройства, которое выжигает адресуемые диоды в матрице ППЗУ; после этого дальнейшие изменения содержимого памяти невозможны;

3. РППЗУ (перепрограммируемые ППЗУ) или стираемые ППЗУ.

В этих устройствах информация может стираться несколько раз с помощью ультрафиолетового облучения. Пере-

программирование осуществляется с помощью программирующего устройства ППЗУ.

ОЗУ – оперативное запоминающее устройство – это память, из которой процессор может считывать или в которую может записывать информацию. Поэтому ее используют для хранения промежуточных результатов вычислений и переменных и для микро-ЭВМ она является своего рода блокнотом для записей. При выключении питания информация в ОЗУ стирается.

Программы пользователя, которые по своей природе изменяемы, также помещаются в оперативной части памяти вместе с данными. На рис. 1.6 ОЗУ и ПЗУ показаны раздельно, потому что они обычно составляются различными

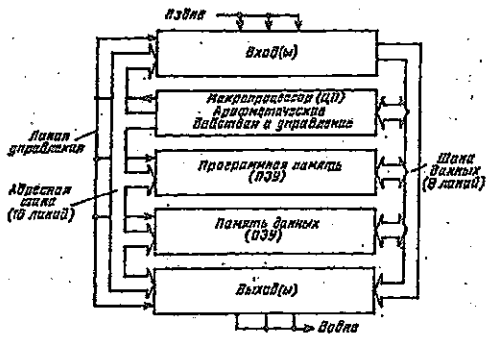


Рис. 1.6. Типовая структура микро-ЭВМ.

Приведенная на рис. 1.6 система может рассматриваться как обобщенная архитектура микро-ЭВМ. Большая часть микро-ЭВМ содержит, по меньшей мере, эти характерные устройства, а также и некоторые другие. Для упрощения таких схем обычно не показывают питание, источник тактовых импульсов (часы) и некоторые входящие в микро-ЭВМ линии, необходимые для ее работы,

Контрольные вопросы

1. Центральный процессор представляет собой интегральную схему, называемую _____.

2. Какой блок на рис. 1.6 должен рассматриваться как ЦП?

3. Перечислить три типа связей в микро-ЭВМ, приведенной на рис. 1.6.

4. Адресная шина на рис. 1.6 для кодированной информации является однонаправленной, шина ____, напротив, является двунаправленной.

5. Обычно постоянные программы располагаются в БИС, называемой ___.

6. Какой тип памяти сокращенно называется ПЗУ?

7. Какой тип памяти сокращенно называется ОЗУ?

8. Временные данные и программы располагаются в БИС, называемой ____ (ОЗУ, ПЗУ).

9. Размещение данных в микро-ЭВМ выполняется ____ (временно, постоянно).

10. Размещение программ в ПЗУ выполняется (временно, постоянно).

11. Ввод или вывод информации в (из) микро-ЭВМ выполняется с использованием ____ (порта, датчика времени).

1.3. Работа на микро-ЭВМ

Обратимся к рис. 1.7, иллюстрирующему работу микро-ЭВМ. На этом примере показаны следующие процедуры: 1— нажатие клавиши *А* клавишного устройства; 2—размещение буквы *А* в памяти; *3*— воспроизведение буквы А на экране дисплея.

Процедура ввод—размещение—вывод, показанная на рис. 1.7, является типичной. Аппаратные электронные средства, используемые в такой схеме, довольно сложны.

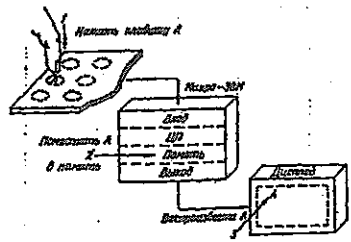


Рис. 1.7 Действия микро-ЭВМ.

Однако анализ процесса передачи данных поможет понять роль различных устройств, составляющих систему. Более под-

робная схема на рис. 1.8 позволяет осмыслить типичную процедуру в микро-ЭВМ — ввод—размещение—вывод. Прежде всего, внимательно рассмотрим содержимое программной памяти на рис. 1.8. Заметим, что команды предварительно были загружены в шесть первых ячеек памяти. Согласно рисунку текущими командами в программной памяти являются:

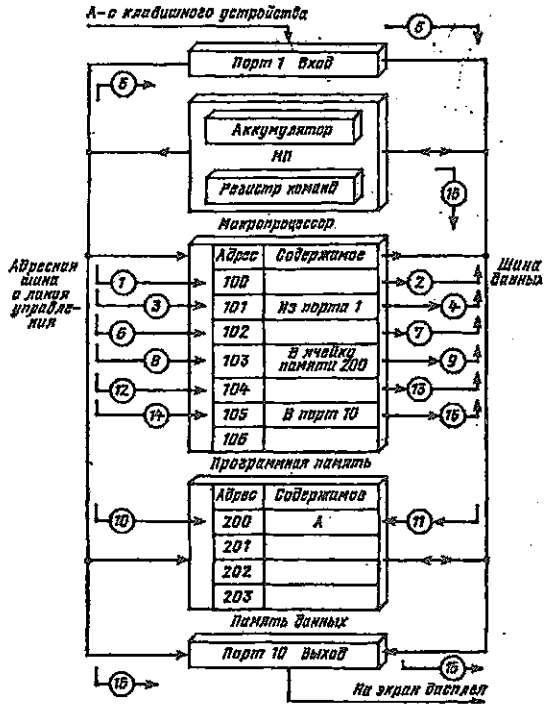


Рис. 1.8. Поэтапные действия микро-ЭВМ в ходе выполнения команды программной памяти.

1. ВВЕСТИ (INPUT) данные через порт ввода 1.

2. РАЗМЕСТИТЬ (STORE) данные, поступающие с порта 1, в ячейке памяти данных 200.

3. ВЫВЕСТИ (OUTPUT) данные через порт вывода 10. Заметим, что приведенная выше программа содержит только три команды. Однако в программной памяти на рис. 1.8 имеется шесть команд. Это обусловлено тем, что обычно коман-

ды делятся на две части. Первая часть команды 1 была ВВЕСТИ данные, вторая указывает нам ее происхождение (порт 1). Первая часть представляет собой действие и называется *операцией,* а вторая — *операндом.* Операция и операнд помещены в ячейки памяти, показанные на рис. 1.8. Что касается первой команды на этом рисунке, операция ВВЕСТИ содержится в ячейке памяти 100, а в ячейке памяти 101— операнд (порт 1), который нам указывает, откуда поступит информация.

На рис. 1.8 введены два новых элемента—*регистры.* Этими двумя специальными регистрами являются *аккумулятор* и *регистр команд.*

Последовательность событий, происходящих в микроЭВМ в ходе выполнения приведенного на рис. 1.7 примера, подробно показана на рис. 1.8. Не забудем, что МП является центром всех преобразований данных и операций.

Обратимся к рис. 1.8, чтобы проследить все этапы.

Этап 1. Микропроцессор выставляет адрес 100 на адресную шину. Линия управления *признает* (или активизирует) ввод *считывания* из ИС программной памяти (считывать данные означает копировать информацию из ячейки памяти). Этот этап на рис. 1.8 обозначен 1 в кружке.

Этап 2. Программная память выставляет первую команду на шину данных (ВВЕСТИ данные), а МП принимает эту кодированную информацию. Это послание помещается в специальное пространство памяти МП, называемое регистром команд. Оно *декодируется* МП (интерпретируется), этой команде нужен операнд.

Этап 3. Микропроцессор выставляет на адресную шину адрес 101. Линией управления активизируется вход считывания из программной памяти.

Этап 4. Программная память помещает операнд (изъятый из порта 1) на шину данных. Этот операнд находился в ячейке памяти 101. Кодированное послание (адрес порта 1) взято на шине данных и помещено в регистр команд. Теперь МП декодирует полную команду (ВВЕСТИ данные, поступающие из порта 1).

Этап 5. Микропроцессор побуждает открыть порт 1 посредством адресной шины и линии управления устройством ввода. Кодированная форма *А* передается в аккумулятор, где и размещается. Важно заметить, что МП все время действует

в последовательности — *извлечение — декодирование — выполнение.* Он извлекает сначала из программной памяти команду, затем расшифровывает ее и, наконец, выполняет. Продолжим теперь рассмотрение программы.

Этап 6. Микропроцессор выставляет на адресную шину адрес ячейки памяти 102 и активизирует вход считывания из программной памяти посредством управляющих линий.

Этап 7. Код команды ПОМЕСТИТЬ данные считывается с шины данных, принимается МП и помещается в регистр команд.

Этап 8. Микропроцессор декодирует эту команду и определяет, что нужен операнд. Он выставляет на шину данных следующий адрес 103 и активизирует вход считывания из программной памяти.

Этап 9. Код операнда «В ячейку памяти 200» из памяти (программы) помещен на шину данных, МП принимает операнд и помещает его в регистр команд. Команда ПОМЕСТИТЬ данные, расположенные в ячейке памяти 200, полностью извлечена и декодирована.

Этап 10. Теперь начинается процесс выполнения. Микропроцессор выставляет на адресную шину адрес 200 и активизирует вход *записи* в память (запись означает, что данные введутся в память).

Этап 11. Микропроцессор выдает помещенную в аккумуляторе информацию на шину данных (кодированная форма Л). Это *А* записано в ячейке памяти *200* и таким образом теперь выполнена вторая команда.

Этап 12. Теперь МП должен извлечь следующую команду. Он адресует ячейку памяти 104 и активизирует вход считывания из памяти.

Этап 13. Команда ВЫВЕСТИ данные помещена на шину данных, МП принимает ее и помещает в регистр команд. Микропроцессор декодирует послание и устанавливает, что нужен операнд.

Этап 14. Микропроцессор помещает адрес 105 на адресную шину и активизирует вход считывания из памяти.

Этап 15. Память помещает код операнда в порт 10 на шину данных. Этот код принимается МП, который помещает его в регистр команд.

Этап 16. Микропроцессор декодирует команду ВЫВЕСТИ данные в порт *10* полностью, активизирует порт

10 посредством адресной шины и управляющей линии устройства выводи. Он помещает код А (постоянно находящийся в аккумуляторе) на шину данных. Наконец А передается портом 10 на видеотерминал.

Большинство микро-ЭВМ передают информацию описанным сейчас и показанным на рис. 1.8 способом. Самые большие различия сосредоточены в элементах ввода и вывода. Может быть так, что потребуется больше этапов для осуществления этих операций.

Важно отметить, что МП является центром всех операций и полностью ими управляет. Он следует последовательности — извлечение—декодирование—выполнение. Выполняемые операции, напротив, диктуются командами, помещенными в памяти.

Контрольные вопросы

1. Список команд для использования в микро-ЭВМ составляет ____.

2. Программа помещается внутри микро-ЭВМ в памяти ____.

3. Большинство команд микро-ЭВМ состоит из двух частей—операции и ____.

4. Команды, составляющие программу микро-ЭВМ, обычно выполняются ____ (последовательно, случайно).

5. После выполнения команды ВЫВЕСТИ данные в порт 10 МП на рис. 1.8 обратится по адресу ячейки памяти ____ за следующей командой.

6. Сокращение ЦП означает ____.

7. Следствием такой команды, как ПОМЕСТИТЬ данные в ячейку памяти 201, была бы передача содержащихся в ЦП данных в ячейку памяти 201, расположенной в памяти ____.

8. Обратимся к рис. 1.8. Результатом команды ПОМЕСТИТЬ данные в ячейку памяти 202 была бы передача данных из МП, расположенных в ____ (аккумуляторе, регистре команд), в ячейку памяти ____ОЗУ.

9. Процесс ____ (считывания, записи) выполняется, когда данные извлекаются из ячейки памяти.

10. Помещение данных в ячейку памяти является операцией ____ (считывания, записи).

11. Для выполнения каждой команды МП действует в последовательности: ____, ____, ____ .

12. После этапа 16 (рис. 1.8) каким будет содержимое аккумулятора МП после выполнения команды вывода данных в порт 10.

13. Обратиться к рис. 1.8. Сохраняются ли команды, расположенные в ячейках памяти от 100 до 105, после этапа 16.

ГЛАВА 2. ОСНОВНЫЕ ЭЛЕМЕНТЫ ЦИФРОВОЙ ТЕХНИКИ

2.1. Логические элементы

Основной элементной базой современной дискретной техники является интегральная микроэлектроника.

На практике различают следующие интегральные схемы: МИС – малые интегральные схемы (10 элементов); СИС – средние интегральные схемы (до 100); БИС – большие интегральные схемы (100-1000) и СБИС – сверхбольшие интегральные схемы (свыше 1000 элементов).

В качестве активных элементов цифровых микросхем сейчас применяются два типа транзисторов: биполярные и полевые (униполярные). Последние имеют структуру **металл – окисел – полупроводник** (МОП) или, как ее еще называют, **металл – диэлектрик – полупроводник** (МДП).

Способ соединения транзисторов между собой в пределах одного элемента определяет их логический базис или, проще логику. Из логических интегральных схем на биполярных транзисторах в настоящее время набольшее распространение имеют: транзисторно-транзисторная логика (ТТЛ) в нескольких модификациях, эмиттерно-связанная логика (ЭСЛ), или, как ее еще называют, логика на переключателях тока (ПТТЛ), и в меньшей мере – диодно-транзисторная логика (ДТЛ).

Для создания микросхем с большой степенью интеграции высокого контро-действия используют инжекционно-интегральную логику (ИИЛ, или $И^2Л$).

Микросхемы на полевых транзисторах также широко используются в настоящее время. Наиболее распространены и перспективны схемы, основанные на совместном включении пары транзисторов с каналами разных видов проводимости, так называемые комплементарные структуры (КМДП-структура).

2.2. Булева алгебра

Математический аппарат, описывающий действия дискретных устройств, базируется на алгебре логики, или, как ее еще называют по имени автора – английского математика Джорджа Буля (1815-1864 гг.) Булевой алгебре.

В практических целях первым применил его американский ученый Клод Шеннон в 1938 г. при исследовании электрических цепей с контактными выключателями.

Булева алгебра оперирует двоичными переменными, которые условно обозначаются, как 0 и 1, и подчиняются условию:

$x = 1$, если $x \neq 0$, и

$x = 0$, если $x \neq 1$.

В ее основе лежит понятие переключательной, или Булевой, функции вида $f(x_1, x_2, ..., x_n)$ относительно аргументов x_1, x_2, ..., x_n, которая, как и ее аргументы, может принимать только два значения – 0 и 1. Как частный случай, двоичные переменные могут постоянно сохранять одно из значений – 0 либо 1. Логическая функция может быть задана словесно, алгебраическим выражением и таблицей, которая называется таблицей **истинности**.

Действия над двоичными переменными производятся по правилам логических операций. Между обычной, привычной нам алгеброй и алгеброй логики имеются существенные различия в отношении количества и характера операций, а также законов, которым они подчиняются.

Простейших логических операций четыре: отрицание (инверсия, операция НЕ (NO)), логическое умножение (конъюкция, операция И (AND)), логическое сложение (дизъюнкция, операция ИЛИ (OR)) и исключающее сложение (операция исключающее ИЛИ (XOR)).

Более сложные логические преобразования можно свести к указанным операциям.

Операция отрицания выполняется над одной переменной и характеризуется следующими свойствами: функция

$y = 1$ при аргументе $x = 0$ и

$y = 0$ при аргументе $x = 1$.

Обозначается отрицание чертой над переменой с которой производится операция:

$y = \bar{x}$ (игрек равен не икс)

и наоборот $\bar{y} = x$.

Операция логического умножения (конъюкция) для двух переменных обозначается следующим образом:

$y = x_1 \wedge x_2$ или $y = x_1 * x_2$

$0 \cdot 0 = 0; \quad 0 \cdot 1 = 0; \quad 1 \cdot 0 = 0; \quad 1 \cdot 1 = 1.$

Операция логического сложения (дизъюнкция). Обозначают ее следующим образом

$$y = x_1 \vee x_2 \quad \text{или} \quad y = x_1 + x_2$$

для двух переменных

$$0 \vee 0 = 0; \quad 0 \vee 1 = 1; \quad 1 \vee 0 = 1; \quad 1 \vee 1 = 1.$$

Двоичные переменные, входящие в логические уравнения, можно представить двумя различными электрическими сигналами. Путем преобразований этих сигналов получают другие, тоже двоичные сигналы, которые соответствуют результатам определенных логических операций.

Имея запись Булевой функции

$$y = f(x_1, x_2, ..., x_n),$$

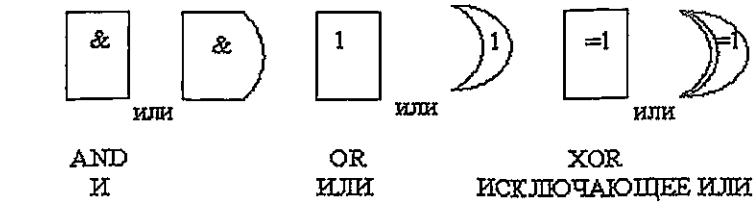
можно составить развернутую электрическую схему, которая будет преобразовывать логические сигналы x_1, x_2, ..., x_n согласно указанной функции.

Устройства, выполняющие в аппаратуре логические операции, называют логическими элементами. Логические элементы различаются между собой характером реализуемой функции, числом входов (по числу одновременно действующих переменных), числом выходов и другими признаками. Работа их оценивается только с точки зрения логики, без учета практического воплощения (технической базы, способы питания и т.п.).

Входы и выходы логических элементов в зависимости от уровня сигнала при котором воспринимается или вырабатывается определенное значение двоичной переменой, подразделяются на прямые и инверсные.

На логические входы можно подавать постоянные логические уровни 1 и 0.

На принципиальных схемах логические элементы согласно ГОСТ 2.743-82 «Обозначения условные графические в схемах. Элементы цифровой техники» изображают прямоугольником в верхней части которого указывают символ функции: & для И, 1 для ИЛИ и = 1 для ИСКЛЮЧАЮЩЕЕ ИЛИ рис.2.1



Например, $y = f(x_1, x_2)$

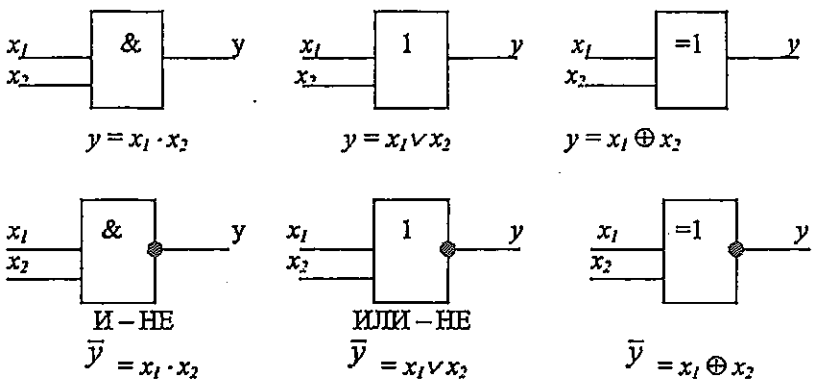


Рис. 2.1.

В таблице 2.1 приведены семь основных логических элементов цифровых схем. В таблице истинности 0 означает низкий уровень напряжения (LOW), а 1 – высокий (HIGH).

Приведенный на рис. 2.2. пример несколько поясняет способ преобразования информации логическими элементами.

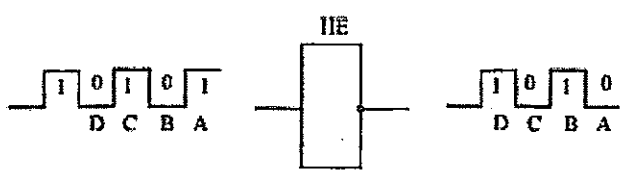


Рис. 2.2.

Согласно таблицы истинности на выходе элемента НЕ будет а=0, в=1, с=0, дв=1 на рис.2.3. показаны другие примеры.

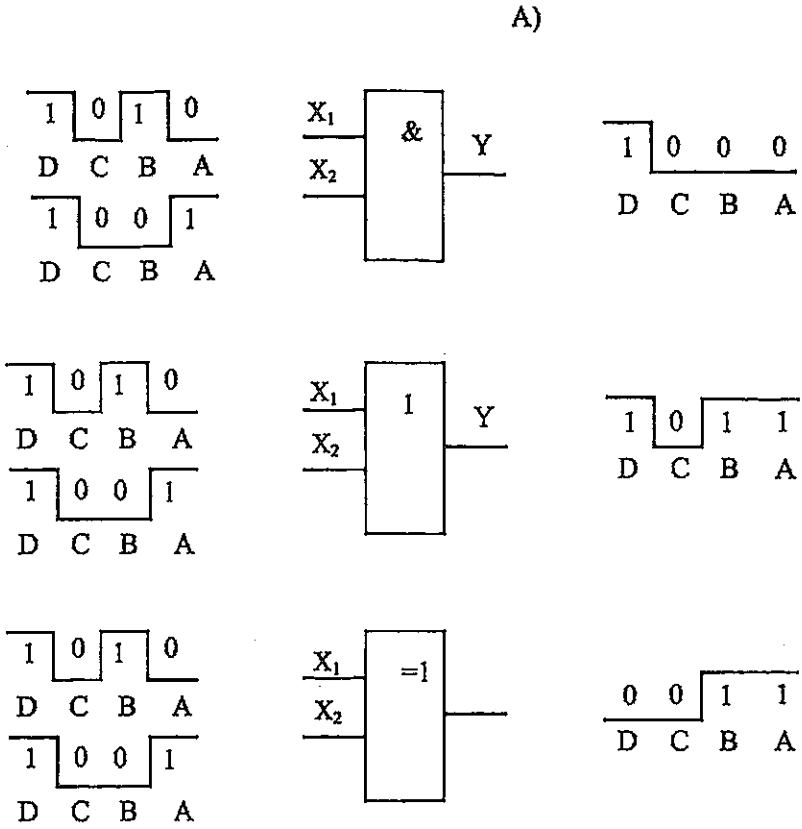
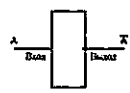
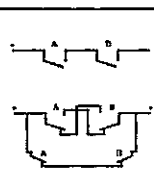
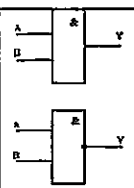
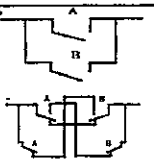
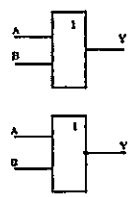
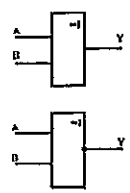


Рис.2.3.

Таблица 2.1.

Семь логических функций

Логические функции	Электрический эквивалент	Обозначения логических элементов	Таблица истинности	Булева функция
Инвертор НЕ (NOT)			Вход Выход A $\overline{A}$ 0 1 1 0	$A=\overline{A}$
И (AND) НЕ - И (NAND)			Входы Выходы A B И НЕ-И 0 0 0 1 0 1 0 1 1 0 0 1 1 1 1 0	$A\cdot B=Y$ $\overline{A\cdot B}=Y$
ИЛИ (OR) НЕ - ИЛИ (NOR)			Входы Выходы B A ИЛИ НЕ-ИЛИ 0 0 0 1 0 1 1 0 1 0 1 1 1 0	$A+B=Y$ $\overline{A+B}=Y$
ИЛИ-ИСКЛЮЧ (XOR) НЕ-ИЛИ-ИСКЛЮЧ (XNOT – OR)			Входы Выходы B A OR НЕ-XOR 0 0 0 1 0 1 1 0 1 0 1 0 1 1 0 1	$A\oplus B=Y$ $\overline{A\oplus B}=Y$

2.3. Комбинации логических элементов

Цифровые системы строятся на основе комбинаций логических элементов. Такие комбинации могут быть описаны таблицей истинности, булевой функцией или логической схемой.

Рассмотрим таблицу истинности (табл. 2.2). Эта таблица описывает все возможные комбинации четырех входов (*A, B, C, D)*. Заметим, что только комбинация 1010 даст 1 или Н-сигнал на выходе. Эквивалентная этой таблице истинности булева функция приведена в заголовке табл. 2.2. Входы подчинены операции И, что дает булеву функцию $D*C*B*A= Y$ (читается как D и НЕ-С и *B* и НЕ-А дают на выходе У),

Таблица 2.2.

Получение булевой функции из таблицы истинности

$$D*\overline{C}*B*\overline{A}=Y$$

$$D*\overline{C}*B*\overline{A}=Y$$

Входы				Выход	Входы				Выход
D	C	B	A	Y	D	C	B	A	Y
0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	1	0
0	0	1	0	0	1	0	1	0	1
0	0	1	1	0	1	0	1	1	0
0	1	0	0	0	1	1	0	0	0
0	1	0	1	0	1	1	0	1	0
0	1	1	0	0	1	1	1	0	0
0	1	1	1	0	1	1	1	1	0

Логическая схема построена исходя из булевой функции (см. рис. 2.4, а). Входы *A* и *C* должны быть инвертированы инверторами. На выходе использован элемент И с четырьмя входами.

Явно более простой вариант такой логической схемы приведен на рис. 2.4, *б*. На этой схеме инверторы обозначены маленькими кружками, называемыми *кружками инверсии,* которые могут рассматриваться как LOW-активные входы. Другими словами, чтобы активизировать элемент И, на рис. 2.4, *б* входы *A* и *C* должны быть LOW, а входы *B и D*— HIGH. Так как входы *D* и *B* должны быть HIGH для активизации элемента И, их называют HIGH-*активными входами.*

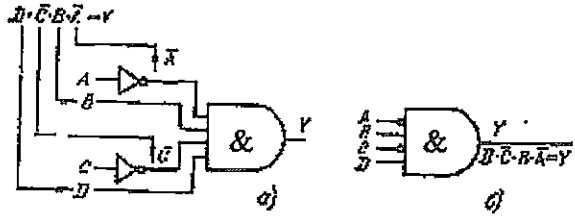


Рис. 2.4. Построение логических схем
а) на основании булевой функции; б) упрощенный вариант.

Рассмотрим другую таблицу истинности (табл. 2.3). Здесь две комбинации входов вызовут 1 или HIGH на выходе. Булевой функцией, соответствующей этой таблице становится тогда такая: D* $\overline{C}$* $\overline{B}$* $\overline{A}$ + D *C *$\overline{B}$* A = Y (читается: D И НЕ-C и НЕ-B и НЕ-A или D и C и НЕ-B и A равно Y на выходе).

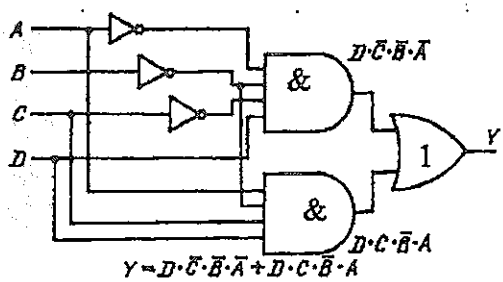


Рис.2.5.

Затем на основании булевой функции получаем логическую схему (см. рис. 2.5). Заметим, что такая булева функция обусловлена сетью логических элементов И-ИЛИ, ближайшим к выходу является элемент ИЛИ. Такая специальная схема называется *формой суммы произведений* или *реализованной формой* булевой функции. Таблица 2.3 показывает состояние составляющих реализуемой булевой функции в колонке выхода таблицы истинности, где на выходе появляется 1.

Таблица 2.3.

Таблица истинности, соответствующая эквивалентному выражению $\overline{D*C*B*A+D*C*B*A}=Y$

Входы				Выход	Входы								Выход
D	C	B	A	Y	D	C	B	A					Y
0	0	0	0	0	1	0	1	0	0	0	0	1	1
0	0	0	1	0	1	0	1	0	1	0	1	1	0
0	0	1	0	0	1	1	1	1	0	0	0	1	0
0	0	1	1	0	1	1	1	1	1	0	1	1	0
0	1	0	0	0									0
0	1	0	1	0									1
0	1	1	0	0									0
0	1	1	1	0									0

Контрольные вопросы

1. Записать выражение булевой функции таблицы истинности (табл. 2.4).
2. Начертить логическую схему системы, соответствующую таблице истинности (табл. 2.4).
3. Обратимся к рис. 2.6,6. Входы *A, B* и *C* _____ (H-L-) активные, тогда как вход *D*____ (H-, L") активный.

Таблица 2.4.

Таблица истинности

Входы				Выход	Входы				Выход
D	C	B	A	Y	D	C	B	A	Y
0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	1	0
0	0	1	0	0	1	0	1	0	0
0	0	1	1	0	1	0	1	1	0
0	1	0	0	0	1	1	0	0	0
0	1	0	1	0	1	1	0	1	0
0	1	1	0	0	1	1	1	0	0
0	1	1	1	1	1	1	1	1	0

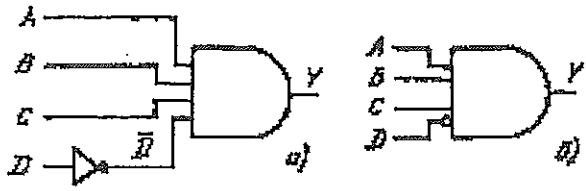


Рис. 2.6 Логическая схема (а) и ее упрощенный вариант (б)

4. Записать выражение булевой функции таблицы истинности (табл. 2.5).

Таблица 2.5.

Таблица истинности

Входы				Выход	Входы				Выход
D	C	B	A	Y	D	C	B	A	Y
0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	1	1
0	0	1	0	0	1	0	1	0	0
0	0	1	1	0	1	0	1	1	0
0	1	0	0	0	1	1	0	0	0
0	1	0	1	0	1	1	0	1	0
0	1	1	0	0	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1

5. Начертить логическую схему системы, соответствующую таблице истинности (табл. 2.5).

2.4. Триггеры и защелки

Логические цепи могут быть разделены на две большие группы. Первая – цепи комбинационной логики, составленные из логических элементов, вторая – последовательные логические цепи, состоящие из элементов, называемых триггерами. Триггеры объединяют в системы с целью образования последовательных логических цепей, предназначенных для

поминания. Триггер запомнит свои входные сигналы даже тогда, когда эти сигналы будут сняты. Логический элемент, напротив, не сможет запомнить свое состояние на выходе, если будут сняты входные сигналы.

В простейшем исполнении триггер представляет собой симметричную структуру из двух логических элементов ИЛИ – НЕ либо И – НЕ, охваченных перекрестной положительной обратной связью. Такие триггеры называют симметричными, рис.2.7.

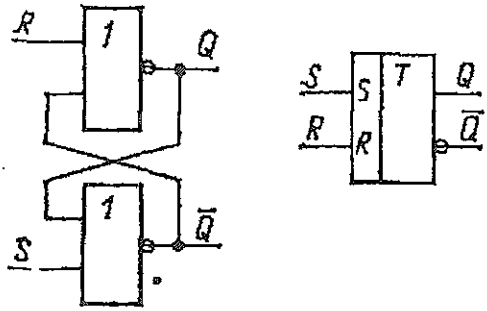


Рис. 2.7.

Этот триггер, собранный на двух логических элементах ИЛИ – НЕ (бистабильная ячейка, ячейка памяти, асинхронный RS-триггер) обладает двумя устойчивыми состояниями, которые обеспечиваются за счет связи выхода каждого элемента с одним из входов другим.

Свободные входы служат для управления и называются информационными или логическими.

За счет перекрестного соединения выводов и входов создаются условия, при которых при отсутствии входных сигналов один из логических элементов будет заперт, а другой открыт.

Состояние триггера часто отождествляют с сигналом на прямом выходе, т.е. говорят, что триггер находится в единичном состоянии, когда $Q = 1$, а $\overline{Q} = 0$, и в нулевом, когда $Q = 0$, $\overline{Q} = 1$.

Вход, по которому триггер устанавливается в единичное состояние ($Q = 1$, а $\overline{Q} = 0$, называют входом S (от англ. Сло-

ва Set - установка), а в нулевое ($Q = 0$, $\overline{Q} = 1$) – входом R (reset – возврат).

В зависимости от сигналов, подаваемых на входы S и R триггера, он может принимать следующие состояния.

R^n	S^n	Q^{n+1}	$\overline{Q}^{n+1}$
0	0	0	1
0	1	1	0
1	0	0	1
1	1	н/о	н/о

– неопределенное состояние

Помимо RS триггеров широко применяются IK, D(DV) и T(TV) – триггеры, функциональное назначение входов которых приведено в табл.2.6.

Таблица 2.6.

Условное обозначение	Назначение
Информационные входы	
S	Установка в 1
R	Установка в 0
I	Установка 1
K	Установка 0
T	Вход счетного триггера (счетный вход)
D	Вход установки D и DV в 1
Управляющие коды	
V	Подготовительный вход разрешения приема информации
C	Исполнительный вход приема информации. Вход синхронизации (тактирующий вход).

2.4.1. JK – триггеры

Этот тип триггеров не имеет неопределенных состояний. Функциональная особенность JK – триггеров состоит в том, что при всех входных комбинациях, кроме RS – триггера, причем вход J играет роль входа S, а K – вход соответствует

R — входу. При входной комбинации $J^n = K^n = 1$ в каждом такте происходит опрокидывание триггера и выходные сигналы меняют свое назначение.

JK- триггеры относятся к универсальным устройствам. Их универсальность имеет двойственный характер. Во-первых, это триггеры, с равным успехом могут быть использованы в регистрах, счетчиках, делителях частоты и других узлах, и, во-вторых, путем определенного соединения выводов они легко обращаются в триггеры других типов.

J^n	K^n	Q^{n+1}
0	0	Q^n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

По способу управления JK- триггеры, также как и RS — триггеры, могут быть асинхронными и синхронными.

Логическая структура простейшего синхронного JK триггера показана на рис. 2.8.

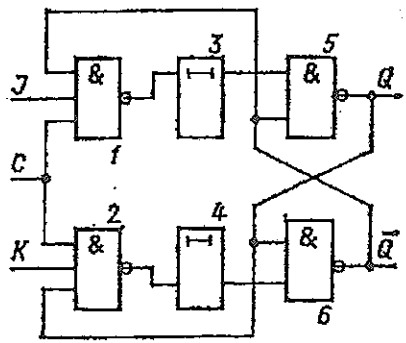


Рис.2.8. К564ТВ1 — JK-триггер:
3,4 — элемент задержки времени

2.4.2. D — триггеры

D-триггеры в отличие от рассмотренных типов имеют для установки в состоянии 1 и 0 один информационный вход (D — вход).

Функциональная особенность триггеров этого типа состоит в том, что сигнал на выходе Q в такте n + 1 повторяет входной сигнал D^n в предыдущем такте n и сохраняет это состояние до следующего тактового импульса. Другими словами, D – триггер задерживает на один такт информацию, существовавшую на входе D.

такт – n		такт – n +1
D^n	Q^n	Q^{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

На рис. 2.9 показана логическая структура синхронного D – триггера со статическим управлением. Триггер выполнен на элементах И – НЕ. Элементы 3 и 4 образуют ячейку памяти, а 1 и 2 – схему управления. В паузах между тактовыми импульсами 1 и 2 закрыты и на их выходах существуют сигналы $q_1 = q_2 = 1$, что служит нейтральной комбинацией для основной ячейки памяти.

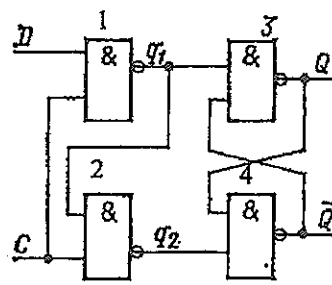


Рис.2.9.

Если в схеме элементы И – НЕ заменить на ИЛИ – НЕ, то получится $\overline{D}$ - триггер.

На рис.2.10., 2.11. показан типовой D-триггер

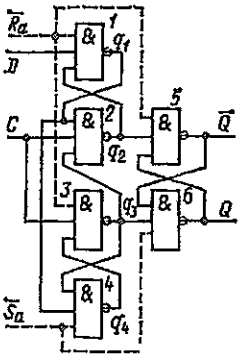


Рис.2.10.

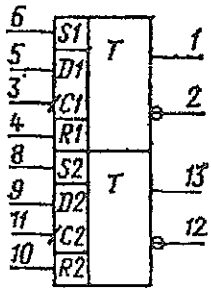


Рис.2.11.

Входы R_a и S_a для асинхронной установки триггера в нулевое и единичное состояния.

2.5. Шифраторы, дешифраторы и семисегментные индикаторы

Рассмотрим функциональную схему калькулятора (рис. 2.12,а). В этой цифровой системе поступающая с *клавиатуры* десятичная информация должна быть переведена в двоично-десятичный код Эта операция *(кодирование)* выполняется цифровым устройством, называемым *шифратором.* Выходящая из ЦП двоично-десятичная информация переводится *дешифратором* в специальный код *семисегментного индикатора.*

На рис. 2.12,б приведена логическая схема *шифратора десятичного ДДК с приоритетом.* У этого шифратора девять входов, активизируемых L-сигналом, и четыре выхода, соединенных со световым индикатором. Соединения с клавиатурой показаны слева, каждый занумерованный ключ подсоединен на соответствующий вход шифратора. В приведенном на рис. 2.12,6 примере показано, что была нажата клавиша 7, в результате чего заземлился вход 7 шифратора. Это повлечет за собой вывод двоично-десятичного числа 0111, что мы можем видеть на индикаторе на рис. 2.12,6.

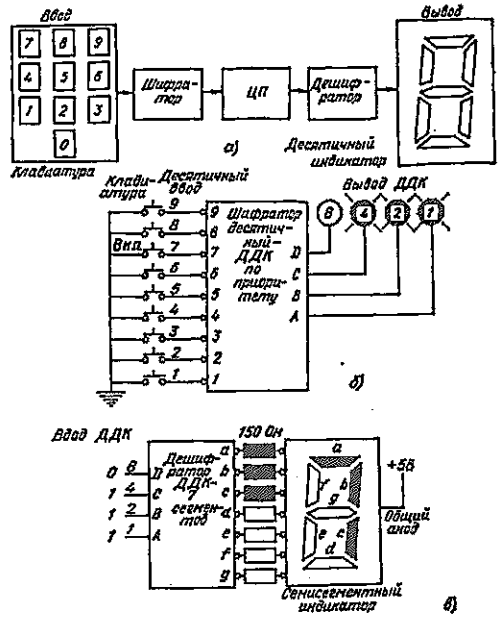


Рис. 2.12. Схемы:
а — функциональная схема калькулятора; б — логическая система клавиатура-шифратор; в—логическая система дешифратор-индикатор.

Большинство шифраторов обладает свойством приоритета. Это означает, что если запрос поступает от двух клавиш одновременно, будут активизированы выходы той из них, которая соответствует более высокой десятичной величине. Совершенно очевидно, что соединения шифратора следовало бы дополнить полной схемой системы питания. В принципе, так и шифратор может выполняться в форме кристалла интегральной схемы, но он мог бы быть составлен и из отдельных элементов (для этого их понадобилось бы около 20).

Рассмотрим систему дешифратор-индикатор, приведенную на рис. 2.12, в. Этот дешифратор двоично-десятичного кода с выводом на семь сегментов переводит 0111 ДДК в его десятичный эквивалент 7, выводимый на семи сегментный индикатор на фотодиодах. Используемый в этой системе индикатор имеет общий анод—все аноды семи диодов (форми-

рующих семь сегментов) подсоединены к выводу +5В общего источника питания. Индикатор на рис. 2.14, *в* имеет L-активные входы, что может быть установлено по наличию кружков инверсии на входах *a*—*g*. Таким образом, для активизации сегмента нужен один L-сигнал. Отметим также, что дешифратор имеет L-активные совместимые выходы.

Семь сопротивлений между дешифратором и индикатором являются элементами защиты, предназначенными для ограничения тока. В примере, приведенном на рис. 2.12, в, активизированы только выходы *a*, *b* и *c* дешифратора (логический 0). Индикатор на этом рисунке имеет только одно соединение с источником +5 В, тогда как шифратор имеет их два, они не приведены на схеме, чтобы ее не усложнять. В действительности шифраторы двоично-десятичный код-индикатор содержат также входы для полного стирания (погасание всех сегментов) и проверки диодов (зажигание всех сегментов).

Контрольные вопросы

1. Шифратор на рис. 2.13 переводит десятичные сигналы с клавишного устройства в ____ (ASCII, двоично-десятичный код).

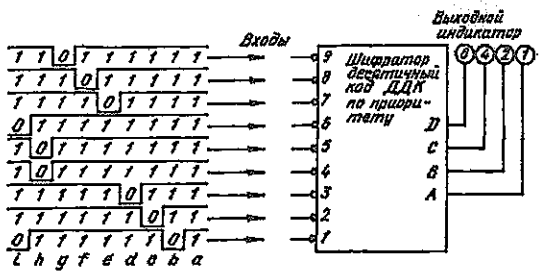


Рис. 2.13.

2. Шифратор на рис. 2.13 имеет ____ (Н-, L-ак-тивные) входы.

3. Перечислить 4-разрядные индикации в двоично-десятичном коде на выходе для каждого из входных импульсов, изображенных на рис. 2.13.

4. Дешифратор на рис. 2.14 позволяет перейти от двоично-десятичного кода к ____ (десятичному, шестнадцатеричному) коду.

5. Входные сигналы дешифратора на рис. 2.14 являются _____ (H-, L-активными), а их выходные сигналы _____ (H-, L-активными).

6. Перечислить десятичные выходные сигналы (показания индикатора) для каждого входного импульса на рис. 2.14.

7. Перечислить активные сегменты для каждого из импульсов на рис. 2.14.

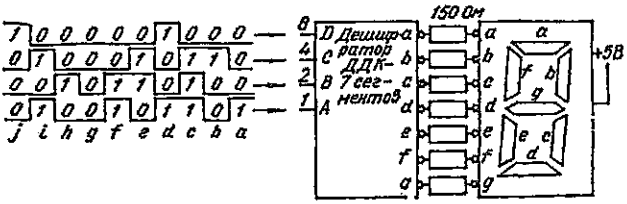


Рис. 2.14.

2.6. Мультиплексоры и демультиплексоры

2.6.1. Мультиплексоры

Мультиплексоры называют также *селекторами данных.* Действие механического коммутатора, показанного на рис. 2.15,а, идентично действию селектора данных электронного мультиплексора.

Вращающийся коммутатор имеет восемь входов и один единственный выход. Вращением механической ручки данные с одного любого входа (*0—7*) могут быть переданы на выход и будут полностью идентичны входным.

На рис. 2.15,б приведена логическая схема мультиплексора-селектора данных на восемь входов. Обозначим их как I_0—I_7 и единственный выход как *Y*. Отметим также, что имеется один L-активный вход активизации *E*. Эти входы и выход представляют собой устройство, полностью аналогичное механическому коммутатору, и мы можем рассматривать вход активизации *E* как общий прерыватель. Внизу на логической схеме мы обозначим выводы управления селекцией данных как S_2, S_1 , S_0. Двоичные данные, поданные на эти входы, определяют, какой из входов данных соединен с выходом.

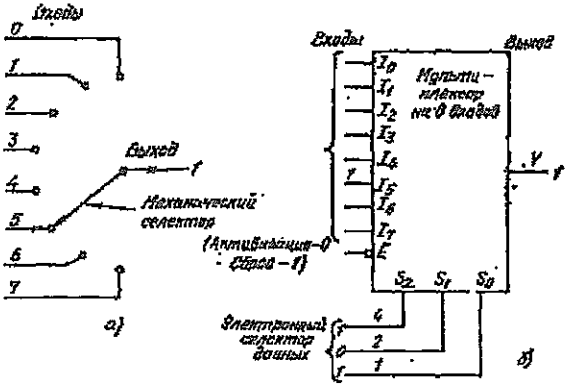


Рис.2.15. Механический коммутатор – аналог мультиплексора/ селектора данных (а) и мультиплексор на восемь вводов (б).

Пример, приведенный на рис. 2.15,6, показывает, что цепь включена или активизирована L-сигналом на входе *E*. На выводы селекции данных S_2, S_1 , S_0 подано 101_2. Этот сигнал избирает вход *5*, что обеспечивает передачу логической 1 с I_5 на вход *Y*. Мультиплексор с восемью входами может быть использован для преобразования одного входного 8-разрядного слова в последовательный эшелон импульсов, переключая 3-разрядный счетчик на вводах селекции данных, что осуществляет последовательную селекцию входов (I_0, I_1, I_2 и т. д.). Мультиплексор-селектор данных используют также для решения сложных логических задач.

2.6.2. Демультиплексор

Демультиплексор, представленный на рис. 2.16, предназначен для выполнения действий, обратных действиям мультиплексора. Де мультиплексор 1x8 обладает одним только входом данных *D* и восемью выходами *(0—7)*. Схема имеет один L-вход активизации и три входа селекции данных.
В примере на рис. 2.16 имеется логическая 1 на входе *D*. Цепь активизируется одним L-сигналом и входы селекции данных избирают выход *5* (101_2). При этих условиях входные данные появляются на выходе *5*. Соединением входов селекции данных с 3-разрядным счетчиком последовательно входящие

данные могут быть распределены на восемь выходов один за другим. Мультиплексоры и демультиплексоры могут быть использованы совместно для преобразования непрерывной информации в форму последовательностей. Мультиплексор будет представлять собой эмиттер, демультиплексор — приемник, который передает данные в их начальной форме.

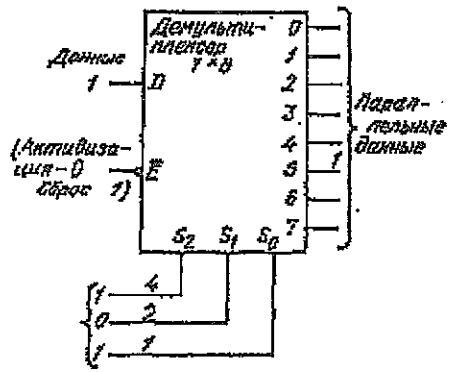


Рис. 2.16. Схема демультиплексора 1х8.

Принцип действия демультиплексора показан на рис. 2.17.

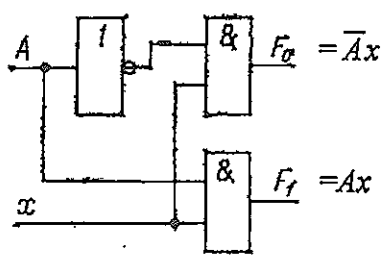


Рис. 2.17.

Демультиплексоры ТТЛ с большим числом выходов работают по тому же принципу, только имеют более сложную схему.

Логическая структура простого демультиплексора вида 1:4 представлена на рис.2.18.

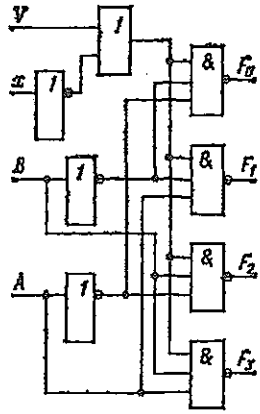


Рис 2.18. ДМЛ на 1:4.

Здесь B и A — адресные входы, *x* - информационный вход, V-разрешающий. Схема функционирует согласно таблице.

Таблица истинности ДМЛ 1:4 (декодера 2:4)

Входы				Выходы			
B	A	*x*	V	F_0	F_1	F_2	F_3
0	0	0/1	0	0/1	1	1	1
0	1	0/1	0	1	0/1	1	1
1	0	0/1	0	1	1	0/1	1
1	1	0/1	0	1	1	1	0/1
0	0	x	1	0	1	1	1
0	1	x	1	1	0	1	1
1	0	x	1	1	1	0	1
1	1	x	1	1	1	1	0

Если у демультиплексора 1:4 на информационном входе поддерживать потенциал U^0 или на разрешающем входе $-U^1$, то прибор будет работать как дешифратор 2:4 .

Таким образом, между обоими типами рассматриваемых устройств нет принципиальной разницы, а различие сводится к виду сигналов на одиночном входе: если они меняются во времени, то это де мультиплексор, если нет - дешифратор. У дешифратора этот вход нередко отсутствует, и выходные сигналы имеют одно, наперёд известное значение. На условных

графических обозначениях у де мультиплексорах в основном поле помещают символ DMX,а дешифраторы обозначают как DC (от анг. Decoder).

Дешифраторы и демультиплексоры служат в качестве коммутаторов-распределителей информационных сигналов и синхроимпульсов, для демультиплексирования данных и организации адресной ячейки в оперативных и постоянных запоминающих устройствах, а также для преобразования двоично-десятичного кода в десятичный с целью управления индикаторными и печатающими устройствами.

Микросхема К155UD3 служит для преобразования четырехразрядного двоичного кода в код «1 из 16 », в зависимости от способа включения может работать как де мультиплексор или как дешифратор.

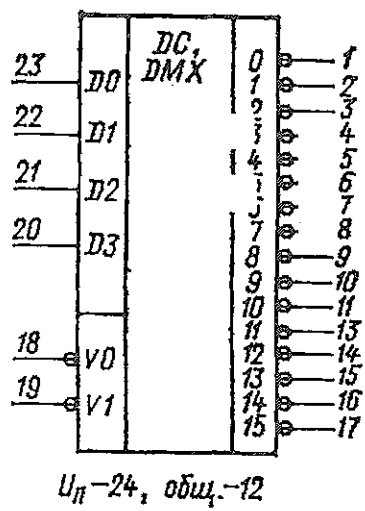


Рис.2.19.

Для создания режима де мультиплексора 1:16 один из выходов V заземляют (т.е. создают уровень V^0), а другой используют в качестве информационного.

Если на обоих разрешающих входах поддерживать уровень V0 = V1 = 0, микросхема работает как дешифратор (четыре входа – 16 выходов).

2.7. Тристабильные элементы

Интегральные элементы семейства ТТЛ (транзисторно-транзисторной логики) используются очень широко. Выход обычного устройства ТТЛ может быть либо логической 1 либо 0. Поэтому невозможно подсоединять выходы стандартных элементов ТТЛ на общую шину микро-ЭВМ (шину данных, например). Были введены специальные элементы ТТЛ, выходы которых могут быть объединены на общей шине. Эти элементы называются *тристабильными* (имеющими три состояния). Им присущи состояние на выходе логического 0, логической 1 и особое состояние высокого сопротивления Z. Когда тристабильный элемент находится в состоянии высокого сопротивления Z, его выход отсоединяется от шины вход активизации, а проходящие через элемент данные не инвертируются. Когда элемент шинного буфера сброшен, его выход находится в состоянии высокого сопротивления (плавает) и не оказывает на шины никакого влияния. В этом состоянии выход элемента отсоединяется от шины, т. е. не выдает на шину и не принимает от нее никакой информации.

Логическая схема элемента шинного буфера представлена на рис. 2.20. Этот буфер обладает тристабильными выходами, и его действие описывается таблицей истинности (табл.2.7). У шинного буфера имеется один L-активный

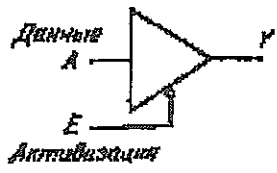


Рис. 2.20. Шинный тристибальный буфер.

Таблица 2.7.

Таблица истинности тристабильного шинного буфера

Функциональное состояние	Входы		Выход
	$\overline{E}$	$\overline{A}$	Y
Активизирован	0	1	0 1
Сброшен	1	0	Высокое сопротивление
	1	1	

Контрольные вопросы

1. Выход элемента ТТЛ может быть либо в состоянии логической 1, либо в состоянии логического 0, либо в состоянии ____ (высокого, низкого) сопротивления.

2. Элемент шинного буфера на рис. 2.21 имеет один вход активизации, активизируемый ____ (H-, L-сигналом).

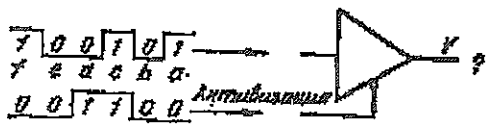


Рис. 2.21.

3. См. рис. 2.21. Перечислите выходы элементов шинного буфера, соответствующие каждой из семи групп импульсов на входе.

2.8. Полупроводниковая память

Триггер (или защелка) является основным элементом памяти, используемым в многочисленных устройствах вычислительной техники. Память в основном делится на две группы: постоянную и оперативную. Сами названия показывают разницу между этими двумя типами памяти. В постоянной памяти нули и единицы располагаются в соответствии с определенной программой разработчика. Оперативная память может быть просто запрограммирована, уничтожена и перепрограммирована пользователем.

Программирование означает процесс *записи* в память. Копирование данных из памяти без разрушения ее содержимого представляет собой операцию *считывания* из памяти. Устройства постоянной памяти называются ПЗУ *(постоянные запоминающие устройства)*, а оперативной— ОЗУ *(оперативные запоминающие устройства)*. Как правило, программы ОЗУ сменяемы; это означает, что они теряются, если прерывается питание ИС даже на короткое время. Типичными являются системы, содержащие оба типа памяти — ПЗУ и ОЗУ. Наиболее часто оба типа памяти являются содержимым различных ИС.

Существуют четыре разновидности ПЗУ. Стандартные ПЗУ программируются разработчиком. Программируемые ПЗУ (ППЗУ) могут быть запрограммированы постоянно пользователем или заготовителем на специальном оборудовании, это программирование необратимо.

Стираемые ППЗУ (СППЗУ) могут быть программируемы и стираемы пользователем. Данные, расположенные в СППЗУ, могут быть стерты ультрафиолетовым облучением большой интенсивности. Стираемые электрически ППЗУ— это другой тип стираемых ППЗУ (СЭППЗУ), операция стирания осуществляется на специальном оборудовании в переменном электрическом поле без применения ультрафиолетового облучения. Рассмотренные ПЗУ, ППЗУ, СППЗУ и СЭППЗУ являются устройствами постоянной памяти и не теряют своего содержимого при прерывании питания ИС.

Группа ОЗУ делится на две подгруппы. Если ОЗУ содержит в качестве элементарных ячеек памяти цепи типа триггеров, оно называется *статическим* ОЗУ (или статической оперативной памятью); *динамические* ОЗУ строятся более просто и основаны на свойствах электрической емкости, но они должны подтверждать содержимое ячеек примерно несколько сотен раз в секунду. Статические ОЗУ не нуждаются ни в каком подтверждении, и их двоичное содержимое сохраняется до тех пор, пока сохраняется питание ИС.

Оперативное запоминающее устройство микро-ЭВМ служит для размещения на определенное время программ и данных пользователя. Постоянное запоминающее устройство используется для размещения команд на машинном языке, которые представляют собой программу-монитор (сокращенно—монитор). Монитор содержит в себе неизменяющиеся подпрограммы инициализации, ввода/вывода и арифметических алгоритмов.

Контрольные вопросы

1. В вычислительной технике сокращение ПЗУ означает _____.
2. В практике оперативная память обозначается сокращением_____ (ОЗУ, ПЗУ).
3. В вычислительной технике сокращение ОЗУ означает _____.
4. Сокращение ППЗУ означает _____.

5. Постоянное запоминающее устройство и ____ (ОЗУ, ППЗУ) являются устройствами размещения данных, сохраняемых постоянно и необратимых.

6. Оперативное запоминающее устройство предназначено для хранения (стираемых, не стираемых) данных.

7. Сокращение СППЗУ означает____ .

8. Сокращение СЭПЗУ означает ____.

9. Электрически стираемые ПЗУ являются устройствами размещения ____ (переменных, постоянных) данных.

10. Устройства оперативной памяти делятся на статические и ____ ОЗУ.

11. Элементарные ячейки памяти ____ (статических, динамических) ОЗУ практически состоят из стандартных триггеров.

12. ____ (статические, динамические) ОЗУ нуждаются в регенерации памяти несколько сотен раз в секунду.

13. Программируемые постоянные запоминающие устройства в микро-ЭВМ могут предназначаться для размещения ____ (программы монитора, временной программы пользователя).

2.8.1. Использование оперативной и постоянной памяти

Организация ПЗУ или ОЗУ может быть наглядно представлена своеобразной таблицей истинности. Таблица 2.8 представляет собой один из возможных вариантов организации ячеек памяти. Здесь речь идет об ОЗУ 16Х4 бит, о чем мы можем сделать вывод, имея 16 4-разрядных групп (эти группы составляют слова памяти). В табл. 2.8 большинство ячеек памяти пусты, за исключением слова 12, которое содержит данные 0101. В действительности пустые. ячейки памяти могут содержать неизвестные сочетания нулей и единиц. На рис. 2.22 представлена логическая схема ОЗУ 16Х4 бит. В этом случае ОЗУ с объемом памяти 64 бит выполняет операцию записи поступающих данных 0101 в ячейку памяти $12_{10}(1100_2)$. На входы данных поступает для записи в память слово 0101_2, а положение слова в ячейке с адресом 12_{10} определено величиной 1100_2 (12_{10}), поступающей на адресные входы.

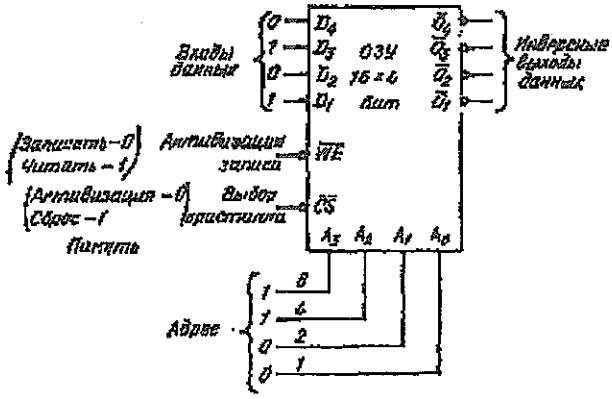


Рис. 2.22. Логическая схема ОЗУ 16х4 бит.

Таблица 2.8.

Структура ОЗУ 16х4 бит

Адрес	Бит D	Бит C	Бит B	Бит A
Слово 0				
Слово 1				
...				
Слово 11				
Слово 12	0	1	0	1
Слово 13				
Слово 14				
Слово 15				

Затем две команды управления *CS* и *W* переводят ОЗУ в состояние записи. Заметим, что входы *активизации записи WE* и *выбора кристалла CS* должны быть в L-состоянии для того, чтобы выполнялась операция записи. Данные 0101_2 помещаются тогда на место слова памяти по адресу 12_{10}, как показано на рис.2.24. Некоторые конструкторы вызывают активизацию памяти по входу управления CS. В табл. 2.9 приведена таблица истинности такого ОЗУ с объемом памяти 64 бит.

Таблица 2.9.

Таблица истинности ОЗУ 16х4 бит

Функциональное состояние	Входы управления		Выходы
	CS	*WE*	0
Запись	0	0	Состояние логической 1
Считывание	0	1	Инверсия размещенных данных
Ожидание	1	*	Состояние логической 1

Примечание: * — не имеет значения.

Оперативное запоминающее устройство находится в состоянии записи, когда две линии управления $\overline{CS}$ и $\overline{WE}$ находятся в L-состоянии. В ходе операции записи 4 бит данных ($\overline{D}_4$, $\overline{D}_3$, $\overline{D}_2$, $\overline{D}_1$) загружаются в ячейки памяти, на которую указывает адрес, и в течение этого времени выходы (O_4, O_3, O_2, O_1) держатся в H-состоянии. Когда входы команд CS=0 и WE=1, ОЗУ находится в состоянии считывания из него данных.

В ходе операции инверсное значение слова данных, на которое указывают адресные входы, появляется на выходах. Данные, расположенные в ОЗУ, не разрушаются операцией считывания. В состоянии ожидания все выходы переходят к H-уровню и никакие данные не проходят через входы *D*.

Контрольные вопросы

1. Оперативное запоминающее устройство с памятью объемом 64 бит, приведенное на рис. 2.23, находится в состоянии _____ (ожидания, записи) во время прохождения импульсов *a*. Следовательно, все выходы находятся в __

(H-, L-состоянии).

2. См. рис. 2.23. Во время прохождения импульсов b, *c*, *d* и *e* ОЗУ находятся в состоянии _____ (записи, чтения).

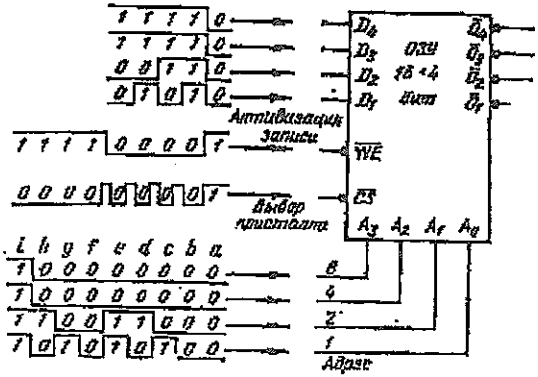


Рис. 2.23.

3. Обратиться к рис. 2.23. В течение импульсов *f, g, h* и *i* ОЗУ находятся в состоянии _____ (записи, чтения).

4. Обратиться к рис. 2.23. Перечислить адреса размещения и данные, помещаемые в ОЗУ, в ходе операции записи в течение импульсов *b c, d, e.*

5. См. рис. 2.23. Перечислить адреса и данные, которые появляются на выходах ОЗУ во время операции считывания, в течение прохождения импульсов *f, g, h, i.*

6. Если ОЗУ, показанное на рис. 2.23, стало бы ПЗУ, какие входы на рисунке следует устранить?

ГЛАВА 3. АРИФМЕТИЧЕСКИЕ ОСНОВЫ ЦИФРОВОЙ И МИКРОПРОЦЕССОРНОЙ ТЕХНИКИ

3.1. Системы счисления

Под системой счисления понимается способ представления чисел с помощью специального набора символов (цифр). Системы счисления делятся на позиционные и непозиционные.

В позиционных системах счисления количественная характеристика каждой цифры зависит от её места расположения (позиции) в числе. Например, в числе 52,25 левая цифра 5 в 1000 раз превосходит аналогичную цифру, расположенную в конце числа.

В непозиционных системах счисления количественное значение цифр не зависит от их места в числе. Например, в числах XIV и IX числа X независимо от их расположения имеют одинаковый «вес» равный 10 единицам.

Всякая позиционная система счисления характеризуется основанием - количеством цифр, принятых для записи чисел. Так, десятичная система (decimal) счисления использует для записи чисел 10 цифр: 0;1;2;3;4;5;6;7;8;9, двоичная две цифры:0;1, восьмеричная - восемь цифр: 0÷7, в шестнадцатеричной (hexadecimal) системе — шестнадцать цифр: 0÷9,A,B,C,D,E,F. В ней буквы A,B,C,D, E, F используются в качестве цифровых знаков таблица 3.1.

Таблица 3.1.

A_{10}	A_2	A_{16}
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Любое число A_p в позиционной системе счисления с основанием p. может быть выражено суммой произведений каждой цифры числа на основании системы счисления в соответствующей их позиции степени:

$$A_p=a_{n-1}\cdot p^{n-1} + a_{n-2}\cdot p^{n-2} + a_{n-3}\cdot p^{n-3} + \ldots + a_{n-n}\cdot p^{n-n} + a_m\cdot p^{-1} + a_{m+1}\cdot p^{-2} + \ldots \quad (1)$$

где, n- количество разрядов числа целой части,

m- количество разрядов дробной части числа.

Например, число 124,69 в десятичной системе счисления может быть представлено как,

$$A_{10}= 1\times10^2 + 2\times10^1 + 4\times10^0 + 6\times10^{-1} + 9\times10^{-2}$$

В двоичной системе счисления число 1111100, 10110 представляется:

$$A_2=1\times2^6+1\times2^5 +1\times2^4 +1\times2^3 +1\times2^2 +0\times2^0 +1\times2^{-1} +0\times2^{-2} + +1\times2^{-3} +1\times2^{-4} + 0\times2^{-5}$$

И, наконец, шестнадцатеричное число 2A5, F4:

$$A_{16}=2\times16^2 + A\times16^1 + 5\times16^0 + F\times16^{-1} + 4\times16^{-2}.$$

Двоичная система счисления. Любая ЭВМ создаётся из электронных элементов, которые могут находиться только в двух состояниях. Одно состояние, когда элемент заперт и через него ток не проходит, а другое — когда элемент открыт и через него идёт ток. Закрытое состояние элемента обозначает 0 low(низкое), а открытое 1 high(высокое). Следовательно, вся информация, которая находится внутри машины, может представляться только этими символами. Значит буквы алфавитов и цифры в машине кодируются через 0 и 1.

Таким образом, в ЭВМ используется двоичная система счисления. Она имеет основание P = 2, которое записывается как 10_2. Для образования двоичных чисел используется только две цифры 0 и 1. Эти двоичные числа называются битами (от английского binary digit). Применение двоичной системы счисления по сравнению с десятичной системой способствует значительному упрощению схем ЭВМ.

Правила выполнения арифметических операций с двоичными числами просты:

Сложение	Вычитание	Умножение
0+0=0	0 − 0 =0	0×0=0
0+1=1	1 − 0 = 0	1×0=0
1+0=1	1 −1 =0	0×1=0
1+1=10	10 − 1=1	1×1=1
	0 − 1= - 1	

Приведем правила десятичного умножения;

Множимое	0	1	0	1
	x	x	x	x
Множители	0	0	1	1
Произведения	0	0	0	1

Два первых правила не требуют никаких пояснений. В двух следующих множителем является 1; когда множителем является 1, при двоичном умножении множимое становится результатом и представляет собой произведение. Когда множитель 0 произведение всегда 0.

Выполним умножение 1101 на 101. Как и с случае умножения десятичных чисел, множимое сначала умножается на число, стоящее в младшем разряде (в рассматриваемом случае − бит в колонке веса 1).

Множимое	1101	13
	x	x
Множитель	101	5
1-е частичное произведение	1101	65
2-е частичное произведение	0000	
3-е частичное произведение	1101	
Конечное произведение	1000001	

Поскольку бит множителя в разряде веса 1 является 1, множимое копируется и составляет первое частичное произведение. Вторым битом множителя является 0, тогда второе

частичное произведение есть 0000 (заметим, что оно сдвинуто на одну позицию влево). Битом разряда веса 4 множителя является 1, тогда для получения третьего частичного произведения снова следует копирование множимого (заметим, что копирование завершается новым сдвигом на одну позицию влево). После этого выполняем сложение трех частичных произведений, что дает результат 1000001_2. Полученный результат $1101_2 \times 101_2 = 1000001_2$ соответствует произведению десятичных чисел $13_{10} \times 5_{10} = 65_{10}$.

Двоично-десятичная система счисления. Эта система счисления иногда используется в ЭВМ для представления десятичных чисел. В двоично-десятичном изображении каждая десятичная цифра представляется четырёхразрядным двоичным числом, называемым тетрадой.

Десятичная цифра	0	1	2	3	4	5	6	7	8
Двоично-десятичный код	0000	0001	0010	0011	0100	0101	0110	1000	1001

При выполнении арифметических действий в 16-ной системе счисления рассмотрим сложение и вычитание, которые выполняются с использованием следующей таблицы сложения:

+	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

Например:	35268	39548
	+91433	- 17058
	C669B	22490

Вопросы для самопроверки. Даны два двоичных и шестнадцатиричных числа, выполнить над ними все арифметические операции.

1)	1110111,1	77,8
	1101,1	3,8
2)	101111101,1	13,8
	11101,1	3,2
3)	10111101,01	54,4
	1101,11	5,2
4)	1111010,1	70,8
	1111,01	5,4
5)	1110101,11	75,2
	1110,01	4,4
6)	1110001,01	71,2
	1011,1	1,8
7)	1011110,01	54,4
	1101,01	3,4
8)	1010101,01	55,4
	1010,11	0,2
9)	110110,01	63,4
	1011,1	1,8
10)	1101011,1	61,8
	1111,01	5,4

3.2. Правила перевода чисел из одной системы счисления в другую

Числа, записанные в десятичной, двоичной и шестнадцатеричной системах счисления могут быть переведены из одной систему счисления в другую по специальным правилам.

Для перевода целого числа из десятичной системы счисления в р -ю нужно разделить число десятичной системы на основание р -ой системы. Полученный остаток будет младшим разрядом искомого числа в р - ой системе. Остаток от следующего деления будет следующим разрядом числа в новой системе счисления и т.д.

Деление продолжаем до получения частного, меньшего, чем основание. Это частное и будет старшим разрядом числа в р -ой системе. Остальные цифры получаются при записи остатков справа налево.

Например:

Число 147_{10} десятичное, необходимо перевести в двоичную и шестнадцатеричную системы счисления. Для него число 147 делим на 2, 16.

$147_{10} \rightarrow A_2$

```
147 | 2
146   73 | 2
  1   72   36 | 2
      1   36   18 | 2
          0   18    9 | 2
              0    8    4 | 2
                   1    4    2 | 2
                        0    2    1
                             0
```

получаем $A_2 = 10010011$ $147_{10} \rightarrow A_{16} = 93$

Для перевода дробных чисел из десятичной системы в р - ю систему вначале целую часть переводят по способу, описанному выше, а дробную часть по следующему способу: дробное число в десятичной системе умножается на основание р - й системы.

Целая часть полученного произведения будет старшим разрядом дроби в новой системе. Затем дробную часть этого произведения снова надо умножить на основание р - й системы.

Целая часть нового произведения будет следующим разрядом дроби в р- системе счисления и т.д. Перевод продолжают до тех пор, пока правая часть не будет состоять из одних нулей или пока не получат требуемое количество разрядов переводимого числа.

Например:

$$147{,}69_{10} \rightarrow A_2$$

0,69
2
I,38
2
0,76
2
I,52
2
I,04 и т.д.

A_2 = 10010011, 1011

Перевод чисел из двоичной и шестнадцатеричной систем счисления в десятичную систему счисления производится по формуле (1) путем разложения чисел по степеням.

Например:

$10010011,1011_2 \rightarrow A_{10}$

$A = Ix2^7+0x2^6+0x2^5+Ix2^4+0x2^3+0x2^2+Ix2^I+Ix2^0+Ix2^{-I}+Ix2^{-2}+Ix2^{-3}+Ix2^{-4}$=I28+I6+2+I+0,5+0,0I25+0,00625=I47,69

$93,B11_{16} \rightarrow A_{10}$

$A = 9xI6^I+3xI6^0+IIxI6^{-I}+IxI6^{-2}+IxI6^{-3}$=I44+3+0,6875+0,003= =I47,69

Перевод чисел из двоичной системы в шестнадцатеричную и наоборот производится согласно таблицы № 3.1.

При переводе двоичной системы в шестнадцатеричную систему пользуются тетрадами, т.е. каждую четверку двоичных чисел заменяют эквивалентной ей 16-ной цифрой.

Например: 1001 0011, 1011

9 3 , B

A_{16}= 93,B

Обратный перевод осуществляют в обратной последовательности, как видно из таблицы № 3.I цифре 9 с шестнадцатеричной системе соответствует 1001 в двоичной системе, цифре 3 соответствует число 0011, букве В соответствует 1011 в итоге получаем

10010011, 1011

Вопросы для самопроверки:

Переведите числа из одной системы счисления в другую:

1) 1352,45
2) 2AB.3C
3) 1101011, 10101
4) 540,35
5) 447,28
6) 1111010, 101
7) 232,67
8) 45,47
9) 4573,725
10) 100010111,101101
11) 347,48
12) 3 Г, АВ

Контрольные вопросы

1. Какое отличие между позиционными и непозиционными системами счисления?
2. Чем вызвано применения двоичной системы счисления?
3. Какова роль шестнадцатеричной системы счисления?
4. Как переводят целую часть десятичного числа в другие системы счисления?
5. Как переводят дробную часть десятичного числа в другие системы счисления?
6. Как переводят числа из любых систем счисления в десятичную?
7. Двоичная система счисления и шестнадцатеричная имеют кратные основание — $2^4 = 16$, связано ли это с тем, что при переводе чисел в этих системах один разряд шестнадцатеричного числа заменяют четырьмя разрядами двоичного?
8. Как можно перевести число из двоичной системы счисления в шестнадцатеричную и наоборот?
9. Выполнить следующие сложения двоичных чисел:

а) 1010 + 0101 б) 1101 + 0101 в) 01011011 + 00001111 г) 00111111 + 00011111

10. Выполнить следующие вычитания двоичных чисел:

а) 1110 - 1000 б) 1010 - 0101 в) 01100110 - 00011010 г) 01111000 - 00111111.

11. Первое число при умножении называется ___, второе—множителем, а результат составляет ____.

12. Выполнить следующие умножения двоичных чисел:

а) 1001	б) 1101	в) 1111	г) 1110
X	X	X	X
11	1001	101	1110.

3.3. Дополнительный код

Сама ЭВМ обрабатывает информацию обычно в двоичном коде. Однако если нужно использовать числа со знаком, используется специальный *дополнительный код,* что упрощает аппаратные средства ЭВМ.

На рис. 3.1, *а* приведено обычное изображение регистра МП или ячейки памяти вне МП. Такой регистр представляют пространством из 8 бит данных. Позиции бит пронумерованы от 7 до 0, а веса двоичных позиций указаны в основании регистра, бит 7 имеет вес 128, бит 6—64 и т.д.

На рис. 3.1,*б* и *в* показаны типовые структуры 8-разрядных регистров для размещения *чисел со знаком.* В обоих случаях бит 7 является знаковым. Он указывает, является ли число положительным (+) или отрицательным (-). При 0 в знаковом бите число положительно, при 1 — отрицательно.

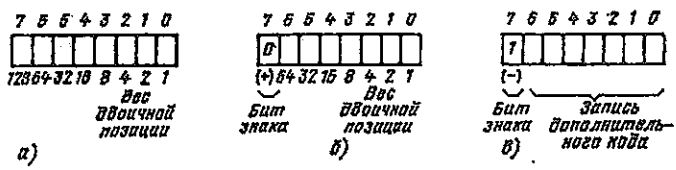


Рис. 3.1. Изображение регистра МП или ячейки памяти: а—расположение двоичных позиций; б—идентификация положительных чисел нулем в знаковом бите; в — идентификация отрицательных чисел единицей в знаковом бите

Если, как показано на рис. 3.1,б, число положительно, оставшиеся ячейки памяти (6—0) содержат двоичное 7-разрядное число. Например, если регистр на рис. 3.1,б содержит 0100 0001, это соответствует числу $+65_{10}$ (64 + 1, знаковый бит положителен). Если в него записано 0111 1111, содержимым будет $+127_{10}$ (знаковый бит положителен: +64

+32 +16 +8 +4 +2 +1), что является наибольшим положительным числом, которое может содержать 7-разрядный регистр.

Если, как это показано на рис. 3.1, в, регистр содержит то же число со знаком, но отрицательное, он будет содержать дополнительный код этого числа. В табл. 3.2 приведена запись в дополнительном коде положительных и отрицательных чисел. Заметим, что все положительные числа имеют 0 в старшем бите, остальные биты составляют двоичное число. Все отрицательные числа имеют 1 в старшем разряде. Рассмотрим строку +0 в табл. 3.2.: запись в дополнительном коде +0 будет 0000 0000. В ближайшей нижней строке видим, что запись в дополнительном коде—1 следующая: 1111 1111. Рассмотрим пошаговое перемещение в обратном направлении от 0000 0000 до 1111 1111.

Какой будет запись в дополнительном коде числа —9? Рассмотрим этапы преобразования. Они следующие:

Десятичное число 9

Этап 1. Запись десятичного числа без знака (9)

Этап 2. Преобразование десятичного числа в двоичный код (0000 1001)

Этап 3. Получить обратный код двоичного числа заменой нулей единицами, а единиц - нулями (1111 0110)

Этап 4. Прибавить единицу к обратному коду.
Здесь прибавить 1 к
1111 0110, что дает
1111 0111

Полученный результат является дополнительным кодом положительного десятичного числа. В приведенном примере дополнительным кодом числа 9 является 1111 0111. Заметим, что знаковый бит — 1, это означает, что рассматриваемое число (1111 0111) отрицательно.

Каким будет десятичный эквивалент числа 1111 0000, записанного в форме дополнительного кода? Процедура преобразований в этом случае следующая:

Дополнительный код	1111 0000
Дополнение до 1	0000 1111

Этап 1. Запись дополнительного кода (1111 0000).

Этап 2. Получается обратный код дополнительного кода заменой нулей единицами, а единиц—нулями (0000 1111)

Этап 3. Добавить 1

```
  00001111
+        1
  --------
  00010000 = 16
```

Таблица 3.2.

Десятичные числа со знаком и их представление в дополнительном коде

Десятичные	Представление чисел со знаком	Примечания
+127	0111 1111	
.	.	
.	.	
.	.	
+8	0000 1000	Положительные числа представлены в той же форме, что и прямые Двоичные числа
+7	0000 0111	
+6	0000 0110	
+5	0000 0101	
+6	0000 0100	
+3	0000 0011	
+2	0000 0010	
+1	0000 0001	
+0	0000 0000	
-1	1111 1111	
-2	1111 1110	Отрицательные числа представлены в форме дополнительного кода
-3	1111 1101	
-4	1111 1100	
-5	1111 1011	
-6	1111 1010	
-7	1111 1001	
-8	1111 1000	
.	.	
.	.	
.	.	
-128	1000 0000	

Таким образом, формирование обратного кода и добавление 1 являются теми же процедурами, которые мы проводили при преобразовании двоичного числа в дополнительный код. Однако следует отметить, что, хотя мы получили двоичное число 0001 0000 = 16_{10}, исходная запись дополнительного кода 11110000= -16, т.е. имеем отрицательное число, поскольку старший бит в дополнительном коде является 1.

Контрольные вопросы

1. Когда числа со знаком помещаются в 8-разрядный регистр микропроцессора, старший (7-й) бит называется

2. Установить, являются ли следующие числа в дополнительном коде положительными или отрицательными:

а) 0111 0000; б) 1100 1111; в) 1000 1111; г) 0101 0101.

3. Используя табл. 3.2. дать дополнительный код следующих десятичных чисел со знаком: а) +1; б) +5; в) +127; г) -1; д) -2; е) -128.

4. Используя процедуру, приведенную в § 3.3, дать дополнительный код следующих десятичных чисел со знаком: а) —10; б) —21; в) —34; г) —96.

5. Расположение бит в дополнительном коде _____ (в ДДК, в двоичном коде) одинаково для положительных двоичных чисел.

6. Используя процедуру, приведенную в § 3.3, дать десятичные эквиваленты следующих чисел в дополнительном коде: а) 1111 1011; б) 0000 1111; в) 1000 1111; г) 0111 0111.

3.4. Арифметика в дополнительном коде

Микропроцессор может использовать числа в форме дополнительного кода, потому что он в состоянии выполнять операции *дополнения (инверсии), инкрементирования* (добавления 1 к числу) и *сложения* двоичных чисел. Микропроцессор не приспособлен для прямого вычитания. Он использует сумматоры и для выполнения вычитания оперирует над дополнительным кодом.

Сложим десятичные числа +5 и +3. Рассмотрим процедуру действии в случае одновременного сложения чисел в десятичном и в дополнительном кодах:

1-е число	(+5)	0000 0101
	+	+

2-е число (+3) 0000 0011
(+8) 0000 1000

Согласно табл. 3.2. +5 =0000 0101 в дополнительном коде аналогично +3= 0000 0011. Тогда числа в дополнительном коде 0000 0101 и 0000 0011 складываются, как обычные двоичные числа, давая сумму 0000 1000 в дополнительном коде, т.е. 0000 1000= $+8_{10}$.

Пусть надо сложить десятичные числа +7 и —3. Согласно табл. 3.2. +7=0000 0111 и -3=1111 1101 соответственно в дополнительном коде. Они затем складываются, как обычные двоичные числа, и результат 1 0000 0100 получается в дополнительном коде:

1-е число (+7) 00000111
\+ +
2-е число (-3) 11111101
(+4) 100000100
Пренебречь переполнением.

Старший бит является переполнением 8-разрядного регистра, и им можно пренебречь. Получаем сумму 0000 0100 или $+4_{10}$.

Сложим десятичные числа +3 и —8. Согласно все той же табл. 2.10 +3=0000 0011 и —8= 1111 1000. Их дополнительные коды 0000 0011 и 1111 1000 складываются, как обычные двоичные числа, что дает 1111 1011= -5_{10}:

1-е число (+3) 00000011
\+ +
2-е число (-8) 11111000
(-5) 11111011

Сложим десятичные числа —2 и —5. В дополнительном коде согласно табл. 3.2.—2=1111 1110 и—5=1111 1011. Два числа 1111 1110 и 1111 1011 складываются, как обычные десятичные числа, что дает 1 1111 1001:

1-е число (-2) 11111110
\+ +
2-е число (-5) 11111011
(-7) 111111001
Пренебречь переполнением.

Старший бит результата является переполнением 8-разрядного регистра, и им пренебрегаем. Таким образом, суммой двух чисел 1111 1110 и 1111 1011 в дополнительном коде будет 1111 1001. Согласно табл. 2.10 сумма 1111 1001= -7_{10}.

Вычтем теперь десятичное число +5 из десятичного числа +8. Первое число +8=0000 1000, второе +5=0000 0101. В дополнительный код (инвертировать и добавить 1) должно быть преобразовано число 00000101, что дает 1111 1011. Затем первое число 0000 1000 складывается с дополнительным кодом второго 11111011, как с обычным двоичным числом, что дает 1 0000 0011:

1-е число (+8) 00001000

2-е число (+5)=0000 0101 Дополнительный код 11111011

(+3) 100000011

Пренебречь переполнением.

Старший бит является переполнением регистра, им пренебрегаем, что дает результат 0000 0011= $+3_{10}$. Заметим, что второе число было представлено в дополнительном коде, затем сложено с первым. *Используя дополнительный код и сумматор, микропроцессор выполняет вычитание.*

Вычтем теперь большее десятичное число +6 из десятичного числа +2:

1-е число (+2) 00000010

2-ечисло (+6)=00000101 Дополнительный код 11111010

(-4) 11111100

Пренебречь переполнением.

Дополнительный код первого числа +2=0000 0010 второе число +6=0000 0110, его дополнительный код (инверсия и добавление 1)—1111 1010. Оба эти кода сложены затем, как обычные двоичные числа, что дает 1111 1100, а согласно табл. 2.10 1111 1100= - 4_{10}.

Контрольные вопросы

1. Сложить следующие десятичные числа со знаком, используя метод дополнительного кода:

а) (+7) б) (+31)

\+ +

(+1) (+26).

2. Сложить следующие десятичные числа со знаком, используя метод дополнительного кода:

а) (+8) б) (+89)
+ +
(-5) (-46).

3. Сложить следующие десятичные числа со знаком, используя метод дополнительного кода:

а) (+1) б) (+20)
+ +
(-6) (-60).

4. Сложить следующие десятичные числа со знаком, используя метод дополнительного кода:

а) (-3) б) (-13)
+ +
(-4) (-41).

5. Вычесть следующие десятичные числа со знаком, используя метод дополнительного кода:

а) (+7) б) (+113)
- -
(+2) (+50)

6. Вычесть следующие десятичные числа со знаком, используя метод дополнительного кода:

а) (+3) б) (+12)
- -
(+8) (+63).

3.5. Буквенно-цифровой код

Когда микро-ЭВМ взаимодействует с телетайпом или видеотерминалом, необходимо прибегать к коду, который одновременно включает в себя числовые и алфавитные знаки. Такие коды называются буквенно-цифровыми.

Наиболее распространен буквенно-цифровой код ASCII (произносится АСКИ)—стандартный американский код обмена информации.

В табл. 3.3. приведена выдержка 7-разрядного кода ASCII. В этот список входят 7-разрядные коды цифр, прописных букв и знаков пунктуации. Полный код ASCII включает кодирование строчных букв и признаков команд.

Таблица 3.3.

Выдержка из алфавитно-цифрового кода ASCII

Символ	Код ASCII	Сим-вол	Код ASCII	Сим вол	Код ASCII
Массив	010 0000	0	011 0000	I	100 1001
!	010 0001	1	011 0001	J	100 1010
«	010 0010	2	011 0010	K	100 1011
#	010 0011	3	011 0011	L	100 1100
$	010 0100	4	011 0100	M	100 1101
%	010 0101	5	011 0101	N	100 1110
&	010 0110	6	011 0110	O	100 1111
,	010 0111	7	011 0111	P	101 0000
(	010 1000	8	011 1000	Q	101 0001
)	010 1001	9	011 1001	R	101 0010
*	010 1010	A	100 0001	S	101 0011
+	010 1011	B	100 0010	T	101 0100
,	010 1100	C	100 0011	U	101 0101
-	010 1101	D	100 0100	V	101 0110
.	010 1110	E	100 0101	W	101 0111
/	010 1111	F	100 0110	X	101 1000
		G	100 0111	Y	101 1001
		H	100 1000	Z	101 1010

Контрольные вопросы

1. Двоичный код, используемый обычно для кодирования цифр и букв, называется _____ кодом.

2. Нуль в коде ASCII представляется как 011 0000, 9 — как _____.

3. Если маскировать три старших разряда цифр от 0 до 9 в коде ASCII, что оставит маска в итоге?

ГЛАВА 4. ОСНОВЫ МИКРОПРОЦЕССОРНОЙ ТЕХНИКИ

Когда программист воспринимает информацию о новом типе МП, он должен выяснить следующие вопросы: архитектура МП; система команд; простейшие системы, использующие данный МП; сигналы управления; назначение выводов.

Понятие архитектуры относится к организации регистров центрального устройства, числа бит шин адресов, данных и т. д.

Система команд — это список операций, которые МП может выполнить. Она включает в себя передачу данных, арифметические и логические операции, команды тестирования данных и ветвлений, операции ввода/вывода (ВВ). В то же время команды используют различные способы адресации.

Схема, представляющая простую информационную систему, показывает, как соединены различные устройства с МП. Простая система может содержать МП, генератор тактовых импульсов (ГТИ), ОЗУ, ПЗУ, порты ВВ, дешифратор адресов и источник питания. Иногда эти функции выполняются ИС или различными прочими составляющими; однако многие микропроцессорные устройства содержат большинство названных элементов.

Сигналы управления являются выходными и управляют другими ИС (ОЗУ, ПЗУ и портами ВВ). Некоторые сигналы могут управлять операциями чтения/записи в памяти или чтения/записи в устройствах ВВ (УВВ).

Наконец, изучение выводов каждой ИС обеспечивает дополнительную информацию о специальных входах и выходах МП. Среди прочих выводов мы найдем выводы питания, ГТИ, ввода-вывода последовательных данных, входов прерываний и управления шинами.

4.1. Архитектура простой микро-ЭВМ

На рис. 4.1 приведена архитектура простои микро-ЭВМ. Микропроцессор является центром всех операций. Ему необходимы питание и тактовые импульсы. Генератор тактовых импульсов может быть отдельным устройством или входить в состав кристалла МП. Типовой МП может содержать 16 адресных линий, которые составляют *однонаправленную шину адресов,* а также обычно восемь линий, которые составляют *двунаправленную шину данных.*

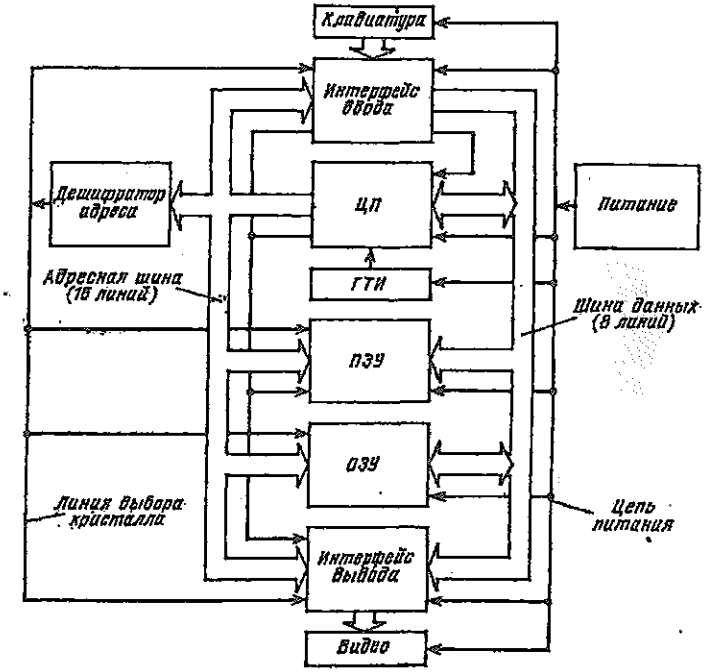


Рис. 4.1. Архитектура микро-ЭВМ.

Архитектура, представленная на рис. 4.1, различает два типа полупроводниковой памяти, используемой системой. Постоянное запоминающее устройство ПЗУ представляет собой постоянную память, которая содержит программу-монитор системы. Оно содержит адресные входы, а также

входы активизации только чтения и выбора кристалла, а также тристабильные выходы, подсоединяемые на шины. Очевидно, ПЗУ имеет также подсоединение питания, которое на схемах обычно не показывается.

Архитектура на рис. 4.1 содержит ОЗУ, т. е. устройство временного размещения данных. В него входят адресные входы, а также входы выбора кристалла и активизации чтения/записи. Это ОЗУ имеет восемь выходов с тремя состояниями, подсоединенных к шине данных. Здесь показан также *источник питания.*

Приведенная система микро-ЭВМ использует клавишное устройство ввода. На этой схеме показаны цепи питания, а также соответствующие линии данных, связанные со специальными ИС—*интерфейсом ввода с клавиатуры.* В задачу интерфейса входит размещение данных и управление их вводом с клавиатуры. В нужный момент интерфейс клавиатуры *прерывает МП* по специальной линии прерывания. Сигнал прерывания заставляет МП:

1) закончить выполнение текущей команды; 2) поддерживать свою нормальную работоспособность; 3) перейти к выполнению специальной группы команд в своем мониторе, по которым ведется управление вводом данных, исходящих с клавишного устройства. Система интерфейса с клавиатурой снабжена адресными входами, линиями выбора кристалла и команд активизации устройства. Активизированное один раз устройство интерфейса с клавиатурой передает данные, поступающие с клавишного устройства на шину данных, микропроцессор их принимает. Если тристабильные выходы интерфейса не активизированы, они возвращаются в свое состояние высокого сопротивления.

Приведенная на рис. 4.1 микро-ЭВМ имеет в качестве выхода группу *семисегментных индикаторов.* Индикаторы запитаны от источника, показанного на схеме справа. Система или специальная ИС *интерфейса вывода на индикаторы* служит для размещения данных и управления состоянием индикаторов. При активизации этого интерфейса по адресной шине, линиям выбора кристалла и активизации он принимает данные, поступающие с шины данных, и размещает их, а также управляет индикатором, на котором размещенные данные высвечиваются.

Адресная линия содержит 65 536 различных сочетаний О

и 1 (2^{16}). Линии адресной шины могут быть подсоединены ко многим устройствам, таким, как ОЗУ, ПЗУ, другие интерфейсы. Для того чтобы активизировать (включить в работу) требуемое устройство, дешифратор адреса считывает данные с адресной шины. Комбинационной логикой дешифратора адреса активизируется линия выбора соответствующего кристалла, активизируя, таким образом, выбранное устройство. Заметим, что для упрощения схемы все 16 линий адресной шины не показываются.

Контрольные вопросы

Все упражнения этого раздела связаны с рис. 4.1. Настоятельно рекомендуем обращаться к нему постоянно.

1. Центром всех операций управления микро-ЭВМ является____ (МП, ОЗУ, ПЗУ).
2. Шина ____ (адреса, данных) является однонаправленной.
3. Посредством 16 линий адресной шины можно получить доступ к____ (16384, 65536) ячейкам памяти и портам ВВ.
4. Назвать по меньшей мере три типа выходов МП.
5. Назвать по меньшей мере четыре типа входов МП.
6. Обычно ПЗУ содержит программу ____ (монитора, изменяемую).
7. Назвать по меньшей мере четыре входа ПЗУ.
8. Назвать выходы ПЗУ.
9. Назвать по меньшей мере четыре типа входов ОЗУ.
10. Назвать выходы ОЗУ.
11. Назвать по меньшей мере пять типов входов специального элемента интерфейса клавиатуры.
12. Назвать по меньшей мере два типа выходов элемента интерфейса клавиатуры.
13. Какова роль выхода прерываний элемента интерфейса клавиатуры?
14. Какие действия предпринимаются МП, когда линия прерываний активизируется клавишным устройством?
15. Назвать по меньшей мере пять типов входов специального элемента интерфейса индикатора.
16. Назвать выходы интерфейса индикатора.
17. Какова роль дешифратора адреса?

4.2. Структура простейшей памяти

Запись в память или считывание из нее происходит при наличии доступа в память. Обычно память выполняется с *последовательным* или *произвольным* доступом. Последовательный доступ означает, что к требуемым данным нужно последовательно пройти через всю память, расположенную до размещения искомых данных.

В случае произвольного доступа данные могут быть записаны в любую ячейку памяти или считаны из нее за определенное фиксированное время, называемое *временем доступа в память*. Оперативные и постоянные запоминающие устройства микро-ЭВМ являются устройствами памяти с произвольным доступом, существенно более быстродействующими, чем устройства с последовательным доступом.

Изучаемый тип микро-ЭВМ обладает адресной шиной из 16 линий, которые могут обеспечить 65536 (2^{16}) различных комбинаций 0 и 1. На рис. 4.2 приведено множество двоичных комбинаций. Обычно принято двоичный адрес представлять в шестнадцатеричной форме. Как видно из рис. 4.2, 0000 0000 0000 0000_2=0000H (0000_{16}).

Напомним, что здесь H указывает на то, что речь идет о шестнадцатеричной системе счисления (H-код). Наиболее значимым адресом на рис.4.2 будет 1111 1111 1111 $111\ 1_2$ или FFFFH.

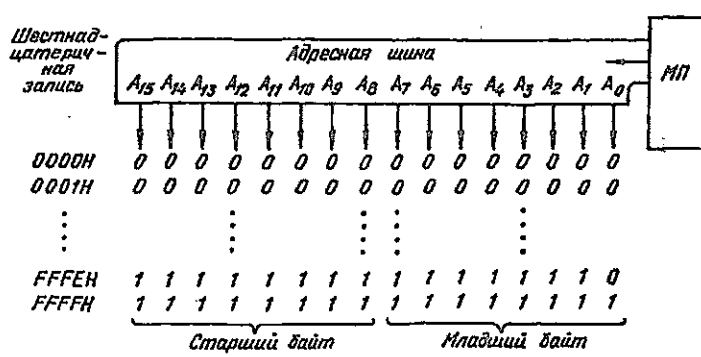


Рис. 4.2. Адресная система микро-ЭВМ.

Схема на рис. 4.3 представляет собой воображаемую память микро-ЭВМ. Адресная шина микропроцессора 16-

разрядная, она может сформировать 65536 (0000—FFFFH) индивидуальных адресов (некоторые из них представлены в шестнадцатеричной записи слева на рис. 4.3). В случае этой специальной микро-ЭВМ первые 256 (00—FFH) ячеек памяти являются содержимым ПЗУ с объемом памяти 256x8 бит (т. е. 256 слов по 8 бит). Если адресная шина формирует адрес 0000H, ПЗУ передает программированную комбинацию из 0 и 1, содержащуюся в постоянной памяти (слово 1100 0011).

Для людей, начинающих работать в области микропроцессорной техники, удобно, когда 256 (00—FFH) первых бит или слов данных в ПЗУ помещены на странице 00H. Заметим, что номер страницы равен представлению в шестнадцатеричном коде старшего байта адреса (рис. 4.3). При этих условиях, манипулируя данными на странице 00H, рассматриваются только две шестнадцатеричные цифры младшего разряда адреса. Некоторые микропроцессоры используют упрощенные команды для доступа к ячейкам памяти первой страницы памяти.

Воображаемая элементарная память на рис. 4.3 показывает, что страницы от 01H до 1FH (адреса от 0100H до 1FFFH) в этом отдельном случае не содержат данных в памяти. Получение доступа в эту открытую зону повлечет непредвиденный результат, поскольку она не содержит ни определенных данных, ни программы. Оперативная память для чтения-записи расположена на странице 20H (рис. 4.3). Устройство размещения данных—это ОЗУ 256x8 бит.

Составители этой системы могли бы расположить это ОЗУ на любой другой странице Доступ к страницам 21H—FFH в этом случае вызовет появление непредвиденного результата по мере вхождения в эту зону, здесь не существует никакой определенной информации.

Если понадобится записать данные в ячейку памяти 2000H (рис. 4.3), восемь элементарных ячеек памяти в начале ОЗУ будут заполнены сочетанием нулей и единиц, поступающих с шины данных. Если микропроцессор захочет затем считать из ячейки памяти по адресу 2000H, в ОЗУ будет прочитано то же сочетание нулей и единиц.

Физически ОЗУ должно быть, очевидно, составлено единственной ИС. Но получается, однако, что очень часто ОЗУ строятся иначе. На рис. 4.3 заштрихованная часть размещения данных в ОЗУ разделена посредине вертикально.

Это указывает на то, что в этом случае зона размещения данных на странице 20Н физически состоит из двух различных ИС. Здесь для реализации зоны памяти 256Х8 на странице 20Н микро-ЭВМ использованы два ОЗУ 256Х4 бит.

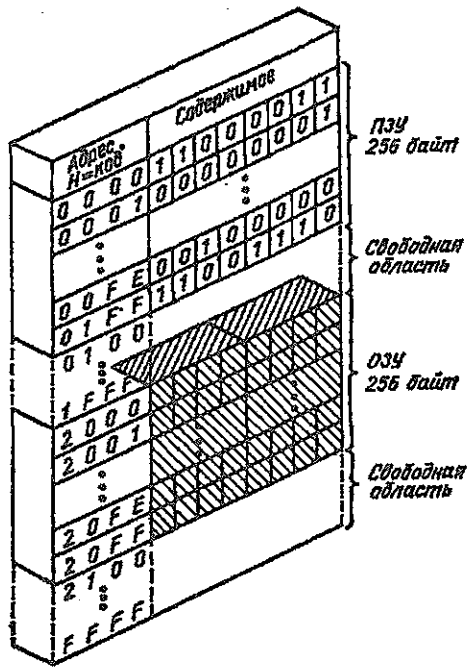


Рис. 4.3. Упрощенное представление памяти.

На рис. 4.4 приведены три варианта реализуемых устройств ИС полупроводникового ОЗУ. На рис. 4.4, *а* единая ИС организована в ОЗУ 256Х8 бит, на рис. 4.4, б две ИС составляют такую же зону размещения данных и, следовательно два ОЗУ 256Х4 бит позволяют составить слово из 8 бит. Наконец восемь ОЗУ на рис. 4.4, *в* формируют также устройство размещения для записи-чтения данных 256х8 бит.

Устройство, представленное на рис. 4.4, в, широко используется для составления обширной памяти. Обычно используют ИС ОЗУ 1024Х1, 4096Х1, 16384х1 бит. Используя способ, показанный на рис. 4.4, в, восемь ОЗУ 4096х1 бит

будет достаточно для формирования оперативной памяти 4096Х8 бит. Отметим, что в этом типе конфигурации такие ОЗУ будут активизированы одной и той же линией выбора кристалла, исходящей из дешифратора адресов.

В системе микро-ЭВМ объем памяти (или память 4096х8 бит) составляет 4 Кбайт (4 К=4096_{10} байт памяти).

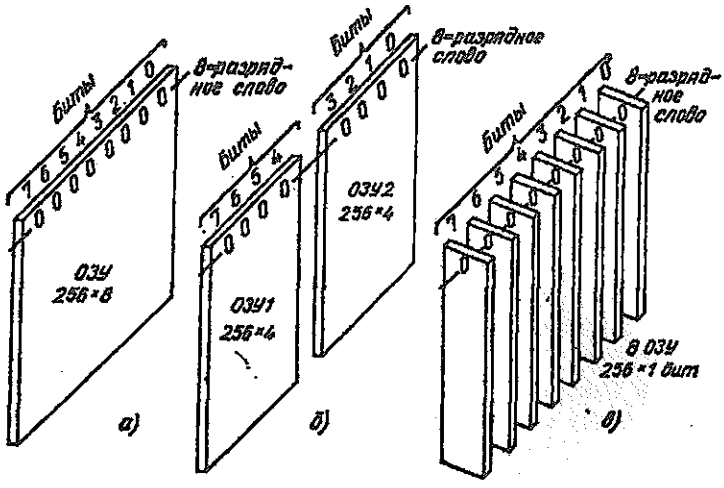


Рис. 4.4. Три способа формирования памяти микро-ЭВМ:
а—на одном кристалле; б—на двух кристаллах;
в—на восьми кристаллах.

Контрольные вопросы

1. Запись в ячейки памяти или считывание из них представляет собой_____ (доступ в, поиски) память.

2. Оперативное и постоянное запоминающие устройства, используемые в микро-ЭВМ, являются примерами памяти с _____ (произвольным, последовательным) доступом.

3. См. рис. 4.2. Зная адрес, выставленный на адресную шину, 0010 0000 0000 0000, определить какая ячейка памяти (в шестнадцатеричном коде) доступна микропроцессору?

4. См. рис. 4.3. Зная, что адресуется ячейка памяти микро-ЭВМ 0001Н, _____ (ПЗУ, ОЗУ) выдаст _____ (привести 8 бит).

5. См. рис. 4.3. Что появится на шине данных, если МП считывает содержимое ячейки памяти FFFFH?

6. См. рис. 4.3. Зная, что адресуется ячейка памяти 2001 Н микро-ЭВМ и что элемент памяти находится в состоянии записи, какой ____ (байт, тетрада) будет записан в ____ (ОЗУ, ПЗУ)?

7. На рис. 4.3 ____ (ОЗУ, ПЗУ) состоит из единственной ИС.

8. Оперативное запоминающее устройство на рис. 4.3 расположено на странице ____.

9. Восемь ИС ОЗУ 16 384x1 бит могли бы быть составлены как ИС на рис. 4.4, *в* для того, чтобы образовать память 16 384x8 бит, что составит____ К.

10. См. рис. 4.3. Какая наибольшая емкость памяти может быть адресована 16 адресными линиями?

4.3. Состав команд

Группа команд, которые может выполнять данный МП, называется его *составом команд*. Состав команд МП может содержать как малое число (восемь), так и большое число (200) основных команд. Составы команд не являются нормализованными. Это неудобство связано как с индивидуальным подходом к разработке, так и с различиями архитектуры и назначений МП.

Имеется много способов классификации команд одного состава. В этой главе согласно нормативам, предложенным научным обществом инженеров-электронщиков, мы изучим следующие команды: арифметические, логические, передачи данных, вызова подпрограмм, возврата из подпрограмм, прочие.

Элементарный МП будет представлен следующим составом арифметических команд: сложение, вычитание, инкрементирование, сравнение, отрицание.

Некоторые конкретные МП могут обладать другими арифметическими командами, такими, как сложение с переносом, вычитание с заемом, умножение и деление.

Элементарный МП наделяется следующими логическими командами: И, ИЛИ, ИЛИ ИСКЛЮЧАЮЩЕЕ, НЕ (отрицание), сдвиг вправо, сдвиг влево.

Некоторые МП, кроме того, наделены такими логичес-

кими командами, как арифметический сдвиг вправо, циклические сдвиги вправо и влево, циклические сдвиги вправо и влево с переносом и тестирование.

Наш же элементарный процессор всегда наделяется *командами передачи данных:* загрузки, размещения, перемещения, ввода, вывода.

Более сложный состав будет содержать команды обмена, сброса и инициализации.

Что касается *команд ветвления,* они следующие: безусловный переход; переход, если нуль; переход, если не нуль; переход, если равенство; переход, если неравенство; переход, если положительно; переход, если отрицательно.

Другие команды условных переходов, имеющиеся в некоторых микропроцессорах, могут зависеть от таких условий, как: больше или меньше, сдвиг или нет, переполнение или нет. Команды ветвления являются командами принятия решений.

Элементарный микропроцессор будет наделен командой *вызова подпрограммы,* (обычно CALL—вызов), чтобы программа могла перейти к специальной группе команд, которые решают поставленную задачу. Все МП обладают командой безусловного вызова, а некоторые наделены командой условного вызова, например, CALL, если нуль; CALL, если не нуль; CALL, если положительно или не положительно, и т. д.

В конце выполнения подпрограммы МП должен иметь возможность возврата в точку отправления из начальной программы. Эта операция выполняется *командой возврата. Эта* команда обычно безусловна, но некоторые МП снабжены и условным возвратом.

Наконец, *прочими командами* элементарного МП будут: нет операции, поместить в стек, выйти из стека, ожидание, останов.

Возможны и другие команды: прерывания активизации или сброса, останова, десятичной коррекции.

Пользователи микропроцессорных систем встречаются с многочисленными способами выражения одних и тех же команд. Из этого множества приведем команду сложения двух чисел для микропроцессора Motorola 6800. Имя команды является *активной формой глагола* и называется операцией. В табл. 4.1 выполняемой операцией является сложение.

Таблица 4.1.

Описание команды ADD

Операция	Мнемоника	КОП	Символика (все обозначения регистров являются их содержимым)
Сложить	ADD M	8BH	A + M →A

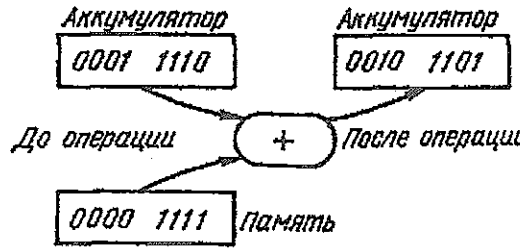


Рис. 4.5. Команда ADD.

Пользователи работают часто с сокращенными формами выражения операции, мнемоническими. В табл. 4.1 ADD M (отметим, что мнемоники всегда записываются большими буквами). Регистр команд и схема декодирования понимают только язык нулей и единиц. *Код операции* (КОП) является шестнадцатеричным представлением 8-разрядного двоичного кода, который заставляет МП выполнить эту команду. В табл. 4.1 КОП для МП Motorola 6800 для ADD M будет 8BH ($1000\ 1011_2$). В колонке символики в табл. 4.1 показывается, что содержимое памяти (M) складывается с содержимым аккумулятором (A) в МП, стрелка указывает, что результат помещается в аккумулятор A.

На рис. 4.5 приведен пример использования команды сложения ADD M. Содержимое (A) аккумулятора (00011110) складывается с содержимым (M) памяти (00001111), сумма (00101101) помещается в аккумулятор (справа). Заметим, что содержимое ячейки памяти не изменилось, тогда как содержимое аккумулятора стало другим.

Контрольные вопросы

1. Группа операций, которые может выполнить МП, представляет его ____.

2. Перечислить семь групп состава команд МП.

3. Операция ADD выполняется по команде арифметических действий, тогда как операция ИЛИ выполняется по команде ____действий.

4. Операция сдвига выполняется командой ____ действий.

5. Перечислить пять основных операций передачи данных, которые можно найти в составе команд МП.

6. Операции принятия решений выполняются по командам ____.

7. Перечислить команды условного перехода состава команд МП.

8. Специальная группа команд, которая в ходе решения текущей задачи может часто повторяться, называется ____ (индексом, подпрограммой).

9. Какую текущую команду отрабатывает МП, ответвляясь в подпрограмму?

10. Какая команда в конце подпрограммы приводит МП в основную программу?

11. Перечислить пять различных команд, принадлежащих составу команд многих МП?

12. См. табл. 4.2. Что следует записать в верхней строке таблицы при операции вычитания?

Таблица 4.2.

Операция вычитания

Операция	?	?	Символика (обозначение регистра—его содержимое)
Вычесть	SUB M	80H	$A-M \rightarrow A$

13. См. рис. 4.6. Число 80H (H-код) представляет собой ____.

14. См. рис 4.6. После выполнения операции SUB *A* в аккумуляторе будет ____.

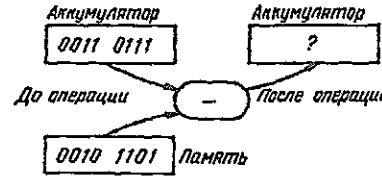


Рис. 4.6. Пример вычитания.

4.4. Структура элементарного микропроцессора

Основным устройством всех информационных систем является *центральный процессор* (ЦП). Из многочисленных ИС роль ЦП систем выполняют микропроцессоры. Обычно в технологии микроинформационной техники программную память, память данных, интерфейс ввода-вывода, дешифратор адресов выполняют на различных ИС, как это показано на рис. 4.1.

Центральным устройством системы является микропроцессор, который содержит обычно элементы размещения данных, называемые регистрами, и устройство счета, называемое арифметико-логическим устройством (АЛУ). Центральное устройство содержит также цепь декодирования команд и секцию управления и синхронизации. Оно снабжено также необходимыми соединениями с устройством ввода/вывода.

Основными функциями центрального устройства микроЭВМ являются следующие:

1) извлечение, декодирование и выполнение команд программы в указанном порядке;

2) передача данных из памяти и в память и из УВВ и в УВВ;

3) ответы на внешние прерывания;

4) установка общей синхронизации и сигналов управления для всей системы.

Большинство центральных устройств содержит по меньшей мере элементы, схематически представленные на рис. 4.7. Наиболее важные секции содержат различные регистры, АЛУ, дешифратор команд,

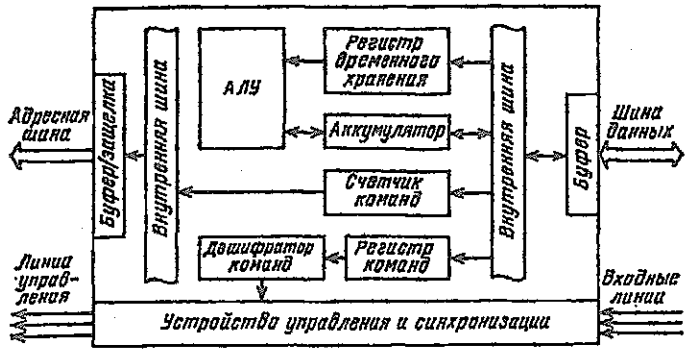


Рис. 4.7. Упрощенная архитектура центрального процессора.

устройства управления и синхронизации, а также УВВ. В настоящее время большинство микропроцессоров содержат множество дополнительных специальных регистров (на рис. 4.7 не показаны).

Арифметико-логическое устройство ЦП выполняет такие операции, как сложение, сдвиг/перестановка, сравнение, инкремент, декремент, отрицание, И, ИЛИ, ИЛИ ИСКЛЮЧАЮЩЕЕ, дополнение, сброс, инициализация.

Если АЛУ должно выполнить операцию сложения посредством команды ADD, процедура могла бы быть аналогичной представленной на рис. 4.8, о. Здесь содержимое аккумулятора OAH складывается с содержимым регистра временного хранения данных 05H. Сумма OFH помещена в аккумулятор.

На рис. 4.8,б приведены основные функциональные элементы типового АЛУ. Оно содержит сумматор и устройство сдвига, а результаты пересылаются в аккумулятор посредством внутренней шины данных.

Регистр состояния слова в АЛУ является устройством чрезвычайно важным (его называют иногда регистром кода условий или индикатором). Этот регистр состоит из группы триггеров, которые могут быть установлены или сброшены исходя из результатов последней операции, выполненной АЛУ. Эти триггеры или индикаторы содержат указатели нуля, отрицательного результата, переноса и т. д. Индикаторы используются для принятия решений, когда вводятся команды

ветвлений. Аккумулятор обычно используется в ходе большинства операций, выполняемых центральным устройством, например, передачи данных.

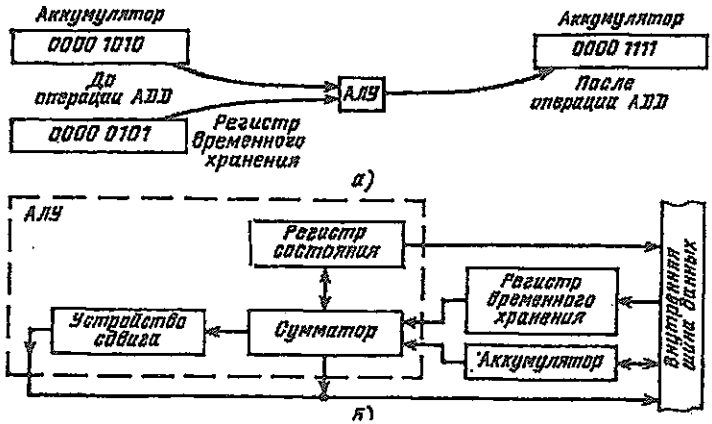


Рис. 4.8. Выполнение операции ADD *(а)* и структура АЛУ *(б)*.

Устройство управления и синхронизации (см. рис. 4.7) является наиболее сложным в центральном процессоре. Оно влияет на все события и управляет их протеканием внутри центрального устройства и во всей микро-ЭВМ. Мы упоминали в предыдущей главе, что каждая команда программы может быть разделена на этапы извлечения и выполнения. Каждый из них в свою очередь может быть разделен на элементарные микропрограммы. Микропрограммы каждой команды находятся в секции декодирования и выполняются блоком управления и синхронизации центрального устройства.

Шестнадцатиразрядный регистр, называемый *счетчиком команд,* представлен на рис. 4.7 как элемент, составляющий часть центрального устройства. Этот регистр служит для хранения адреса следующей команды, чтобы извлечь ее из памяти. Так как команды выполняются последовательно, счетчик команд считает прямым счетом, если только нет контрпорядка. Большая часть выпускаемых микропроцессоров имеет 16-разрядный счетчик команд, который может ад-

ресовать 64 К слов памяти посредством адресной шины. Нормальная последовательность выполнения команд программы может быть изменена специальными командами ветвления, вызова подпрограмм, возврата из подпрограмм или прерывания. Эти команды повлекут переход содержимого счетчика команд на другую величину, отличную от следующего старшего адреса. Чтобы вернуть программу в исходное состояние после последовательности ее запуска, оператор должен восстановить в счетчике команд номер первой команды программы.

Последовательность извлечение- декодирование- выполнение команд является основой функционирования вычислительной машины. Первая команда, извлеченная из памяти программы, определяет код операции первой команды и помещается в регистр команд устройством управления центральным процессором. Код операции истолковывается дешифратором команд, который указывает затем процессору процедуру управления и синхронизации, которой должна следовать программа для выполнения заданной команды.

Центральное устройство, показанное на рис. 4.7, является элементарным. Большая часть центральных устройств МП содержит, по меньшей мере, несколько дополнительных регистров (8 и 16 бит). Существуют очень большие различия в количестве и типе регистров в зависимости от типов МП.

Контрольные вопросы

1. Какую часть микро-ЭВМ обозначают сокращением ЦП?
2. См. рис. 4.1. Где располагается ЦП в блоке всей системы микро-ЭВМ?
3. Центральный процессор обычно содержит:

а) устройство размещения данных, называемое ____;

б) устройство счета, называемое ____;

в) устройство ____;

г) устройство и синхронизации.

4. Какие четыре основных назначения ЦП в микро-ЭВМ?
5. Какое устройство микро-ЭВМ сокращенно называется АЛУ?
6. См. рис. 4.9. Каково содержимое аккумулятора после операции И?

7. См. рис. 4.9. После операции И индикатор нуля будет _____ (установлен, сброшен).

8. Регистр состояний АЛУ называется также регистром кода _____, а триггеры называются ___

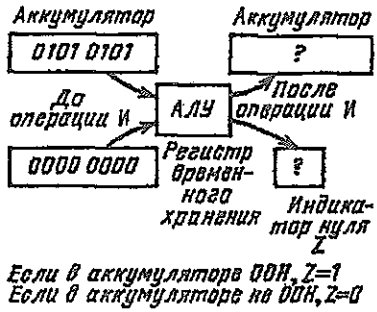


Рис. 4.9. К упражнениям 4.47 и 4.48.

9. Какая важная часть ЦП предназначена для управления всеми событиями внутри системы?

10. Регистр ЦП, удерживающий адрес последующей команды, извлекаемой из программной памяти, называется _____.

11. Обычно счетчик команд инкрементируется в прямом направлении для адресации команд в программной памяти в порядке возрастания адреса, за исключением случаев, когда его содержимое изменяется командами

12. В начале процедуры выполнения команды КОП первой команды помещается в регистр _____ (аккумулятора, команд) МП.

13. Часть МП, интерпретирующая КОП, помещенный в регистр команд, и определяющая последующую процедуру управления и синхронизации для выполнения команды, является _____.

4.5. Функционирование микро-ЭВМ

Пусть требуется выполнить простую операцию сложения трех чисел, например $10+5+18=33_{10}$. Короткая и простая микропрограмма выполнения этой операции могла бы быть записана в следующей последовательности.

Команда 1: загрузить (LOAD) первое число (10_{10}) в ЦП.

Команда 2: сложить (ADD) второе число (5_{10}) с первым.

Команда *3* сложить (ADD) третье число (18_{10}) с двумя предыдущими.

Команда *4:* поместить (STORE) сумму (33_{10}) в ячейку памяти 2000Н.

После загрузки в память программы эти команды могли бы извлекаться из нее как команды памяти, показанной на рис. 4.10. Заметим, что первая команда программы начинается с адреса 0000Н. Эта команда (LOAD число OAH) использует 2 байт памяти. Первый байт памяти содержит оперативную часть команды, другой—операнд. Код операции LOAD для микропроцессора, используемого в этом примере, будет 86Н ($1000\ 0110_2$). Операнд OAH ($0000\ 1010_2$) является первым числом, подлежащим загрузке в аккумулятор микропроцессора. Заметим, что рис. 4.10 является широко распространенным представлением содержимого памяти и адресов в шестнадцатеричной записи. В реальной действующей машине такая информация представляется в форме напряжения Н- и L-уровней.

Предположим, что программа размещена в блоке ОЗУ микро-ЭВМ (рис. 4.1), в которую входят устройства, представленные на рис. 4.7. В таком случае рис. 4.11 иллюстрирует каждую операцию программы (LOAD, ADD, ADD, STORE).

Операция загрузки (LOAD) первой команды подробно приведена на рис. 4.11, а и показывает, что содержимое ячейки памяти 0001Н загружено в аккумулятор, который содержит после этого $0000\ 1010_2$ — первое слагаемое число. В результате операции загрузки стирается предыдущее и записывается новое содержимое аккумулятора. Вторая команда, операция ADD, детализирована на рис. 4.11,б. Содержимое ячейки памяти 0003Н ($0000\ 0101_2$) складывается с содержимым аккумулятора $0000\ 1010_2$, что дает сумму $0000\ 1111_2$, помещаемую в аккумулятор, и мы можем заметить, что содержимое аккумулятора изменяется при операции ADD.

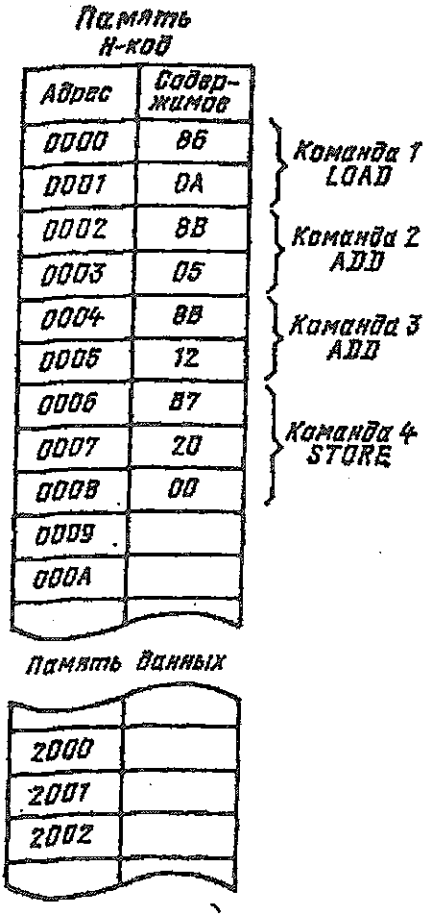


Рис. 4.10. Часть программы сложения 10+5+18.

На рис. 4.11, в показана вторая операция ADD (команда *3);* содержимое аккумулятора—сумма 00001111_2 сложена с содержимым ячейки памяти 0005H, т. е. выполняется операция 0000 1111 +0001 0010==0010 0001. Окончательно 0010 0001 появляется в аккумуляторе после второй операции ADD.

Операция STORE (РАЗМЕСТИТЬ) по команде *4* представлена на рис. 4.11,г. Содержимое аккумулятора (0010

0001_2) передано и размещено в ячейке памяти по адресу 2000Н. Заметим, так как это важно, что ячейки памяти данных были идентифицированы в памяти программы двумя раздельными байтами (0007Н и 0008Н). Ячейка памяти программы 0006Н содержит КОП В7Н прямой команды STORE (см. рис. 4.10).

Рассмотрим извлечение, декодирование и выполнение команды LOAD по адресам 0000Н и 0001Н в программе.

Этот тип команды будет выполнен, вероятно, за время около 2—6 мкс большинством микро-ЭВМ. Рисунок 4.12 иллюстрирует процедуру выполнения центральным процессором этой специальной операции.

Рассмотрим сверху слева направо последовательность действий на рис. 4.12: счетчик команд прежде всего устанавливает адрес первого этапа программы.

После этого 16-разрядный адрес передается в адресный регистр, затем на адресную шину и в память программы. Для активизации памяти программы ЦП выдает сигнал считывания программы (1 на линии *R/W*), *в* то время как дешифратор адресов (который не входит в состав ЦП) активизирует выбор кристалла CS нулем. Затем счетчик команд инкрементируется до 0001 Н, ячейка памяти программы 0000Н становится доступной и ее содержимое считывается на шину данных. Код операции (86Н) команды LOAD передается в регистр ЦП. Этап извлечения КОП команды LOAD завершен.

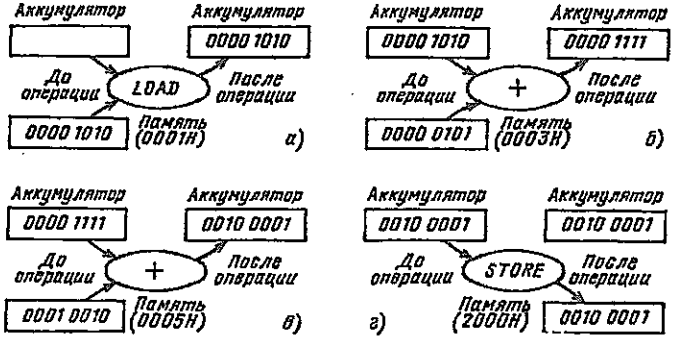


Рис. 4.11. Команды:
а—LOAD; б—ADD; в—ADD; г — STORE.

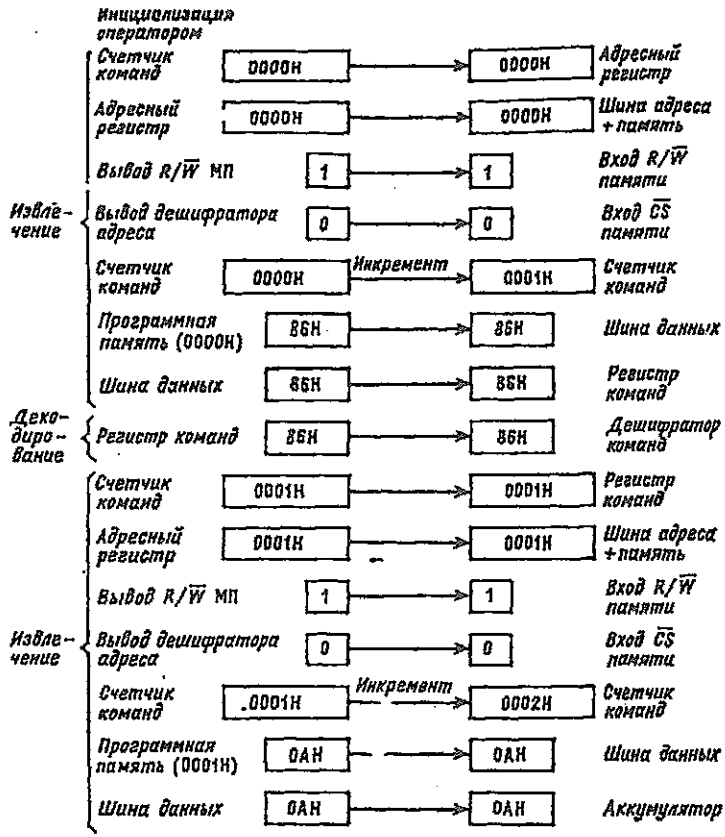


Рис. 4.12. Подробный анализ команды LOAD непосредственных данных.

Затем КОП (86Н), содержащийся в регистре команды ЦП, интерпретируется дешифратором команд. В данном случае ЦП определяет, идет ли речь о команде LOAD непосредственно, что означает загрузку им содержимого памяти, адрес которой следует непосредственно за КОП, в аккумулятор. Содержимое счетчика команд (0001 Н) передается в адресный регистр, на адресную шину и в память.

Центральный процессор выдает импульс HIGH считывания на вход *R/W* памяти. Импульс LOW поступает на вход

CS памяти, что активизирует память. В третьей строке (снизу) на рис. 4.12 счетчик команд инкрементируется до 0002Н, что подготавливает его к извлечению последующей команды. Ячейка памяти 0001 Н становится доступной, и ее содержимое (ОАН) поступает на шину данных, затем передается в аккумулятор. В ходе этапа извлечения КОП всегда помещается в регистр команд.

Контрольные вопросы

1. Две части команды микропроцессоров называются операцией и ____.

2. См. рис. 4.10. Когда ячейка памяти 0002Н доступна, ее содержимое передается в регистр ____ ЦП через шину ____ (адреса, данных).

3. См. рис. 4.10. Ячейка памяти 0003Н содержит 05Н, что рассматривается ЦП как____ (операция, операнд) команды.

4. Двоичное 16-разрядное число может быть представлено шестнадцатеричным числом _____ цифрами.

5. Привести три последовательных этапа цикла команды микропроцессора.

6. См. рис. 4.12. Этап извлечения является основным при операции ____ (считывания, записи), содержимое памяти помещается в ____ (аккумулятор, регистр команд) микропроцессора.

7. В ходе извлечения (дешифратор команд, счетчик команд) указывает адрес КОП извлечения из памяти.

8. См. рис. 4.12. Какие другие входы памяти необходимы для активизации считывания байта кроме адресного входа?

9. См. рис. 4.10. Какое шестнадцатеричное число должно быть помещено в ячейку памяти 2000Н после выполнения ЦП команды 4?

10. См. рис. 4.12. После последовательного выполнения последовательности действий счетчик команд восстанавливается ____(дешифратором адреса, оператором).

ГЛАВА 5. АППАРАТНЫЕ СРЕДСТВА МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ

5.1. Архитектура МП 8086

При создании микропроцессора 8086 ставилась задача расширения функциональных возможностей микропроцессора 8080 по следующим направлениям: выполнение всех арифметических и логических операций над 16-разрядными данными; введение операций над цепочками байтов и слов (строками); обеспечение прямой адресации к памяти емкостью до 1 Мбайт, т.е. формирование 20-разрядной адресной шины; обеспечение возможности написания и использования программ, динамически перемещенных в памяти; введение программных и аппаратных средств для создания многопроцессорных систем.

5.2. Структурная схема микропроцессора

Структурную схему микропроцессора можно условно разбить на два устройства рис. 5.1а:
- операционное устройство
- шинный интерфейс

Шинный интерфейс обеспечивает функции, связанные с выборкой команд, операндов из памяти, записью операндов в память, установкой очередности команд, а также формированием адресов операндов и команд.

Операционное устройство выполняет команды и возвращает результаты в память через интерфейс. Оба устройства работают параллельно и в большинстве случаев обеспечивают значительное совмещение выборки и выполнения команд, причем интерфейсное устройство обеспечивает извлечение кодов команд заблаговременно, в то время как операционное устройство выполняет текущую команду. Это повышает пропускную способность мультиплексированной шины адреса/данных ША/D и увеличивает быстродействие МП.

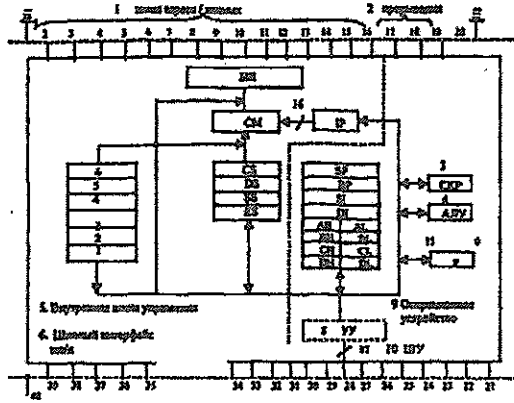


Рис. 5.1 а.

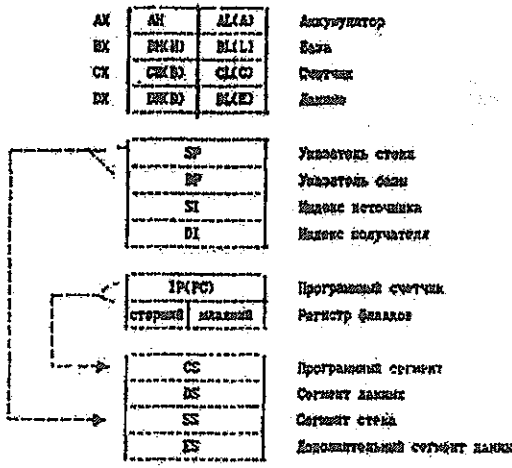


Рис. 5.1 б.

Операционное устройство рис.5.1 а содержит группу общих регистров (РОН), арифметико-логическое устройство (АЛУ), основу которого составляет комбинационный 16-битный сумматор с последовательно-параллельным переносом, регистр флажков (F), схему коррекции результатов (СКР), используемую при работе с данными, представленными в двоично-десятичных ко-

дах. Операционное устройство выполняет команды, обменивается данными и адресами с шинным интерфейсом, оперирует общими регистрами и флажками. В его составе имеется блок микропрограммного управления (УУ), который дешифрирует команды и формирует необходимые управляющие сигналы. Как видно из рис.5.1а в МП 8086 имеется 14 16-битных регистров (вместо семи 8-битных регистров в 8080, в скобках указаны соответствующие регистры МП 8080), которые по своему назначению можно разделить на три группы.

Регистры AX, BX, CX, DX рис.5.1б образуют группу регистров общего назначения (РОН) и обладают свойством раздельной адресации старших (H) и младших (L) байтов, т.е. каждый из них можно использовать как 16-битный регистр или два 8-битных регистра. Это обеспечивает простую обработку 8- и 16-битных данных (байт и слов). Остальные регистры МП можно использовать только как 16-битные. Эти регистры предназначены в основном для хранения данных, они находятся в полном распоряжении программиста и могут участвовать без ограничений в выполнении арифметических и логических операций. Однако имеется много команд, которые специализируют регистры данных на определенные функции табл.5.1.

Таблица 5.1.

Регистр	Назначение	Функция
AX	Аккумулятор	Умножение, деление и ввод-вывод слов
AL	Аккумулятор (младший)	Умножение, деление и ввод-вывод байт Преобразование, десятичная арифметика
AH	Аккумулятор (старший)	Умножение и деление байт
BX	База	Базовый регистр; преобразование
CX	Счетчик	Операции с цепочками; циклы
CL	Счетчик (младший)	Динамические сдвиги и рабации
DX	Данные	Умножение и деление слов; косвенный ввод-вывод

SP	Указатель стека	Стековые операции
BP	Указатель базы	Базовый регистр
SI	Индекс источника	Операции с цепочками; индексный регистр
DI	Индекс получателя	- « -

Программы получаются наиболее компактными, если в командах арифметических и логических операций и в командах пересылки данных использовать аккумулятор AX. Регистр AL в основном соответствует аккумулятору A 8080. Регистр BX почти эквивалентен основному указателю памяти МП 8080 - регистровой паре HL .

Регистры SP, BP, SI и DI образуют группу указательных и индексных регистров. Они предназначены для хранения 16-битных значений смещений, используемых для адресации в пределах текущего сегмента памяти и обеспечивают при этом косвенную адресацию и динамическое вычисление эффективного адреса памяти. При этом регистры-указатели SP и BP содержат смещения адреса в пределах текущего сегмента памяти, выделенного под стек, а индексные регистры SI. и DI содержат смещения адреса в пределах текущего сегмента памяти, выделенного под данные.

Регистры CS, DS, SS и ES, образующие группу сегментных регистров, играют важную роль во всех действиях ЦП, связанных с адресацией памяти. Хотя МП имеет 20-битную шину физического адреса памяти, он оперирует 16-битными логическими адресами, состоящими из базового адреса сегмента и внутрисегментного смещения. Внутреннее устройство преобразования адресов превращает два логических адреса в 20-битный физический адрес.

Пространство памяти I M байт разделено на логические сегменты емкостью 64 K байт. Выполняя программу, МП может одновременно обращаться к четырем сегментам. Их базовые (начальные) адреса содержатся в сегментных регистрах CS (программный), DS (данных), SS (стека) и ES (дополнительный). Содержимое любого из этих регистров определяет текущий начальный адрес сегмента памяти, выделенного пользователем под информацию, соответствующую названию регистра.

Содержимое регистра CS указывает на текущий сегмент кода (программы), откуда выбираются команды. Выборка очередной команды осуществляется относительно содержимого *CS с* использованием значения указателя команд IP. Регистр DS указывает на текущий сегмент данных, в котором содержатся переменные. Регистр SS адресует текущий сегмент стека, в этом сегменте реализуются все стековые операции, включая и те, которые связаны с операциями возврата. Наконец регистр ES определяет текущий дополнительный сегмент, который обычно используется для хранения данных.

Последнюю группу регистров МП образуют 16-битные указатель команд IP (аналог программного счетчика PC) и регистр флажков F. Шинный интерфейс осуществляет изменение содержимого PC таким образом, что он содержит смещение следующей команды от начала текущего сегмента кода, т.е. указывает на следующую по порядку команду. При обычной работе PC содержит смещение следующей команды, которую будет выбирать шинный интерфейс ид памяти программ. Оно совпадает со смещением той очередной команды, которую будет выполнять операционное устройство. Поэтому при запоминании содержимого PC в стеке, например при вызове подпрограммы, оно автоматически корректируется, чтобы адресовать следующую команду, которая будет выполняться. Непосредственный доступ к PC имеют команды передачи управления.

В регистре F рис.5.2 используются следующие девять разрядов: CF - перенос, PF - четность, AF - вспомогательный перенос, ZF - нулевой результат, SF - знак, TF - пошаговый режим, IF - размещение прерывания, DF -направление, OF - переполнение, X - состояние не определено.

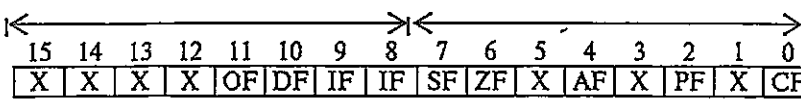


Рис.5.2. Структура регистров флажков.

Флаги AF, CF, PF, SF и ZF эквивалентны соответствующим признакам МП К580 и характеризуют признаки результата последней арифметической, логической или иной операции, влияющей на состояние этих признаков. Установка того или другого флага производится в следующих случаях:

CF- при переносе "I" из старшего бита или при заеме единицы в старший бит;

AF- при выполнении операции производится перенос "I" из младшей тетрадь; байта в старшую или осуществляется заем "I" из старшей тетрады;

ZF - если в результате выполнения операции получено нулевое значение;

SF - при получении "I" в старшем бите результата;

PF- если в представлении результата операции содержится четное число единиц.

К этой группе флагов относится также флаг OF, который устанавливается при наличии переполнения в результате выполнения арифметических операций над числами со знаком.

Три дополнительных флага предназначены для управления некоторыми действиями микропроцессора:

DF - управляет направлением обработка данных в операциях с цепочками байтов или слов. При DF=1 цепочка обрабатывается снизу вверх, т.е. происходит автоматическое уменьшение адреса текущего элемента цепочки. При DF=0 цепочка обрабатывается сверху вниз, т.е. происходит автоматическое увеличение адреса;

IF - предназначается для разрешения или запрещения (маскирования) внешних прерываний. При IF=0 внешние прерывания запрещены, т.е. процессор не реагирует на их запросы;

TF - применяется для задания процессору пошагового режима, при котором процессор после выполнения каждой команды останавливается и ждет внешнего запуска. Пошаговый режим задается установкой флага TF= I и обычно необходим при отладке программ.

Кроме выше перечисленных узлов шинный интерфейс имеет также сумматор адреса - СМ для вычисления адресов памяти.

Связь внутренних узлов МП с ША/D осуществляется через буфер шины БШ.

Контрольные вопросы

1.Из каких устройств состоит МП?

2.Какие регистры содержит операционное устройство?

3.Укажите группу регистров общего назначения?
4.Перечислите группу сегментных регистров?
5.Назначение регистра флажков?

5.3. Организация памяти

МП 8086 обеспечивает адресацию памяти емкостью I М байт. Адресное пространство памяти, показанное на рис. 5.3 представляет собой одномерный массив байт, каждый из которых имеет 20-битный физический адрес.

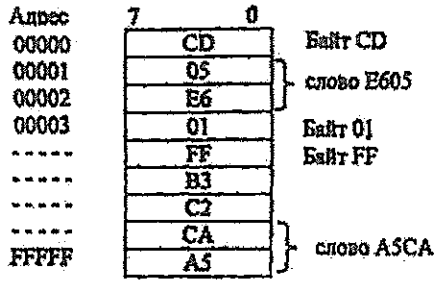


Рис.5.3.

При использовании МП 8086 память микро-ЭВМ физически организуется в виде последовательности слов рис. 5.4 в отличие от микро-ЭВМ на базе 8080, где память и физически и логически организуется в виде последовательности байтов.

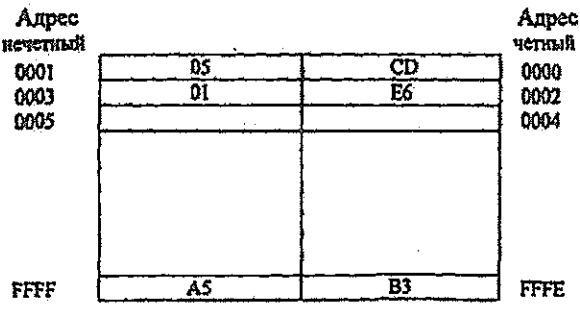


Рис. 5.4.

Любые два смежных байта в памяти образуют 16-битное слово. Младший байт слова имеет меньший адрес, а старший - больший.

Адресом слова считается адрес его младшего байта. Таким образом, 20-битный адрес памяти можно рассматривать и как адрес байта и как адрес слова. Например, в соответствии с рис. 5.3 адрес 00000 может обозначать и байт с этим адресом, что условно записывается в виде ([00000]) = CD, и слово с таким же адресом, что записывается в виде ([00000]) = 05CD.

Команды, байты и слова данных можно свободно размещать по любому адресу байта, что экономит память за счет плотной упаковки программ. Однако целесообразно размещать слова в памяти по четным адресам рис.5.4, так как МП может передавать такие слова за один цикл шины.

В МП 8086 адресное пространство памяти условно можно разбить на сегменты по 64 К байт. Каждому сегменту программой назначается базовый (начальный) адрес, являющийся адресом его первого байта в адресном пространстве памяти. Все сегменты начинаются на 16-байтных границах памяти. Сегменты могут быть соседними, не перекрывающимися, частично или полно перекрывающимися рис. 5.5

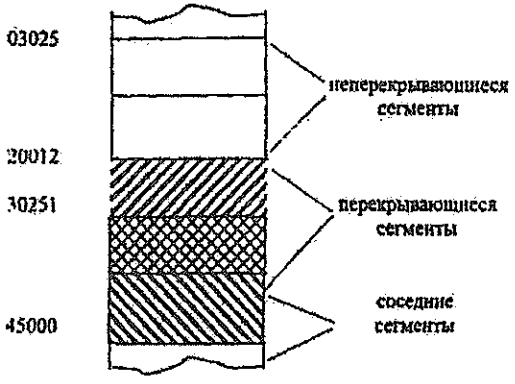


Рис.5.13.

Физическая ячейка памяти может принадлежать одному или нескольким сегментам.

Сегментное регистра CS, DS, SS и. ES содержат начальное адреса (базы) четырех текущих сегментов: сегмента кода (программы), сегмента данных, сегмента стека и дополни-

тельного сегмента данных соответственно. Для обращения к командам и данным, находящимся в других сегментах, необходимо изменять содержимое сегментных регистров.

Удобно считать, что каждая ячейка памяти имеет два адреса: физический и логический. Физический адрес представляет собой 20-битное значение в диапазоне от 00000 до FFFFF. Именно физический адрес выдается на шину адреса в начале каждого никла шины, связанного с обращением к памяти.

Программы оперируют не физическими, а логическими адресами. Логический адрес состоит из двух 16-битных без знаковых значений:

базового (начального) адреса сегмента, который называется также простой базой или сегментом, и внутрисегментного адреса или смещения. Для любой ячейки памяти база идентифицирует первый байт содержащего ее сегмента, а смещение определяет расстояние в байтах от начала сегмента до этой ячейки.

Когда устройство шинного интерфейса обращается к памяти для выборки команды или считывания/записи данных, оно образует из логического адреса **сегмент: смещение** физический адрес (например 1758:0100, где 1758 — логический адрес сегмента, 0100 логический адрес внутрисегментного смещения). Для этого в СМ база сегмента сдвигается влево на 4 бита и суммируется со смещением (рис. 5.6).

Устройство шинного интерфейса получает логический адрес ячейки памяти из различных источников в зависимости от типа выполняемого обращения к памяти.

Команды всегда выбираются из текущего сегмента кода - базовый адрес сегмента находится в регистре CS , а смещение в регистре IP. Стековые команды всегда обращаются к текущему сегменту стека: базовый адрес находится в регистре SS, а смещение - в регистре SP. Данные выбираются из регистра DS, но программист может заставить МП обращаться к данной, находящейся в другом сегменте.

Смещение данной вычисляет операционное устройство в соответствии с определенным в команде режимом адресации. Результат этого вычисления называется эффективным или исполнительным адресом EA операнда в памяти. При вычислении EA перенос из старшего бита игнорируется.

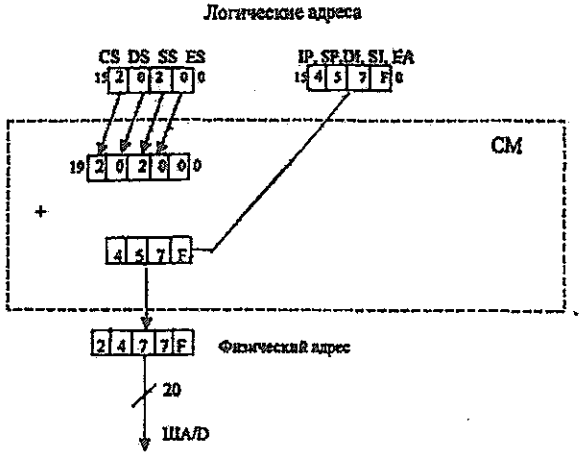


Рис. 5.6.

Сумматор адресов осуществляет, например, следующие вычисления: CS+IP - при выборке очередной команды, SS + SP- при обращении к стеку, DS + SI и ES +DI - при обработке строк, DS + EA - при обращении к запоминающему устройству с произвольной выборкой.

Контрольные вопросы

1. Какую адресацию памяти обеспечивает МП 8086?
2. На что условно можно разбить память МП 8086?
3. Что такое логический и физический адреса и как они определяются?
4. Что такое смещение?
5. Определите физический адрес:
6. 0250:0100, 3C20:2830, 4A40:0000, F520:3540, 4000:2000, AABC:2F3A.

5.4. Система команд

В отличие от МП 8080 в МП 8086 язык ассемблер более сложен, так как в МП 8086 применена сегментная организация памяти и организация одновременной адресации четырех сегментов.

В языке имеется более 100 базовых символических ко-

манд, в соответствии с которыми ассемблер генерирует более 3800 машинных команд. Кроме того, в распоряжении программиста имеется около 20 директив, предназначенных для распределения памяти, инициализации переменных, условного ассемблирования и т.д.

В зависимости от типа команд операнд может представлять собой байт или слово и храниться в РОН или памяти. В большинстве команд для адресации операндов используется специальный байт, который называется постбайтом и размещается вслед за первым байтом, содержащим код операции. Постбайт, структура которого представлена на рис.5.7, указывает способ адресации одного или двух операндов и состоит из трех полей. Поля mod и *r/m* в соответствии с табл.5.2.

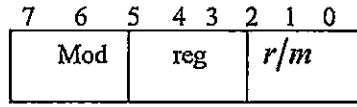


Рис.5.7.

Таблица 5.2.

Код регистра	Регистр	
	16-разрядный	8-разрядный
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

Таблица 5.3.

Код в поле	Значение исполнительного адреса EA
000	(BX+SI)+disp
001	(BX+DI)+disp
010	(BP+SI)+disp
011	(BP+DI)+disp
100	(SI)+disp
101	(DI)+disp
110	(BP)+disp
111	(BX)+disp

Если mod≠11, то в полях mod и *r/m* содержится информация, согласно которой устанавливается один из способов адресации – прямая, косвенная регистровая, по базе, индексная и по базе с индексированием. Если для реализации выбранного способа адресации требуется дополнительная адрес-

ная информация, то она указывается в одном или двух байтах в виде постбайтом. Наличие или отсутствие смещения и его размерность определяются полем mod следующим образом:

Если mod=00, то disp отсутствует;

Если mod=01, то disp размерностью 1 байт указывается за постбайтом, причем перед использованием этого смещения при формировании исполнительного адреса ЕА оно расширяется со знаком до 16 разрядов (расширение со знаком подразумевает заполнение старшего байта значением знакового разряда, указанного в disp);

Если mod=10, то за постбайтом следует 16-разрядное смещение disp, рассматриваемое как число со знаком.

Для каждой комбинации значений поля mod=00, 01 или 10 формирование исполнительного адреса ЕА определяется полем r/m в соответствии с табл.5.3.

Исключение из описанных способов кодирования полей mod и r/m составляет случай mod=00, r/m=110, соответствующий ЕА≠disp, причем disp – есть 16-разрядное смещение. Таким образом, при обращении к памяти имеется 24 варианта вычисления адреса ЕА, используемого в качестве смещения в сегменте при вычислении физического адреса.

Поле reg постбайта используется для адресации тогда, когда в команде задаются два операнда. В этом случае второй операнд всегда находится в регистре, код которого указывается в поле постбайта в соответствии с табл.5.3. В командах, где требуется только один операнд, поле reg постбайта используется совместно с байтом кода операции (КОП) для увеличения вариантов кодирования операций. Всего в 8086 используется восемь способов адресации, из которых пять реализуются с помощью постбайта. На рис.5.8 представлены форматы команд, иллюстрирующие задание различных способов адресации. (Некоторые форматы могут задавать разные способы адресации в зависимости от значения конкретных полей).

Регистровая адресация. Операнд находится в одном из РОН, код которого указывается в байте КОП (рис.5.8а) или в постбайте при mod=11 (рис.5.8 б, в, д). В командах с двумя операндами может быть использовано два регистра, причем код второго задается полем reg постбайта в соответствии с табл.5.4. В командах, оперирующих словами, байт КОП содержит ω=1 и код регистра определяет один из восьми 16-

разрядных регистров AX-DI. В командах, операндами которых являются байты (ω=0), код регистра определяет один из восьми 8-разрядных регистров AL-BH в соответствии с табл.5.3.

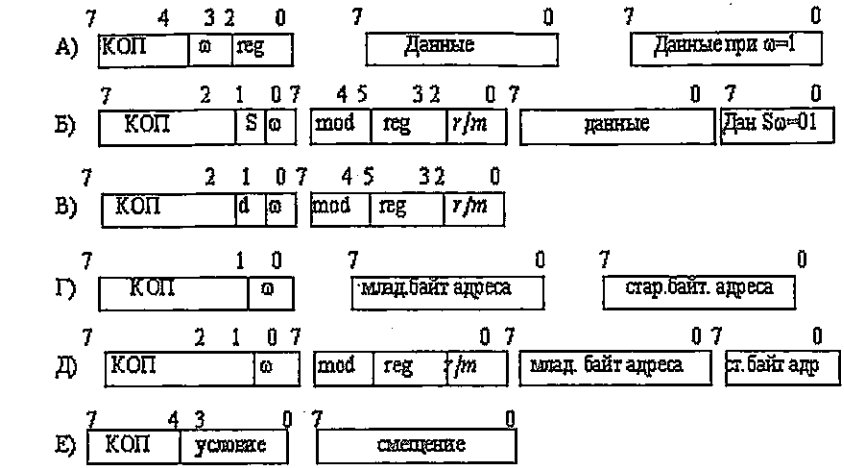


Рис. 5.8. Формат команд микропроцессора 8086.

Непосредственная адресация. Операнд содержится в формате команды в виде одно- или двухбайтовой константы (рис. 5.8., а, б). Этот способ задается особой комбинацией значений разрядов в поле КОП и может быть использован в большинстве команд с двумя операндами. Такие команды имеют постбайт, что делает возможным выполнение операций над константой и операндом расположенном в памяти или регистре. Способ непосредственной адресации неприменим в командах загрузки сегментных регистров и в команде занесения данных в стек. В этих случаях используется промежуточная загрузка константы в один из регистров, указанных в табл.5.3.

В командах с однобайтовыми операндами (ω=*0*) операнд имеет длину I байт и располагается в конце команды (рис. 5.8 а, б) в поле данных. В командах с операндами-словами (ω=*1*) операнд обычно занимает поле данных длиной *2* байт (рис.5.8 а). Однако некоторые команды с операндами-словами могут иметь однобайтовый операнд, что указывается

с помощью присвоения 1-разрядному полю S значения I (рис.5.8 б). Если S = I, то однобайтовое поле данных, содержащееся непосредственно в команде, перед использованием в операции расширяется со знаком до 16-разрядного слова. Расширение со знаком позволяет использовать 16-разрядные операнды в диапазоне значений от -128 до +127, которые в команде записываются однобайтовым операндом вместо двухбайтового. Двухбайтовый операнд используется в случаях, когда значение требуемого операнда (константы) выходит за пределы указанного диапазона.

Прямая адресация. Исполнительный адрес операнда задается с помощью двух байтов, которые следуют за первым байтом (рис.5.8 г) или за постбайтом (рис.5.8 д). Младший байт адреса идет по порядку первым. Прямая адресация позволяет обращаться к операндам в пределах одного сегмента памяти ёмкостью 64 К байт.

Возможна также длинная прямая адресация, при которой команда содержит 16-разрядный базовый адрес сегмента, а также 16-разрядное смещение в сегменте (исполнительный адрес), что позволяет при программировании осуществить обращение к операнду, находящемуся в произвольном (не текущем) сегменте, т.е. в пределах всей памяти емкостью в 1 М байт. Однако такой способ прямой адресации применим только в командах переходов и командах вызова подпрограмм, в которых он позволяет осуществлять межсегментные переходы. Невозможность применения этого способа адресации в командах, задающих операции над данными, представляет определенное неудобство.

Косвенная регистровая адресация. Исполнительный адрес операнда содержится в одном из регистров BX, SI или DI, код которого задается в постбайте (рис.5.8б, в, д).

Адресация по базе. Реализуется с использованием постбайта, в котором указывается один из регистров BX, BP или DI, содержащий 16-разрядный базовый адрес. Исполнительный адрес операнда формируется путем сложения этого базового адреса и 8- или 16-разрядного смещения указанного в команде непосредственно за постбайтом (рис.5.8 б); 8-разрядное смещение рассматривается как число со знаком, т.е. его значение лежит в диапазоне от -128 до +127. Использование адресации по базе позволяет осуществить доступ к элементам упорядоченных структур данных, когда смещение

конкретного элемента данных известно, а базовый адрес структуры вычисляется по программе.

Адресация с индексированием. В формате команды непосредственно за постбайтом задается 16-разрядный базовый адрес, а в постбайте указывается один из регистров BX, BP, SI или DI, в котором хранится индекс (рис.5.8 д). Адресация с индексированием логически эквивалентна адресации по базе с 16-разрядным смещением. Различие заключается в том, что 16-разрядное смещение интерпретируется как базовый адрес, а регистр хранит индекс. Однако поскольку любой из указанных четырех регистров может применяться как при адресации по базе, так и при адресации с индексированием, указанное различие является несущественным.

Адресация по базе с индексированием. Если в предыдущих способах адресации либо базы, либо смещения были фиксированы при выполнении команды и задавались в формате команды непосредственно, то адресация по базе с индексированием позволяет задавать оба этих параметра в регистрах (рис.5.8 б, в). Это дает возможность вычислить во время выполнения программы как базовый адрес структуры данных, так и смещение одного из ее элементов.

Поскольку любой из регистров BX и BP может использоваться в качестве базового, а любой из регистров SI и DI может служить индексным, имеется четыре различные комбинации регистров, которые могут реализовать адресацию по базе с индексированием. Кроме того, адресация по базе с индексированием может быть дополнена 8- или 16-разрядным смещением, которое является третьим слагаемым при вычислении исполнительного адреса EA; 8-разрядное смещение, как ранее, рассматривается как число со знаком, находящемся в диапазоне от -128 до +127.

Относительная адресация. Исполнительный адрес вычисляется как сумма содержимого указателя команд IP и 8- или 16-разрядного смещения со знаком, заданного непосредственно в команде (рис.5.8е). Следует подчеркнуть, что в момент вычисления исполнительного адреса значение указателя команд IP равно адресу первого байта следующей команды. Способ относительной адресации так же, как и прямая адресация, имеет ограниченное применение и используется только в командах переходов, вызова подпрограмм и управления циклами.

В завершение рассмотрения различных способов адресации отметим, что исполнительные адреса EA, получаемые описанными выше способами, являются в действительности 16-разрядными смещениями в сегменте. Как было показано на рис. 5. , 20-разрядный физический адрес получается путем сложения смещения в сегменте с предварительно сдвинутым на четыре разряда влево базовым адресом сегмента. Поскольку имеется четыре сегментных регистра CS, DS, SS и ES, для выполнения каждого преобразования логического адреса в физический должен быть выбран определенный регистр, содержащий базовый адрес соответствующего сегмента. Это осуществляют аппаратные средства микропроцессора, которые автоматически выбирают сегментный регистр согласно цели каждого обращения к памяти, как показано в табл. 5.4. В таблице указан также источник логического адреса, определяющий смещение в сегменте.

Таблица 5.4.

Вид обращения к памяти	Сегментный регистр	Логический адрес
Выборка команды	CS	IP
Операции со стеком	SS	SP
Обращение к исходной строке	DS	SI
Обращение к строке — результату	ES	DI
Использование BP в качестве базового регистра	SS	EA
Прочие обращения к переменной	D5	EA

В ряде случаев для эффективной передачи данных между сегментами удобно извлекать операнды, находящиеся в стековом сегменте, не прибегая к операциям со стеком, хто относится, например, к извлечению параметров подпрограмм, которые находятся глубоко в стеке. Для подобных целей существует специальный вид обращения к памяти с использованием базового регистра BP. Данные, для доступа к которым используется регистр BP, выбираются из стекового сегмента, причем содержимое регистра.

SS служит в этом случае базовым адресом, а смещение задается исполнительным адресом EA, в формировании которого участвует содержимое регистре BP.

При выполнении команд обработки данных может появиться необходимость адресовать операнды, находящиеся вне текущего сегмента, выбираемого процессором автоматически. Для этих целей, как уже отмечалось, применяется префикс замены сегмента, имеющий размерность I байт. Префикс помещается непосредственно перед байтом кода операции и содержит код сегментного регистре, используемого в качестве замены.

Систему команд 8086 можно разбить на шесть групп: команды передачи данных, арифметические команды, логические команды и сдвиги, команды анализа и преобразования строк, команды передачи управления и команды управления микропроцессором.

5.4.1. Команды передачи данных

Команды передачи данных удобно разделить на четыре подгруппы:
- общие команды передачи данных;
- команда передачи данных с привлечением стека (стековые команды);
- команды ввода-вывода;
- команды передачи цепочек байт или слов.

Команда MOV (переслать), общий вид команды
MOV *pol, ist*
где ist - источник
pol - получатель
Команда *MOV* имеет несколько форматов:
MOV mem/reg_1, mem/reg_2 – регистр ↔ регистр;
MOV mem/reg_1, data – регистр/ память ← данные;
MOV reg, data – регистр ← данные;
MOV ac, mem – аккумулятор ← память;
MOV mem, ac – память ← аккумулятор;
MOV Sreg, mem/reg – сегментный регистр ↔ память/регистр;
MOV mem/reg, Sreg – память/регистр сегментный регистр.

Самой гибкой и универсальной является команда MOV mem/reg_1, mem/reg_2. она содержит постбайт режима адреса-

ции, с помощью которого можно задавать любой допустимый режим адресации.

С помощью одной этой команды осуществляются передачи регистр-регистр/память и память-регистр, причем регистром может быть любой общий регистр. Длина команды составляет 2, 3 или 4 байта.

Пример 1. Передать слово из памяти в регистр

(SI) = 3456 MOV CX, [SI]
(DS) = ABCD (CX) = 3AAD
(AF126) = 3AAD

Пример 2. Передать слово из регистра в память

(BX) = 159D MOV [DI], BX
(SI) = 3456
(AF126) =159D
(DS) = ABCD
(AF126) = 3AAD

Пример 3. Передать байт регистр-память

MOV [DI], CL

Пример 4. Передать слово регистр-регистр

MOV BX, DX

Пример 5. Передать данные в память или регистр

(DI) = 1000 MOV word ptr [DI], 8400 H
(DS) = A345 (A4450) = 8400
MOV byte ptr2 [BX], 34 H
MOV [BP], [SI], 2CH
MOV word ptr CX, 1000H

Пример 6. Передать данные из памяти в аккумулятор и наоборот

MOV AL, [DI]
MOV [BP], AX

Пример 7. передать данные из сегментного регистра в память/регистр

MOV [BX], ES

Команда обмена XCHG имеет два формата:

XCHG reg, mem/reg;
XCHG ac, reg

Пример 8. Произвести обмен между памятью и регистром

(CX) = 37AA		
(BP) = 2346	XCHG CX, [BP]	(A2346) = 37AA
(SS) = A000		(CX) = DCBA
(A2346) = DCBA		

Команды LEA, LDS, LES служат для передачи в регистры не данных, а адреса.

Форматы команд

LEA reg, mem;
LDS reg, mem;
LES reg, mem.

Пример 9.

(BX) = 0400	LEA BX, [BX] [SI]	
(SI) = 0030		(BX) = 0430

Команды LDS и LES выполняют почти одни и те же действия: вычисляется эффективный адрес EA памяти, который суммируется с содержимым регистра DS; затем слово из памяти по полученному адресу загружается в адресуемой командой общий регистр, а следующее слово из памяти загружается в регистр DS (команда LDS) или ES (команда LES)

Пример 10.

(DS) = C000	LDS SI, [DI]	(SI) = 0180
(DI) = 0010		(DS) = 2000
([C0010]) = 0180		
([C0012]) = 2000		

Команды ввода-вывода IN и OUT

Форматы команд следующие:

IN ac, port;
IN ac, DX;
OUT port, ac;
OUT DX, ac.

Двухбайтные команды IN ac, port. OUT port, ac содержат во втором байте прямой адрес порта.

Однобайтные команды IN ac, DX, OUT DX, ac также допускают передачи байт и слов, но теперь 16-битный адрес порта находится в регистре DX и максимальный адрес порта равен FFFF. Косвенная адресация через регистр DX удобна в тех случаях, когда адрес нужного порта вычисляется по ходу выполнения программы.

Пример 11.
IN AX, 90H – ввода слова
IN AL, DX – ввода байта
OUT 92H, AL – вывод байта
OUT DX, AX – вывод слова

5.4.2. Команды арифметических операций

Команды сложения ADD, ADC, INC.

Форматы команд следующие:
ADD mem/reg_1, mem/reg_2 – сложить содержимое память/регистр-память/регистр;
ADD mem/reg_1, data – сложить содержимое память/регистр – данные;
ADD ac, data – сложить содержимое аккумулятор – данные;
ADC mem/reg_1, mem/reg_2 – сложить с переносом;
ADC mem/reg_1, mem/reg_2;
ADC ac, data;

INC mem/reg – инкрементировать память/регистр.

Пример 12.

а)	(CX) = 0049 (SI) = 09EA	ADD SI, CX	(SI) = 0A33
б)	(AX) = A0A0 (DI) = 3000 (DS) = 1000 (13000) = B456	ADD AX, [DI + 8]	AX = A546
в)	(CX) = 4567	ADD CX, OABCH	(CX) = F134
г)	(AL) = 4E	ADD AL, 0C5H	(AL) = 13

Пример 13.

а)	(AX) = 89AB (DI) = 0300 (DS) = 1200 (12300) = 3748 (CF) = 1 - флажок переноса	ADC [DI], AX	(12300) = C0F4
б)	(AL) = F4 (CF) = 0 - флажок переноса	ADC AL, 35H	(AL) = 29

Пример 14.

а)	(DI) = FF00	INC [DI]	(AFF00) = 0

(DS) = A000
(AFF00) = FF
(CF) = 0 - флажок переноса

б) (SI) = 00FF INC SI (SI) = 0100
(CF) = 1 - флажок переноса

5.4.3. Команды вычитания SUB, SBB, DEC, NEG

Форматы команд следующие:

SUB mem/reg_1, mem/reg_2 — вычитание содержимого память/регистр-память/регистр;
SUB mem/reg_1, data — память/регистр — данные;
SUB ac, data — аккумулятор — данные;
SBB mem/reg_1, mem/reg_2 — вычитание с заемом память/регистр — память/регистр;
SBB mem/reg_1, data — память/ регистр — данные;
SBB ac, data - аккумулятор- данные;
DEC mem/reg - декремент память/регистр;
NEG mem/reg - изменение знака (или образование дополнительного кода).

Пример 15.

а) (CL) = 83 SUB CL, [BP + 8] (CL) = E9
(SS) = 1000
(BP) = FFFF
(20006) = 9A

б) (SI) = A000 SUB [SI + 20H], 2AAH
(DS) = 2500 (2F020) = 31AC
(2F020) = 3456

в) (AL) = 73 SUB AL, 0B7H (AL) = BC

г) (BH) = 78 SBB BH, DH (BH) = 6F
(DH) = 08
(CF) = 1

д) (BP) = 0F6A SBB [BP+40H], 5555H
(SS) = 2F00 (2FFAA) = CAAB
(2FFAA) = 2000
(CF) = 0

е) (AX) = 6B3A SBB AX, 4D2CH (AX) = 1E0D
(CF) = 1

ж) (SI) = 6000 DEC [SI] (30000) = 4F
(DS) = 2A00
(30000) = 50

	(CF) = 1		
з)	(DX) = 0000	DEC DX	(DX) = FFFF
	(CF) = 0		
и)	(BX) = 0006	NEG BX	(BX) = FFFA

5.4.4. Команды умножения MUL и IMUL

Форматы команд следующие:

MUL reg – произвести умножение содержимого регистра с содержимым аккумулятора;

MUL mem - произвести умножение содержимого ячейки памяти с содержимым аккумулятора;

IMUL reg ;
IMUL mem } - умножение знаковых чисел

В операции над байтами функции аккумулятора выполняет регистр AL, а 16-битное произведение образуется в регистрах AH-AL. Регистр AH называется расширением аккумулятора AL. Если reg или mem идентифицирует слово, оно умножается на содержимое аккумулятора AX, а произведение длиной 32 бита формируется в регистрах DX-AX. В этой операции расширение аккумулятора AX является регистр DX.

Пример 16.

а)	(AX) = A950	MUL DL	(AX) = 1200
	(DL) = 20		
б)	(AX) = 0350	MUL [SI + 15H]	(AX) = BA00
	(DS) = 1000		(DX) = 0003
	(SI) = F000		
	(1F015) = 0120		

Команда IMUL осуществляется практически такие же действия, что и команда MUL, но сомножители и произведение интерпретируются как знаковые двоичные числа в дополнительном коде.

Пример 17.

(AX) = A590	IMUL Dl	(AX) = F200
(DL) = 20		

5.4.5. Команды деления DIV и IDIV

Форматы команд следующие:

DIV reg - произвести деление содержимого аккумулятора на содержимое регистра, результат занести в аккумулятор;

DIV mem - произвести деление содержимого аккумулятора на содержимое ячейки памяти, результат занести в аккумулятор;

IDIV reg
IDIV mem } - деление знаковый числе

При делении частное формируется в регистре AL (или AX), а остаток в регистре AH (или DX). Дробное частное округляется до целого путем отбрасывания дробной части результата.

Пример 18.

а) (AX) = 002D DIV CL (AX) = 0307
(CL) = 06

б) (AX) = FF57
IDIV CH (AX) = 000D
(CH) = F3

5.4.6. Команды логических операций и команды сдвигов

Логические операции реализуемые в 16 разрядном процессоре, представлены булевыми операторами AND (логическое умножение), OR (логическое сложение),XOR (исключающее ИЛИ, т.е. сложение по модулю 2), NOT (инверсия) и командой TEST, которая выполняет конъюнкцию операндов, но не изменяет их значений (неразрушающая проверка). Все логические операции являются поразрядными, т.е. выполняются независимо для всех бит операндов.

Форматы команд следующие:

AND, OR, XOR, TEST } $mem/reg_1, mem/reg_2$

AND, OR, XOR, TEST } *mem/reg,data*

AND
OR
XOR } *ac, data*
TEST

NOT mem/reg

Команда поразрядной конъюнкции AND в основном применяется для перевода в нулевое состояние тех бит операнда, которые определяются другим операндом — маской. Маска должна содержать нули в сбрасываемых битах и единицы в остальных.

Команда поразрядной дизъюнкции OR применяется для установки в 1 определенных бит операнда с помощью маски, а также упаковки байт или слов из полей других элементов данных.

С помощью команды исключающего ИЛИ XOR можно инвертировать определенные биты операнда, сравнивать операнды на абсолютное равенство и переводить регистр в нулевое состояние.

Пример 19.

а) (AX) = 3456
(SI) = F260 AND AX, [SI + 10H] (AX) = 2002
(DS) = EA40
(F9670) = AAAA

б) (DL) = 33
(DS) = 1000 OR TEMP, DL (TEMP) = BF
(TEMP) = BC

в) (AX) = 7777
XOR AX, DX (AX) = CB49
(DX) = BCDE

г) (CX) = 1234
(SI) = 6000
TEST [SI], CX
(DS) = CC00
(D2000) = 0010
(DI) = 3500

д) (BP) = 6000
AND [BP] [DI], 0F000H ([B1500]) = 3000
(SS) = A800
(B1500) = 39AF

е) (DX) = 332F OR DX, 0F0F0H (DX) = F8FF
ж) (BX) – FF20
(DS) = 2000 XOR [BX], 00FH ([3BF20]) = 3750
(3BF20) = 375F
з) (CX) = 5678 NOT CX (CX) = A987
и) (SI) = 8800
(DS) = CC00 NOT [SI + 100H] (D4900) = F0F0
(D4900) = 0F0F

5.4.7. Команды передачи управления

Команды передачи управления подразделяются на команды безусловных переходов, условных переходов, вызовов, возвратов, управления циклами и командами прерываний.

5.4.7.1. Команды безусловных переходов

Форматы команд следующие:
JMP disp L;
JMP disp;
JMP mem/reg;
JMP addr;
JMP mem.

При выполнении команд безусловных переходов происходит модификация IP или IP и CS, а их прежнее содержимое теряется.

Двухбайтная команда JMP disp L содержит во втором байте смещение, которое интерпретируется как знаковое целое. При выполнении команды значение смещения прибавляется (с расширением знака до 16 бит) к содержимому PC, которое соответствует адресу команды, находящейся после команды JMP. Диапазон значений байта смещения составляет – 128 ÷ + 127. Если смещение положительное, осуществляется переход вперед, а если отрицательное – переход назад.

Пример 20.
(IP) = 2A72 JMP14 (IP) = 2A86

Трехбайтная команда JMP disp производит такое же действие, как предыдущая команда, но содержит 16-битное смещение. Оно по прежнему интерпретируется как знаковое целое, поэтому область перехода увеличивается до – 32768 ÷ + 32767 байт относительно адреса команды, находящейся после

команды JMP disp.

Пример 21.

(IP) = 3E60 JMP 0002 (IP) = 4060

Команда JMP mem/reg реализует косвенный безусловный переход в программе. Здесь адресом перехода, загружаемым в PC, служит содержимое 16-битного регистра или слова памяти, определяемое постбайтом режима адресации.

Примеры 22.

(BX) = 30BA JMP BX (IP) = 30BA

(BX) = 3542 JMP [BX] (IP) = ABCD
(DS) = 421A
(456E2) = ABCD

Последние две команды JMP реализуют прямой и косвенный межсегментные переходы, т.е. допускают передачу управления любой ячейки в адресном пространстве памяти.

Пример 23.

(IP) = A23B JMP 7AF0 1234 (IP) = 7AF0
(CS) = 15A3 (CS) = 1234

(DS) = 3000
(DI) = 2F30 JMP [DI + 100H] (IP) = F34A
(32B30) = F34A (CS) = 015F
(32B30) = F34A
(32B32) = 015F

5.4.7.2. Команды условных переходов

В системе команд 16-разрядных микропроцессоров есть 19 двухбайтных команд уловных переходов, называемых также разветвлениями. Все они имеют единый формат, представленный на рис.5.9. При выполнении этих команд анализируется некоторое условие, закодированное текущими состояниями флажков (а в команде JCXZ —содержимым регистра CX), и в зависимости от удовлетворения условия переход осуществляется или нет. Данные команды позволяют проверить оба состояния всех флажков арифметических операций (кроме флажка AF), а также ряд комбинаций состояний нескольких флажков. Если условие истинно, управление передается по адресу перехода путем прибавления к содержимому

IP однобайтного знакового смещения (с расширением знака до 16 бит), а если условие ложно, выполняется следующая по порядку команда. Таким образом, все условные переходы в МП 8086 являются короткими. Время выполнения команд составляет четыре такта (переход не осуществляется) или восемь тактов (переход осуществляется) синхронизация.

Рис. 5.9. Формат команд условных переходов.

Пример 24.

(IP) = 3560

JNC AO (IP) = 3500

(CF) = 0

Пример 25.

(IP) = A000

JCXZ 80 (IP) = A000

(CX) = 0005

Команду JCXZ удобно помещать в начале цикла, особенно в том случае, если возможно ситуация, при которой цикл (со счетчиком CX) не выполняется ни разу.

При программировании может возникнуть необходимость передачи управления за пределы действия команд условного перехода. Пусть, например, при ZF=1 необходимо передать управление команде с меткой MORE, которая находится через 0400 байт от данной точки в программе. В этом случае приходится использовать две команды:

```
JNZ NEXT; флажок ZF=0
JMP MORE; флажок ZF=1
NEXT:
```

Ассемблер сформирует команду JMP с двухбайтным смещением.

Следует отметить, что большинство команд условных переходов имеет две (и даже три) мнемоники, подчеркивающие содержательный смысл проверяемого условия и введенные для удобства программирования.

Команды позволяют проверить все отношения между знаковыми и без знаковыми числами. Фигурирующие в определении команд термины «больше» и «меньше» относятся к

знаковым числам, представленным в дополнительном коде, а «выше» и «ниже» - к без знаковым. Например, число BE «меньше» и «выше» числа 37.

Достаточно подробная информация о всех командах условных переходов содержится в табл.5.5.

Таблица 5.5.

Команды условных переходов

Мнемоника	Условие	Отношение	КОП	Функция
				Перейти, если
JA/JNBE	CFVZF=0	>	77	Выше/не ниже или равно
JAE/JNB	CF=0	≥	73	Выше или равно/не ниже
JB/JNAE	CF=1	<	72	Ниже/не выше или равно
JBE/JNA	CFVZF=1	≤	76	Ниже или равно/не выше
JC	CF=1		72	Если перенос
JE/JZ	ZF=1	=	74	Равно/нуль
JG/JNLE	(*)	>	7F	Больше/ не меньше или равно
JGE/JNL	SF⊕OF=0	≥	7D	Больше или равно/ не меньше
JL/JNGE	SF⊕OF=1	<	7C	Меньше/не больше или равно
JLE/JNG	(**)	≤	7E	Меньше или равно/не больше
JNC	CF=0		73	Нет переноса
JNE/JNZ	ZF=0	≠	75	Не равно/ не нуль
JNO	OF=0		71	Нет переполнения
JNP/JPO	PF=0		7B	Нет паритета/паритет нечетный
JNS	SF=0		79	Нет знака
JO	OF=1		70	Есть переполнения
JP/JPE	PF=1		7A	Есть паритет/паритет четный
JS	SF=1		78	Есть знак
JCXZ	(CX)=0		E3	Содержимое регистра CX=0

(*) (SF⊕OF) VZF = 0 (**) (SF⊕OF) VZF = 1

Контрольные вопросы

1.Составить программу сложения чисел (числа выбираются самостоятельно).

2.Составить программу сложения,вычитания длинных чисел.

3.Составить программу умножения и деления чисел.

4.Составить программу управления и контроля объектами систем автоматики, телемеханики и связи.

ГЛАВА 6. DEBUG И АРИФМЕТИКА

Итак, начнем наше обучение на ассемблере с представления того, как компьютер считает.

В повседневной деятельности мы привыкли к счету с применением десяти цифр 1,2.3 и т. д. Вследствие чисто технических особенностей компьютер применяет другой метод счета, в котором задействованы только две цифры 0 и 1. Например, до 5 он считает следующим образом: 1, 10, 11, 100,101. Числа 10,11,100 и т.д. являются двоичными, то есть базирующимися на системе счисления, состоящей всего из двух цифр - единицы и нуля, в отличие от десяти цифр, соответствует более привычной для нас десятичной системе. Таким образом, двоичное число 10 соответствует десятичному 2.

Однако двоичные числа обладают существенным недостатком — выглядят они длинно и громоздко. Шестнадцатеричные числа — гораздо более компактный способ записи двоичных чисел. В этой главе вы познакомитесь с двумя способами записи чисел: шестнадцатеричным и двоичными. Это поможет понять процесс счета в компьютере и способ хранения чисел—в битах, байтах и словах. Если вы уже имеете представление о двоичных и шестнадцатеричных числах, битах, байтах и словах, то вам будет намного легче ознакомиться с работой в отладчике программ.

6.1. Шестнадцатеричные числа

Так как с шестнадцатеричными числами легче обращаться, чем с двоичными числами (по крайней мере, из-за длины), то мы начнем со знакомства с шестнадцатеричными числами, и будем использовать для этого **DEBUG.EXE**, специальную программу, которую вы найдете на дополнительном диске **DOS** или **WINDOWS.**

Мы будем использовать **Debug** в этой главе для ввода и пошагового выполнения программ, написанных на машинном языке. Как и **Бейсик, Debug** обеспечивает удобную диалоговую среду. Но в отличие от Бейсика, он не распознает

десятичных чисел. Для **Debug** число 10 является шестнадцатеричным, а не "десяткой". И так как **Debug** говорит только на шестнадцатеричном языке, вам понадобится узнать кое-что о шестнадцатеричных числах. Но сначала разберемся с программой Debug.

6.2. Debug

Почему программа называется **Debug**? "Bugs" (дословно "насекомые") в переводе со слэнга программистов означает "ошибки в программе". В работающей программе этих ошибок нет, в то время как неработающая ("limping") программа имеет, по крайней мере, одну ошибку ("bug"). Используя Debug для пошагового запуска программы и наблюдая, как программа работает на каждом этапе, мы можем найти ошибки и исправить их. Этот процесс называется отладка ("debugging"), отсюда и произошло название программы **Debug**.

В соответствии с компьютерным фольклором, термин "debugging" (дословно "обезжучивание", "обезнасекомливание") имеет глубокие корни - он появился в тот день, когда перестал работать компьютер Гарвардского университета Марк 1. После долгих поисков техники обнаружили источник своих бед - небольшую моль, попавшую между контактами реле. Они удалили, моль и внесли запись в сменный журнал о процессе под названием "debugging", произведенном над Марком 1.

Найдите **Debug** на вашем дополнительном диске **DOS** или **WINDOWS** в подкаталоге **COMMAND** или **SYSTEM32**, и мы начнем. Пожалуй, вам даже стоит скопировать **DEBUG.EXE** на ваш рабочий диск, так как мы будем его постоянно использовать.

Примечание: начиная с этого места, в тексте будут приводиться распечатки команд пользователя и ответов компьютера. Напечатайте текст примера, нажмите клавишу "Enter", и вы увидите ответ компьютера, соответствующий тому, который приводится в распечатке. Однако, ответы компьютера могут не всегда совпадать с приведенными примерам из-за возможных различий между вашим компьютером и компьютером, с помощью которого писалась эта глава (мы обсудим эти различия позже). Отметим, что во всех примерах используются заглавные буквы. Это сделано только для того, чтобы избежать путаницы между буквой "I" и цифрой "1". Если вы

предпочитаете, то можете набирать примеры строчными буквами.

Теперь, после упомянутых выше—соглашений, запустим **Debug**, набрав его название после приглашения **DOS** (которое в этом примере выглядит как "C>"):

```
C> DEBUG.EXE
- - ответ
```

или через **WINDOWS\SYSTEM2\DEBUG.EXE**

```
- ответ
```

Дефис, который вы. видите в качестве ответа на вашу команду - это приглашение программы **Debug**, в то время как "C>" - это приглашение **DOS**. Это означает, что **Debug** ждет вашей команды.

Чтобы покинуть **Debug** и вернуться в **DOS**, напечатайте **"Q"** (англ. "Quit") около дефиса и нажмите "Ввод". Если хотите, попробуйте выйти и затем обратно вернуться в Debug:

```
-Q
C>DEBUG
```

6.3. Арифметика микропроцессора

Зная кое-что о шестнадцатеричной арифметике программы **Debug** (отладчик) и двоичной арифметике микропроцессора (МП), мы можем начать изучение того, как МП выполняет свои математические операций. Он использует внешние команды, называемые инструкциями.

Debug, наш гид и интерпретатор, много знает о микропроцессоре, расположенном внутри IBM PC. Мы будем использовать его, чтобы исследовать внутренние процессы, происходящие в МП; и начнем с запроса к Debug, чтобы тот высветил все, что он может сообщить о маленьких кусочках памяти, называемых регистрами, в которых могут храниться числа. Регистры похожи на переменные в Бейсике, но они не совсем то же самое. В отличие от Бейсика микропроцессор 8086 содержит ограниченное число регистров, и эти регистры не являются частью памяти вашего IBM PC.

Мы просим Debug показать содержимое регистров микропроцессора с помощью команды "R" (сокращение от англ. "Register"):

```
-R
AX=0000  BX=0000  CX=00000  DX=0000
SP=FFEE BP=0000 SI=0000  DI=0000
```

DS =3756 ES=3756 SS=3756 CS=3756 IP =0100 NV UP DI PL NZ NA PO NC

3756:0100 E485 IN AL, 85

(Возможно, на своем дисплее вы увидите другие числа во второй и третьей строках; эти числа показывают количество памяти компьютера. Эти различия будут сохраняться и дальше, позже мы их обсудим.)

Сейчас **Debug** выдал нам достаточно много информации. Обратим внимание на первые четыре регистра, **AX, BX, CX** и **DX**, о значениях которых **Debug** сообщил, что они все равны **0000**. Это регистры общего назначения; Остальные регистры **SP, BP, SI, DI, DS, ES, SS, CS** и **IP** являются регистрами специального назначения.

Четырехзначное число, показанное сразу вслед за именем регистра, является шестнадцатеричным. Раньше мы узнали, что одно слово описывается ровно четырьмя шестнадцатеричными цифрами. Так что, как вы видите. Каждый из 13 регистров микропроцессора является словом и имеет длину 16 бит. Поэтому компьютеры, созданные на основе микропроцессора, называются шестнадцатиразрядными.

Мы уже отмечали, что регистры похожи на переменные Бейсика. Это означает, что мы можем изменять их состояние, и. мы будем это делать. Команда **Debug "R"** не только высвечивает регистры. Если указать, в команде имя регистра, то **Debug** поймет, что мы хотим взглянуть на содержимое именно этого регистра и может быть изменить его. Например, мы можем изменить регистр **AX** так, как это показано ниже:

```
-R AX
AX=0000
:3A7
```

Давайте опять просмотрим содержимое регистров, чтобы убедиться в том, что в регистре **AX** теперь содержится 3A7h:

```
- R
AX=03A7 BX=0000 CX=00000 DX=0000 SP=FFEE BP= 0000
SI=0000 DI=0000
DS=3756 ES=3756 SS=3756 CS= 3756 IP=0100 NV UP DI
PL NZ NA
PO NC
3756:0100 E485, INAL, 8S
```

Так и есть. Итак, мы можем помещать шестнадцатерич-

ное число в регистр с помощью команды "R", указывая имя регистра и вводя его новое значение после двоеточия, как мы только, что сделали. В дальнейшем мы будем использовать эту команду для ввода чисел в регистры микропроцессора.

Теперь используем Debug как интерпретатор, чтобы работать непосредственно с микропроцессором. Зададим МП инструкцию на сложение чисел из двух регистров: сначала поместим число в регистр BX, затем дадим инструкцию МП прибавить число, хранящееся в BX, - к числу, хранящемуся в AX, и поместить ответ обратно в AX. Для начала нам нужно ввести число в регистр BX. Давайте сложим 3A7h и 92Ah. Используйте команду "R", чтобы поместить 92Ah в BX.

6.4. Память и микропроцессор

Итак, регистры AX и BX должны, наверное, содержать 3A7h и 92Ah, в чем мы можем убедиться с помощью команды "R":

```
AX = 03A7 BX = 092A  CX = 00000  DX = 0000 SP =
FFEE BP = 0000
  SI =0000  DI= 0000
DS = 3756  ES = 3756  SS = 3756  CS = 3756    IP =0100
NV UP  DI PL  NZ NA
PO NC
3756:0100 E485 IN AL, 85
```

Теперь, в регистрах AX и BX имеются нужные нам числа, каким образом мы сообщим микропроцессору о том, что их, то есть содержимое регистров BX и AX, нужно сложить? Для этого мы введем некоторые числа в память компьютера.

IBM PC имеет по меньшей мере 128 Мбайт памяти - значительно больше, чем нам здесь потребуется. Мы поместим два байта машинного кода в одном из уголков этого огромного количества памяти. В этом случае машинный код будет представлен двумя двоичными числами, с помощью которых мы сообщим микропроцессору о том, что надо прибавить регистр BX к AX. Затем для того чтобы мы могли увидеть результат, мы выполним эту инструкцию с помощью программы Debug.

В какой именно части памяти мы должны поместить эту двухбайтную инструкцию, и как мы сообщим микропроцессору о том, где ее искать. При включении микропроцессор разбивает память на части по 64 Кбайт каждая, называемые

сегментами. Чаще всего мы будем иметь дело с одним из этих сегментов, не зная реально, где именно в памяти он начинается. Мы можем поступать таким образом из-за способа, которым микропроцессор размечает память.

Все байты памяти помечаются числами, начиная с 0h и выше. Но помните четырехзначное ограничение шестнадцатеричных чисел? Это означает, что наибольшее число, которое может использовать микропроцессор - это шестнадцатеричный эквивалент числа 65535. Это, в свою очередь, показывает, что максимальный объем памяти, размечаемый МП, составляет 64К байт. Однако, как мы знаем из практики, микропроцессор может адресовать больше 64 Кбайт памяти. Как он это делает? - С помощь небольшого трюка: он использует два числа, одно для номера 64 Кбайт сегмента, второе для каждого байта, или смещения, внутри сегмента. При такой адресации МП может использовать до одного миллиона байт памяти.

Все адреса (метки), которые мы будем использовать в дальнейших примерах - это смещения от начала сегмента. Мы будем записывать адреса как номер сегмента и вслед за ним, через двоеточие, смещение внутри сегмента. Например, 3756:0100 будет означать, что мы находимся на смещении 100h внутри сегмента 3756h.

Каждым адрес соответствует одному байту в сегменте, и адреса расположены в возрастающем порядке, так что 101 h - байт следующий в памяти за 100h.

В записи наша двухбайтовая инструкция о сложении BX и AX будет выглядеть так: **"ADD AX, BX";**

Мы поместим эту инструкцию по адресу 100h и 101h в любом сегменте, который Debug начнет использовать. В соответствии с соглашением об адресации, оговоренном выше, мы сбудем говорить, что инструкция размещена по адресу 100h, так как это место размещения первого байта инструкции.

Команда Debug для исследования и изменения памяти называется «E» (от англ. "Enter"). Используйте эту команду для ввода двух байт инструкции ADD, как показано ниже.

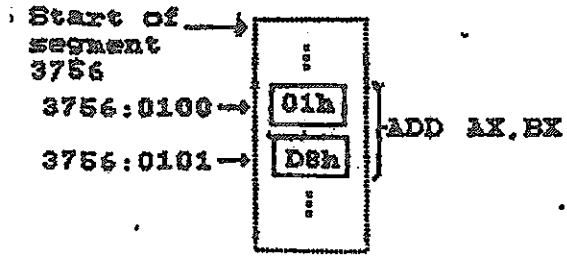


Рис. 6.1. Наша инструкция, начинающаяся с байта 100h от начала сегмента.

```
- E 100
3756:0100 E4.01
-E 101
3756:0101 85.D8
```

Числа 01h и D8h расположены по адресам 3756: 0100 и 3756:0101. Номер сегмента, который вы увидите, возможно, будет другим. Но это различие не будет влиять на нашу программу. Как и в приведенном примере, Debug высветит два различных двузначных числа в ответ на команды. Эти числа (E4h и 85h в нашем примере) – старые значения ячеек памяти на смещении 100h и 101h от начала сегмента, выбранного программой Debug, то есть эти числа – данные, оставшиеся в памяти от предыдущей программы после запуска Debug. (Если вы только что включили свой компьютер, то числа должны быть 00).

6.5. Сложение в машинных кодах

Теперь регистры должны выглядеть так:

```
AX=03A7  BX=092A  CX=00000  DX=0000  SP=FFEE
BP=0000           SI=0000  DI=0000
DS=3756  ES=3756  SS=3756  CS=3756  IP=0100  NV
UP 01  PL NZ
NA PO NC
3756:0100 01D8 ADD AX, BX
```

Инструкция ADD помещена в память именно там, где мы хотели ее разместить. Это видно из третьей строки сооб-

щения. Первые два числа, 3756:0100, дают нам адрес (l00h) первого числа нашей инструкции. За ними мы видим два байта, означающие ADD:01D8. Байт, равный 01 h, расположен по адресу l00h, а D8h -по адресу 101h. В конце строки располагается сообщение на машинном языке. Эту запись микропроцессор будет интерпретировать как инструкцию о сложении. После того, как инструкция размещена в памяти, необходимо сообщить микропроцессору о том, где она расположена.

Микропроцессор находит номер сегмента и адрес смещения в двух специальных регистрах, CS и IP, которые вы можете видеть распечатанными на предыдущем листинге. Номер сегмента хранится в CS или, сегменте кода (англ. "Code Segment"). Если вы посмотрите на распечатку регистров, то увидите, что Debug уже установил для нас CS (в нашем примере CS = 3756). Таким образом, полный адрес начала нашей инструкции 3756:0100.

Следующая часть этого адреса (смещение внутри сегмента 3756) хранится в регистре IP - указателе инструкции (англ. "Instruction Pointer"). Микропроцессор использует смещение, взятое из регистра IP, чтобы найти нашу первую инструкцию. Мы можем сообщить микропроцессору, где ее искать, записав в регистр IP адрес нашей первой инструкции IP =0100.

Но регистр IP уже установлен в 100h. Мы немного словчили: Debug устанавливает IP в .l00h всякий раз, когда его запускают. Зная это, мы специально выбрали l00h адресом первой инструкции и таким образом освободились от необходимости выполнять установку регистра IP отдельно. Этот прием стоит запомнить.

После ввода инструкции и правильной установки регистров мы попросим Debug ее выполнить. Для этого мы применим команду Debug -"T" (от англ. "Trace"), которая выполняет одну инструкцию за шаг и затем показывает содержимое регистров. После каждого запуска IP будет указывать на следующую инструкцию, в нашем случае, будет указывать на 102h. Мы не помещали никакой инструкции в 102h, поэтому в последней строке распечатки мы увидим инструкцию, оставшуюся от предыдущей программы.

Давайте с помощью команды "T" попросим Debug выполнить инструкцию;

```
- T
AX=0CD1 BX=092A CX=00000   DX=0000   SP=FFEE   BP=0000
SI=0000 DI=0000
DS=3756 E8=3756 SS=3756 05=3758   IP=0102   NV   UP   DI   PL
NZ   NA PO NC
3756:0102 AC LODSB
```

Вот и все. Регистр AX теперь содержит число CDlh, которое является суммой 3A7h и 92Ah. А регистр IP указывает на адрес 102h, так что в последней строке распечатки регистров мы видим инструкцию, расположенную в памяти по адресу 102h, а не по адресу 100h.

Как отмечалось ранее, указатель инструкции вместе с регистром CS всегда указывают на следующую инструкцию, которую нужно выполнить микропроцессору. Если мы опять напечатаем "T", то выполнится следующая инструкция. Но не делайте этого сейчас - ваш микропроцессор может зависнуть.

А что, если мы захотим выполнить введенную инструкцию еще раз, то есть сложить 92Ah и CDIh сохранить новый ответ в AX? Что нам надо сделать для того, чтобы объяснить микропроцессору где найти следующую инструкцию, и чтобы этой следующей инструкцией оказалась та же "ADD AX, BX", расположенная по адресу 100h. Можем ли мы изменит значение регистра IP на 100? Давайте попробуем. Используйте команду "R", чтобы установить IP в 100, посмотрите распечатку регистров:

```
AX=0CD1 BX=092A CX=00000 DX=0000   SP=PFEE   BP=0000
SI=0000   DI=0000
DS=3768   ES=3756 SS=3756 CS=3756 IP=0100 NV UP DI PL NZ
NA PO NO
3756:0100      ADD AX, BX
```

Попробуйте еще раз ввести команду "T" и посмотрите, содержит ли регистр AX число 15FBh. Действительно содержит.

Как видите, перед тем, как использовать команду "T", вам необходимо проверить регистр IP на соответствующую его значению инструкцию, располагаемую в нижней части листинга, показываемого командой "R". Таким образом, вы будете уверены, что микропроцессор выполняет нужную инструкцию.

А сейчас, установите регистр IP в 100h, убедитесь, что в

регистрах содержится AX = I5FBh, BX = 092Ah и снова попробуйте произвести вычитание.

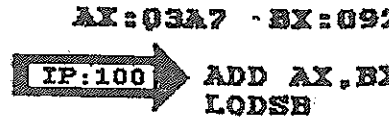


Рис 6.2. Перед выполнением инструкции сложения.

Рис. 6.3. После выполнения инструкции сложения.

6.6. Вычитание в машинных кодах

Мы собираемся написать инструкцию для вычитания BX из AX, так что после двух вычитаний в регистре AX появится результат 3A7h. Тогда мы вернемся к той точке, с которой начали. Кроме того, вы увидите, каким образом можно немного сэкономить усилия при вводе двух байтов в память.

Когда мы вводили два байта инструкции ADD, то печатали команду "E" дважды: один раз с 0100h для первого адреса и второй раз с 0101h для второго адреса. Однако мы можем ввести второй байт и без использования еще одной команды "E", если мы отделим второй байт от введенного пробелом. После окончания ввода нажмите клавишу "Enter". "Попробуйте этот метод на нашей инструкции вычитания:

```
-E 100
3756:0100   01 .29   D8 .08
```

Листинг регистров (помните о необходимости установки регистра IP в 100h) должен теперь 'показать инструкцию "SUB AX, BX", которая вычитает содержимое регистра BX из регистра AX и размещает результат в AX. Порядок записи AX и BX может быть разным, но инструкция, как и выражение AX = AX -BX, написанное на Бейсике, предполагает, что микропроцессор 8088, в отличие от Бейсика, всегда помещает ответ в первую переменную (в первый регистр).

Выполните эту инструкцию с помощью команды "T". AX должен содержать CD1. Измените IP так, чтобы он указывал

на эту инструкцию, и выполните ее опять (не забывайте сначала проверять инструкцию внизу листинга регистров). AX теперь должен содержать O3A7H.

Отрицательные числа в микропроцессоре

В главе 3 мы узнали, как микропроцессор использует форму двоичного дополнения для отрицательных чисел. Сейчас мы поработаем непосредственно с инструкцией SUB, чтобы проводить вычисления с отрицательными числами. Давайте дадим микропроцессору небольшой тест, чтобы посмотреть, получим ли мы FFFFh в качестве -1. Мы вычтем единицу из нуля, и, если мы были правы, то в результате вычитания в регистре AX должно оказаться FFFFh (-1). Установите значение AX равным нулю и BX равным единице, затем запустите инструкцию по адресу 100h. Мы получили то, что ожидали: AX = FFFFh.

6.7. Байты в микропроцессоре

До этого вся наша арифметика совершалась над словами, то есть четырьмя шестнадцатеричными цифрами. Знает ли микропроцессор, как выполнять математические операции над байтами? Да, знает.

Так как одно слово состоит из двух байт, каждый регистр общего назначения может быть разделен на два байта, известных как старший байт (первые две шестнадцатеричные цифры) и младший байт (следующие две шестнадцатеричные цифры). Название каждого из полученных регистров складывается из первой буквы названия регистра (от "A" до "O"), стоящей перед "X" в слове, и буквы "H" для старшего байта или буквы "L" для младшего. Например, DL и DH - регистры длиной в байт, а DX длиной в слово. (Применяемая здесь терминология не всегда удобна, так как в словах, хранящихся в памяти, младший байт идет первым, а старший вторым.)

Рис.6.4. Разбиение регистра AX на два байтовых регистра (AH и AL).

Проверим байтовую арифметику на инструкции ADD. Введите два байта 00h и C4h, начиная с адреса 0100h. Внизу листинга регистров вы увидите инструкцию "ADD AH, AL", которая суммирует два байта регистра AX и поместит результат в старшим байт AH.

Затем загрузите в регистр AX число 0102h. Таким образом вы поместите 01h в регистр AH и 02h в регистр AL. Установите регистр IP в 100h, выполните команду "T", и вы увидите, что регистр AX теперь содержит 0302. Результат сложения 01h + 02h будет 03h, и именно это значение находится в регистре AH.

Но предположим, что вы не собирались складывать 01h и 02h. Допустим, на самом деле вы хотели сложить 01h и 03h. Если регистр AX уже содержит 0102, можем ли мы изменить значение регистра AL на 03h. Нет. Вам придется изменять значение регистра AX на 0103h. Почему? Потому что Debug позволяет нам изменять только шестнадцатиразрядные регистры. Невозможно изменить только младшую или только старшую часть регистра с помощью Debug. Но, как вы видели в предыдущей главе, это не проблема. Имея дело с шестнадцатеричными числами, мы можем разделить слово на два байта, разбив четырехзначное число пополам. Так что слово 0103h становится двумя байтами 01h и 03h.

Чтобы испытать в действии эту инструкцию сложения, загрузите в регистр AX число 0103h. Инструкция "ADD AH,AL" по-прежнему находится в памяти по адресу 0100h, так что установите регистр IP в 100h и, имея в регистрах AH и AL значения 0lh к 03h, запустите выполнение инструкции. После этого AX будет содержать 0403h: 04h, а сумма 01h+ 03h, находится в регистре AH.

6.8. Умножение и деление

Мы видели, как микропроцессор складывает и вычитает два числа. Теперь мы увидим, что он может также умножать и делить. Инструкция умножения называется "MUL", а машинный код для умножения AX на BX - F7h E3h. Мы введем его в память, но сначала несколько слов об инструкции MUL.

Где инструкция MUL сохраняет ответ? В регистре AX? Не совсем, здесь надо быть аккуратными. Как вы скоро увидите, умножение двух 16-битных чисел может дать 32-разрядный ответ, так что инструкция MUL сохраняет резуль-

тат в двух регистрах DX и AX. Старшие 16 бит помещаются в регистре DX, а младшие - в AX. Мы можем также вписать эту комбинацию регистров как DX : AX.

Давайте вернемся к Debug и к микропроцессору. Введите инструкцию умножений F7h E3h по адресу 0100h, как вы это делали для инструкции сложения и вычитания, и установите AX=7C4Bh и BX=100h. Вы увидите инструкцию в листинге регистров как "MUL BX", без всяких ссылок на регистр AX. При умножении слов микропроцессор всегда умножает регистр, имя которого вы указываете в инструкции, на регистр AX, и сохраняет ответ в паре регистров DX:AX.

Перед тем, как мы запустим эту инструкцию умножения, давайте произведем умножение вручную. Как мы можем подсчитать 100h * 7C4Bh? Три цифры 100 имеют в шестнадцатеричной системе такой же эффект, как и в десятичной, так что умножение на 100h просто добавит три нуля справа от шестнадцатеричного числа. Таким образом, 100h * 7C4Bh = 7C4B000h. Этот результат слишком длинен для того, чтобы поместиться в одном слове, поэтому мы разбиваем его на два слова 007Ch и 4B00h.

Используйте Debug для запуска инструкции. Вы увидите, что DX содержит слово 007Ch, и AX содержит слово 4B00h. Другими словами, микропроцессор возвращает результат инструкции умножения слов в паре регистров DX:AX. Там как результат умножения двух слов не может быть длиннее двух слов, но часто бывает длиннее одного слова (как мы только что видели), инструкция умножения слов всегда возвращает ответ в паре регистров DX:AX.

А как насчет деления? Когда мы делим числа, микропроцессор сохраняет как результат, так и остаток от деления. Посмотрим на выполнение деления в МП.

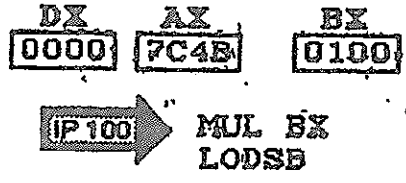


Рис. 6.5. Перед выполнением инструкции умножения.

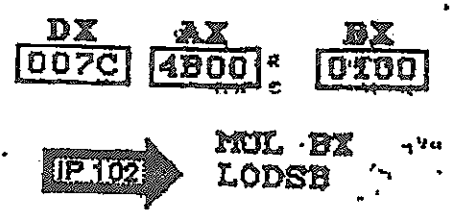


Рис.6.6. После выполнения инструкции умножения.

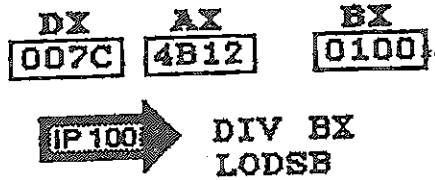


Рис.6.7. Перед выполнением инструкции деления.

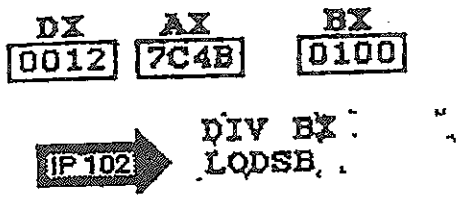


Рис.6.8. После исполнения инструкции деления.

Поместим инструкцию F7h F3h по адресу 0100h (и 101h). Как и инструкция MUL, DIV использует, пару регистров DX:AX, не сообщая об этом, так что все, что мы видим - это "DIV BX". Загрузим в регистры значения: DX = 007Ch и AX=4B12h; регистр BX по-прежнему должен содержать 0100h.

Подсчитаем результат вручную: 7C4B12h/100h= 7C4Bh с остатком 12h. После выполнения инструкции деления по адресу 0100h мы получим для AX=7C4Bh результат нашего деления и для DX =0012h, остаток.

6.9. Вывод символов на экран

Теперь, мы узнали достаточно для того, чтобы сделать что-либо более основательное. Начнем с того, что заставим OS вывести символ на экран, а затем займемся еще более ин-

тересной работой - создадим программу, состоящую из нескольких .инструкций, т.е. с ее помощью научимся еще одному способу записи данных в регистры. Посмотрим, сумеем ли мы достигнуть того, чтобы OS заговорил.

INT - мощное прерывание

К четырем математическим инструкциям ADD, SUB, MUL и DIV мы добавим новую инструкцию, называемую "INT" (от англ. "Interrupt" - прерывание).

INT немного похожа на оператор Бейсика GOSUB. Мы будем использовать инструкцию для того, чтобы заставить OS напечатать символ на экране.

Перед тем, как мы узнаем о том, как работает INT, рассмотрим один пример. Запустите Debug и поместите 20h в AX и 41 h в DX. Инструкция INT для функций OS имеет вид "INT 21 h", в машинном коде CDh 21h. Как и инструкция DIV из последней главы, это двухбайтная инструкция. Поместите "INT 21h" в память, начиная с адреса 100h, и используйте команду "R", чтобы убедиться, что инструкция читается как "INT 21h" (не забудьте установить IP в 100h, если вы этого еще не сделали).

Теперь мы готовы выполнить эту инструкцию, но не сможем, как в предыдущей главе, использовать команду трассировки, потому что она выполняет одну инструкцию за шаг, а инструкция INT вызывает большую программу из OS, выполняющую некоторые действия. При этом INT работает почти так же, как программы в Бейсике, которые могут вызывать подпрограммы с помощью оператора GOSUB.

Мы не собираемся выполнять каждую из инструкций целой "подпрограммы" OS с помощью трассировки. Напротив, мы хотим запустить нашу программу, состоящую из одной строки, но остановиться перед выполнением инструкции, размещенной по адресу 102h. Мы можем это сделать с помощью команды Debug "'G" (сокр. of англ. "Go"), после которой пишется адрес, на котором мы хотим остановиться:

```
-G 102
AX = 0241  BX = 0000  CX = 00000  DX=0000  SP =  FFEE
BP = 0000
SI = 0000  DI= 0000
DS = 3970  ES = 3970  SS = 3970  CS= 3970 IP = 0102 NV
```

UP DI PL NZ NA PO NC
3970:0102 8BE5 MOV SP, BP

OS напечатал букву "A" и; затем возвратил управление в нашу программу (учтите, что инструкция, размещенная по адресу 102h, является данными, оставшимися от другой программы, так что последняя строка вашего листинга может выглядеть по другому.)

Программа в некотором смысле состоит из двух инструкций, вторая из которых расположена по адресу 102h, то есть имеется что-то вроде этого:

INT 21

MOV SB, BP (либо что-то похоже, на вашем компьютере)

В дальнейшем мы заменим эту случайную вторую инструкцию на одну из известных, но сейчас мы предложили Debug запустить нашу программу, остановить выполнение, когда она дойдет до этой второй инструкции, и показать содержимое регистров, что и было сделано.

А как OS узнал, что нужно печатать "A"? Значение 02h в регистре AH говорит OS, что надо напечатать некоторый символ. Иное значение этого регистра сообщает OS о том, что нужно выполнить другую функцию. (Часть функций мы рассмотрим позже, но если вам любопытно, вы можете найти список этих функций, а руководстве по OS.)

Для самого же символа OS использует число в регистре DL, рассматриваемое как ASCII код символа, выводимого на экран. Для вывода на экран заглавной буквы "A" в регистр DL заносится значение 41 h, ASCII код заглавной буквы "A".

Коды символов ASCII показаны в таблице № для всех символов, которые может вывести на экран ваш IBM PC. Значения кодов представлены как в десятичном, так и в шестнадцатеричном виде. В связи с тем, что Debug понимает только шестнадцатеричные числа, работа с выводом символов н. экран является хорошей практикой перевода десятичных чисел в шестнадцатеричные. Возьмите символ из таблицы и переведите самостоятельно его код в шестнадцатеричную форму. Теперь, проверьте свой перевод, введя, полученное шестнадцатеричное число в регистр DL и снова запустив инструкцию INT (помните о необходимости установки IP в l00h).

Результат применения команды трассировки к инструкции INT может вызвать удивление своей непредсказуемостью. Допустим, мы не исполняли команды "G 102", а про-

извели трассировку для того, чтобы увидеть, что же произойдет. Если вы. решили попробовать выполнить такую процедуру самостоятельно, то не заходите слишком далеко, так как можете увидеть, что ваш компьютер начнет вытворять нечто странное. После того как вы, протрассируете несколько шагов, выйдете из Debug с помощью команды "Q". Это ликвидирует беспорядок, который вы за собой оставите.

```
-R
AX = 0200 BX = 092A CX = 00000 DX = 0041 SP = FFEE BP = 0000
SI=0000 DI=0000
DS=3970  ES= 3970 SS = 3970 CS = 3970 IP= 0100 NV UP DI PL NZ
NA PO NC
    3970:0100 CD21
-T
    AX = 0200 BX = 092A CX = 00000 DX = 0041 SP=FFE8 BP=0000
    SI = 0000 DI = 0000
DS =3970 ES =3970 SS = 3970 CS =3372 IP = 0180  NV UP DI PL NZ
NA
PO NC
3372:0180 80FC4B CMP AH, 4B
-T
    AX = 0200 BX = 092A CX = 00000 DX= 0041 SP=FFEE BP=0000
    SI = 0000 DI = 0000
    DS = 3970 ES = 3970 SS = 3970 CS = 3372 IP = 0183 NV  UP DI PL
NZ NA
    PO NC
    3372:01837405  JZ 018A
-T
    AX=0200 BX=092A CX =00000 DX =0041 SP = FFEE   BP = 0000'
SI =0000 DI=0000
DS = 3970 ES = 3970 SS = 3970 CS = 3372 IP = 0185 NV UP  DI PL NZ
     NA
PO NC
3372:0185 2E CS:
3372:0186 FF2EABOB   JMP FAR [OBAB] CS:OBAB= OBFF
```

Обратите внимание на, то, что первое число, являющееся составляющей адреса, изменилось с 3970 на 3372. Три последние инструкции являются частью OS, а программа ,для OS, находится в другом сегменте. Фактически существует очень много инструкций, которые OS выполняет перед тем, как напечатать символ; поэтому даже такая, кажущаяся про-

стой задача, не так проста. Теперь стало ясно, почему мы использовали команду "G" для выполнения программы только до адреса 102h. В противном случае мы бы увидели лавину инструкций от OS. (Если у вас иная версия OS, то инструкции, которые вы увидите при трассировке, могут быть другими.).

6.9.1. Инструкция прерывания - INT 20h

Помните, что наша инструкция INT была 21h? Если мы изменим 21h на 20h, мы получим "INT 20h", другую инструкцию прерывания, которая сообщает OS о том, что мы хотим выйти из нашей программы и чтобы управление опять вернулось к OS. В нашем случае "INT 20h" вернет управление к Debug, так как мы выполняем нашу программу из Debug, а не из OS.

Введите инструкцию CDh 20h, начиная с адреса 100h, а затем попробуйте проделать следующее (только не забудьте проверить инструкцию "INT 20h" с помощью команды "R"):

```
-G 102
Program terminated normally
-R
AX = 0000 BX = 0000 CX = 00000 DX = 0000 SP = FFEE BP = 0000
SI = 0000 DI=0000
DS = 3970 ES = 3970 SS = 3970  CS = 3970 IP = 0100 NV UP DI PL NZ
NA
PO NC
3970:0100 CD20 INT 20 ,
-G
Program terminated normally
-R
AX = 0000 BX = 0000 CX = 00000 DX= 0000 SP = FFEE BP = 0000
SI = 0000 DI = 0000
DS = 3970  ES = 3970 SS = 3970  CS= 3970 IP = 0100 NV  UP DI PI  NZ
NA
PO NC
   3970:0100 CD20  INT 20
```

Команда "G" выполняет всю программу, которая сейчас состоит всего из одной инструкции, так как "INT 20" — это инструкция выхода), и затем возвращается к началу. Когда мы начали выполнение, IP был установлен в 100h. Значения

регистров в этом примере равны 0 только потому, что мы заново запустили Debug.

Мы можем ввести инструкцию INT 20h в конец программы так, чтобы грациозно передать управление OS (или Debug) и поэтому сейчас введем эту инструкцию вместе с "INT 21h в программу, состоящую из двух строк.

6.9.2. Программа из двух строк - соединение частей вместе

Начиная с адреса; 100h, введите две инструкции "INT 21h", "INT 20h" (CDh 21h CDh 20h) одну за другой (мы теперь всегда будем начинать программы *с* адреса 100h).

Когда у нас была только одна инструкция, то мы могли "пролистать" эту инструкцию командой "R", но теперь у нас две инструкции. Чтобы увидеть их, у нас есть команда "U" (от англ. "Unassemble"), которая работает аналогично команде Бейсика "List":

```
-U 100
3970:0100 CD21              INT 21
3970:0102 CD20              INT 20
3970:0104 D98D460250B8          ESC     09,[D1 + 024<][DI +
B850]
3970:010A   8D00            LEA     AX,[BX+S1]
3970:0100   50              PUSH    AX
3970:010D   E82A23          CALL    243A
3970:0110   8BE5            MOV     SP,BP
3970:0112   8EC41A          ADD     SP,+1A
3970:0115   5D              POPBP
3970:0116   C3              RET
3970:0117   55              PUSH    BP
3970:0118   83EC02          SUBSP.+02
3970:011 B  8BEC            MOV     BP,SP
3970:011D   823E0E0000  CMP BYTE PTR (OOOE),00
```

Первые две инструкции, которые представлены в списке, являются инструкциями, которые мы только что ввели. Остальные инструкции остались в памяти от предыдущих программ. По мере того как наша программа будет расти, количество выводимых на экран кодов также будет увеличиваться.

Поместите в регистр AH значение 02h, а в регистр DL код любого символа (как вы это делали раньше, когда изменяли регистры AX и DX), и затем напечатайте команду "G",

чтобы увидеть ваш символ на экране. Например, если вы поместили в DL число 41 H, то вы увидите:

```
-G
A
Program terminated normally
-
```

Попробуйте вывести на экран еще несколько символов, пока мы не перейдем к изучению другого способа установки этих регистров.

6.9.3. Ввод программ

Начиная с этой страницы, большая часть программ будет иметь в длину более чем одну инструкцию, и чтобы просмотреть эти программы, мы будем использовать инструкцию разассемблирования ('U"), поэтому наша последняя программа появится в виде:

```
3970:0100    CD21    INT 21
3970:0102    CD20    INT 20
```

До этого мы вводили инструкции программ в виде чисел, например CDh, 21h. Но это слишком тяжелая работа, и как оказывается, имеется более простой способ ввода инструкций.

В дополнение к команде разассемблироваиия в программе Debug имеется команда, позволяющая вводить мнемонические, или человекочитаемые, инструкции. Так что вместо того, чтобы вводить непонятные числа программы, мы можем применить команду ассемблирования для следующего ввода:

```
-A 100
3970:0100   INT 21
3970:0102   INT 20
3970:0104
```

После ввода инструкций необходимо нажать клавишу "Enter", и вновь появится приглашение Debug.

Команда "A" сообщает Debug о том, что мы хотим ввести инструкции в мнемонической форме, а число 100 в команде означает, что ввод инструкций начнется с ячейки 100h. Команда ассемблирования значительно упрощает ввод программ.

Использование команды MOV для пересылки данных между регистрами.

Хотя раньше мы во всем полагались на Debug, мы не всегда будем запускать программы с ее помощью. Обычно программа сама устанавливает регистры AH и DL перед инструкцией "INT 21h". Чтобы уметь это сделать, мы изучим еще одну инструкцию, MOV. Применение инструкции MOV позволит создавать программы, способные запускаться непосредственно из OS. В дальнейшем мы будем использовать инструкцию MOV для того, чтобы загружать числа в регистры AH, и DL. Начнем изучение MOV с осуществления пересылки чисел между регистрами. Поместите 1234h в AX (12h в регистр AH и 34h в AL) и ABCDh в DX (ABh в DH и CDh в DL;). С помощью команды "A" введите инструкцию:

396F:0100 88D4 MOV AH, DL

Эта инструкция пересылает число из DL в AH, копируя его в AH, AL при этом не используется. Если вы протрассируете эту строку, то увидите, что AX=CD34h и DX= ABCDh. Изменился только AH. Теперь он содержит копию числа из DL.

Как и оператор Бейсика "LET AH=DL", инструкция MOV пересылает число из второго регистра в первый, и по этой причине мы пишем AH перед DL. Несмотря на то, что имеются некоторые ограничения, о которых мы поговорим позже, мы можем применить иные формы инструкции MOV, для копирования чисел между парами регистров. Например, переустановите IP и попробуйте ввести следующее:

396F:0100 89C3 MOV BX, AX

Вы только что загрузили из регистра в регистр слово, а не байт. Инструкция MOV всегда копирует или слова или байты, но никогда слова в байты. Это вполне понятно: как вы загрузите слово в байт?

В начале мы установили загрузку числа из регистре AH в регистр DL. Проделаем то же самое с другой формой инструкции MOV:

396F:0100 B402 MOV byte ptr AH,02

Эта инструкция загружает число 02h в регистр AH без применения регистра AL. Второй байт инструкции, 02h является числом, которое мы хотим загрузить. Попробуйте загрузить в AH другое число с помощью команды "E 101" измените второй байт, чтобы он был равен, например, C1h.

Сложим все части вместе и построим длинную программу. Она будет печатать звездочку, "*",выполняя все опе-

рации сама, не требуя от нас установки регистров (AH и DL). Программа использует инструкции MOV для того, чтобы установить регистры AH и DL перед вызовом INT 21п из OS:

```
396F:0100   B402    MOV   byte ptr AH, 02
396F:0102   B22A    MOV   byte ptr DL, 2A
396F:0104   CD21    INT 21
396F:0106   CD20    INT 20
```

Введите программу и проверьте ее командой "U"("U 100"). Убедитесь, что IP указывает на ячейку 100h, затем попробуйте командой "G" запустить ее. В итоге на вашем экране должен появиться символ "*".

```
-G
*
Program terminated normally
```

У нас есть законченная, содержательная программа, запишем ее на диск в виде .COM файла для того, чтобы мы могли запускать ее прямо из OS. Это можно сделать просто набрав ее имя. Так как у программы нет имени, то мы должны его присвоить.

Команда Debug "N" (сокр. от англ. "Name") присваивает файлу имя перед записью на диск. Напечатайте:

```
-N WRITESTR.COM
```

Эта команда не запишет файл на диск - она только назовет его **WRITESTR.COM**.

Далее, мы должны сообщить Debug о том, сколько байт занимает программа для того, чтобы, он знал размер файла. Если вы посмотрите на разассемблированный листинг программы, то вы увидите, что каждая инструкция в нем занимает два байта (что в общем не всегда верно). У нас четыре инструкции, следовательно программа имеет длину 4*2=8 байт. (Мы могли бы также задействовать команду Debug "H". Напечатав "H 108 IDO", где 108 -адрес инструкции после INT 20, мы получим 8.)

Так как у нас теперь есть число байт, то нам надо куданибудь его записать. Debug-для этого использует пару регистров BX:CX, и поэтому, поместив 8h в CX, мы сообщим Debug о том, что программа имеет длину в восемь байт. BX должен быть предварительно установлен в ноль.

После того как мы установили имя N; длину программы, мы можем записать ее на диск с помощью команды Debug "W" (от анг. "Write"):

```
-    W
Writing 0008 bytes
```

Теперь на диске есть программа WRITESTR.COM, и мы сейчас с помощью "Q" покинем Debug и посмотрим на нее. Используйте команду OS Dir, чтобы увидеть файл:

```
C> DIR  WRITESTR.COM
Volume in drive A has no label
Directory of A:\
WRITESTR.COM      8 6-30-83  10:05a
1 File(s)              18432 bytes free
```

Листинг директории сообщает, что WRITESTR.COM находится на диске и его длина составляет восемь байт, как и должно быть. Чтобы запустить программу, напечатайте "Writestr" в ответ на приглашение OS и нажмите "Enter". Вы увидите "*", появившуюся на дисплее. Вот и все.

6.9.4. Вывод на экран строки символов

В качестве последнего примера в этой главе мы используем "INT 21h" с другим номером функции в регистре AH для того, чтобы вывести на экран целую строку символов. Мы сохраним эту строку в памяти и затем сообщим OS, где ее искать, так что в процессе работы над такой программой мы познакомимся более близко с понятиями адреса и памяти. Как мы уже отметили, функция номер 02h для прерывания "INT 21 h" печатает один символ на экране. Другая функция, номер 09h, печатает целую строку и прекращает печать, когда находит символ "$". Давайте поместим строку в память. Мы начнем с ячейки 200h, так что запись строки не перепутается с кодом самой программы. Введите следующие числа, используя инструкцию "E 200":

```
48  65  6C  6C
6F  2C  20  44
4F  53  20  68
65  72  65  2E
24
```

Последнее число 24h является ASCII-кодом для знака "$", и оно сообщает OS, что это конец строки символов. Через минуту вы увидите, что сообщает эта строка, запустив программу, которую сейчас введете:

```
396F:0100   B409          MOV   byte ptr AH, 09
396F:0102   BA0002  MOV     byte ptr DX, 0200
   396F:0105     CD21          INT 20
   396F:0107     CD20          INT 21
```

200h это адрес строки, которую мы ввели, а загрузка 200h в регистр DX сообщает OS о том, где ее искать. Проверьте программу командой "U" и затем запустите ее командой "G":

```
-G
   Hello, DOS here.
   Program terminated normalty
```

Мы сохранили в памяти несколько введенных символов, и теперь самое время познакомиться с другой командой Debug, "D" (от англ. "Dump"). То, как эта команда демпирует (выводит содержимое) памяти на экран, похоже на действия, совершаемые командой "U" при распечатке инструкций. Также, как и при использовании команды "U", поместите адрес после "D", чтобы сообщить Debug, откуда начинать дамп. Например, команда "D 200" выводит содержимое участка памяти, в котором хранится только что введенная строка.

```
-  D 200
396F:0200 48 65 6C 6C 6F 2C 20 44-4F 53 20 68 65 72 65 2E Hello, Dos
here
396F:0210 24 5D C3 55 83  EC 30 8B-EC C7 06 10 00 00 00 E8
$]CU.10.1G.....h
```

После каждого числа, обозначающего адрес (как 396F:0200 в нашем примере), мы видим 16 шестнадцатеричных байт, вслед за которыми записаны 16 ASCII-символов для этих байт таблица №.6.1 Так например, в первой строке вы видите почти все ASCII-коды и символы, которые вы ввели. Символ "$" является первым символом в следующей строке, остальная часть строки представляет собой беспорядочный набор символов.

Когда вы видите точку (".") в окне ASCII, знайте, что это может быть как точка, так и специальный символ, например, греческая буква "pi". Команда Debug "D" выдает только 96 из 256 символов символьного набора IBM PC, поэтому точка используется для обозначения остальных 160 символов.

В дальнейшем мы будем использовать команду "D" для проверки чисел, введенных в качестве данных, независимо от того, являются ли эти данные символами или обычными числами. (Если вам необходима более глубокая информация на эту тему, смотрите ту часть руководства по OS, которая посвящена Debug.)

Программа, выводящая строку на экран, закончена, так что мы можем записать ее на диск. Процедура записи та же, которую мы использовали для записи на диск WRITESTR.COM, за исключением того, что на этот раз нам нужно установить значение длины программы, достаточное для того, чтобы включить строку по адресу 200h. Программа начинается со строки 100h, и из только что выполненного дампа памяти можно видеть, что символ, следующий за знаком "$", заканчивающим нашу строку, расположен по адресу 211h. Как и раньше, мы можем использовать команду "H", чтобы определить разность между этими двумя числами. Найдите 211h-100h и сохраните это значение в регистре CX, опять установив BX в ноль. Используйте команду "N", чтобы дать имя программе (добавьте расширение .COM, чтобы запускать программу прямо из OS), и затем командой "W" запишите программу и данные в дисковый файл. Вот и все о выводе символов, на экран, кроме одного последнего замечания: вы должны помнить, что OS не печатает символа "$". Это происходит потому, что OS использует этот знак для пометки конца строки символов. Следовательно, мы не можем прямо использовать OS, чтобы напечатать строку, содержащую "$", однако, в последующих главах мы увидим, как обойти это препятствие.

6.10. Создание программы без использования Debug

До этого момента мы сначала запускали Debug, а затем вводили инструкции программы. Теперь, мы оставим Debug в покое и станем создавать программы без него, используя для их написания любой текстовый редактор. Программы будут иметь более удобочитаемый вид, упрощающий их восприятие. Мы начнем с создания исходного файла - так называется текстовая версия ассемблерной программы. Сейчас мы создадим исходный файл для программы Writestr, которую мы написали в параграфе 6.9.3. Чтобы освежить вашу память, при-

ведем версию программы, созданной с помощью Debug:

```
396F:0100  B402    MOV   byte ptr AH, 02
396F:0102  B261    MOV   byte ptr DL, 2A
396F:0104  CD20    INT   21
396F:0106  CD20    INT   20
```

Используйте любой текстовый редактор для ввода приведенных ниже строк в файл под названием "WRITESTR.ASM" (расширение .ASM означает, что это ассемблерный исходный файл). Здесь, как и в Debug, строчные буквы работают так же хорошо, как заглавные, но мы будем продолжать применять заглавные буквы для того, чтобы избежать путаницы между цифрой "1" (единица) и строчной буквой "I":

```
CODE_SEG      SEGMENT
   MOV        AH, 2h
      MOV     DL, 2Ah
      INT     21h
      INT     20h
CODE-SEG          ENDS
                  END
```

Это та же самая программа, которую мы создали в параграфе 6.9.3, но она содержит некоторые необходимые изменения и дополнения. Не. обращая пока внимания на три новые строчки в нашем исходном файле заметим, что, перед каждым шестнадцатеричным числом стоит "h". Эта буква говорит ассемблеру о, том, что число, после которого она стоит, шестнадцатеричное. В отличие от Debug, который воспринимает все числа шестнадцатеричными, ассемблер предполагает их десятичными. Буква h сообщает ассемблеру о том, что введенное число является шестнадцатеричным.

Примечание: Прежде чем двинуться дальше, хотим предупредить: ассемблер может запутаться с числами, подобными ACh, начинающимися с буквы и похожими на имя или инструкцию. Чтобы избежать этого, печатайте нуль перед шестнадцатеричным числом начинающимся с буквы. Например, печатайте 0ACh, а не ACh.

Посмотрите, что произойдет, если мы напишем в программе ACh, а не 0ACh. Вот программа:

```
CODE_SEG        SEGMENT
MOV             DL, ACh
INT         20h
CODE_SEG        ENDS
END
```

А вот ответ ассемблера:

```
C»MASM TEST;
Microsoft( R) Macro Assembler Version 4.00
Copyrihgt (C) Microsoft Corp 1981,1983.1984,1985. All rights reserved.
TEST.ASM(2): error 9: Symbol not defined AC
51070 Bytes Symbol space free
    0 Warning  Errors
        1 Severe Errors
C»
```

Это определенно не похвала. Но, изменив ACh на 0ACh, мы удовлетворим ассемблер.

Скажем также о разделении команд в ассемблерных программах. Мы использовали символ «TAB» для того, чтобы сделать исходный текст лучше читаемым. Сравните программу, которую вы ввели, например, с такой версией:

```
CODE_SEG        SEGMENT
MOV             AH, 2h
MOV             DL, 2Ah
INT         21h
INT         20h
CODE_SEG        ENDS
END
```

Выглядит достаточно неупорядоченно и, хотя самому ассемблеру ее внешний вид безразличен, аккуратно введенный текст программы настраивает на серьезное к ней отношение.

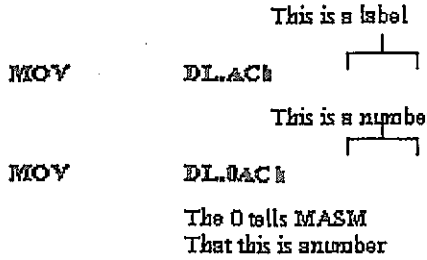


Рис. 6.9.

Теперь мы вернемся к трем новым строкам в исходном файле. Эти строки являются Псевдооператорами. Они называются так потому, что в отличие от инструкций, генерирующих код, они всего лишь сообщают ассемблеру дополнительную информацию Псевдооператор END означает окончание исходного текста, по его наличию ассемблер узнает о том, что работа окончена. Позднее мы узнаем, что END полезен также и для других целей. Сейчас мы отложим в сторону дальнейшие дискуссии об этом псевдооператоре и двух других и посмотрим, как использовать сам ассемблер.

6.11. Создание исходных файлов

После того, как вы ввели строки WRITESTR.ASM, надо упомянуть еще одно соглашение перед тем, как мы перейдем к непосредственному ассемблированию нашей программы. Ассемблер может использовать такие исходные файлы, которые содержат только стандартные ASCII-символы. Если вы применяете для создания исходного файла текстовый процессор, то имейте в виду, что не все текстовые процессоры записывают файлы, используя только стандартные ASCII-символы. Word Star является первым таким "преступником", Microsoft Word - вторым. Для этих процессоров используйте специальный недокументный или неформатированный режим записи файла. Перед тем, как вы попробуете ассемблировать WRITESTR.ASM, убедитесь, что это именно ASCII-файл. Находясь в OS, напечатайте:

C»TYPE WRITESTR.ASM

Вы должны увидеть тот же текст, который ввели в текстовом редакторе. Если вы увидите в вашей программе странные символы, то для ввода текста программ нужно использовать другой текстовый редактор. Кроме того, в файле необходимо оставлять пустую строку после оператора END.

Теперь давайте начнем ассемблировать Writestr для чего найдем каталог MASM войдем в него, найдем в нем masm.exe, нажатием двух клавиш Ctr+Enter наберем его в командной строке и затем наберем наш ассемблерный файл.

```
C»MASM WRITESTR.ASM;
The IBM Personal Computer Assembler
Version 1.00 (C) Copyright IBM Corp 1981
Warning    Severe
Errors     Errors
0          0
C»
```

Работа еще не закончена. На этом этапе ассемблер создал файл, называющимся WRITESTR. OBJ, который вы найдете на диске. Это промежуточный файл, называющийся объектным файлом. Он содержит программу на машинном языке, вместе с большим количеством "бухгалтерской" информации, используемой другой программой, называющейся Linker (редактор связей, компоновщик).

6.12. Компоновка

Теперь необходимо, чтобы редактор связей взял .OBJ-файл и сделал его .EXE-версию. Скопируйте LINK EXE с вашего OS диска на диск, содержащий исходный файл и ассемблер. Затем следующее:

```
C»LINK WRITESTR;

IBM Personal Computer Linker
Version 1.10 (C) Copyright IBM Corp 1982

Warning: No STACK segment
There was 1 error detected
C»
```

Ошибка? В действительности нет. (В некоторых версиях

MS-DOS редактор связей (linker) вообще не считает это сообщение за ошибку.) Несмотря на то, что компоновщик предупреждает нас о том, что отсутствует сегмент стека, он нам и не нужен. После того, как мы узнаем, каким образом можно использовать все особенности микропроцессора, мы увидим, почему нам может позже понадобиться сегмент стека.

Наконец, у нас есть .EXE -файл, но и это еще нее все. Нам надо сделать еще один шаг - создать .COM-версию, которая является тем же самым, что мы создали с помощью Debug. Опять-таки, вы увидите позже, почему нам нужны все эти шаги. А сейчас давайте создадим .COM- версию Writestr.

Для последнего шага нам понадобится программа EXE2BIN.EXE с дополнительного диска OS. Exc2bin, как и предполагает ее название, превращает .EXE-файл в .COM, или двоичный (от англ. "binary") файл. Существуют некоторые различия между .EXE и .COM- файлами, мы будем разбирать их значительно позже, а сейчас создадим .COM-файл. Напечатайте:

```
C»EXE2BIN WRITESTR WRITESTR.COM
C»
```

Лаконичный ответ. Чтобы увидеть, сработала ли программа Exe2bin, давайте посмотрим на файлы с именем Writestr, которые мы создали:

```
C»DIR WRITESTR."
Volume in drive A has label
Directory of C:\

WRITESTR  ASM 78 7-25-635:00p
WRITESTR  OBJ 46  7-25-637:02p
WRITESTR  EXE 640 7-25-637:04p
WRITESTR  COM 8      7-25-637:06p
      4 Files 23552 bytes free
C»
```

Это точное число файлов, включая WRITESTR:COM. Напечатайте "writestr", чтобы запустить .COM- версию и убедитесь, что *ваша* программа функционирует правильно (напоминаем, что она должна печатать звездочку на экране). Точная длина первых трех файлов может немного варьироваться.

Результаты могут показаться немного разочаровывающими, так как кажется, что вернулись к результату в главе

6.9.4, но это не так: мы на самом деле сделали большое дело. Это станет ясно когда мы снова начнем работать с вызовами. Обратите внимание, что мы еще ни разу пока не побеспокоились о том, где наша программа, будет помещена в память как мы это делали с помощью IP в Debug. Все, проблемы размещения в памяти были решены, без нашего участия.

Очень скоро вы оцените эту способность ассемблера. Она сделает программирование гораздо проще. Например, помните, в последней главе мы потратили память, поместив основную программу по адресу 200h. Мы увидим, что ассемблер позволяет нам помещать процедуры непосредственно после основной программы без какого-либо пропуска. Посмотрим, как наша программа выглядит в Debug.

6.13. Обратно в Debug

Давайте введем созданный COM-файл обратно в Debug и разассемблируем его, чтобы увидеть, как Debug реконструирует программу из машинного кода WRITESTR.COM:

```
C> DEBUG WRITESTR.COM
-U
397F:0100   B402    MOV   AH, 02
397F:0102   B261    MOV   DL, 2A
397F:0104   CD21    INT 21
397F:0100   CD20    INT 20
```

Получили именно то, что мы уже имели в параграфе 6.9.3. Это все, что Debug видит в WRITESTR.COM. Оператор END и дополнительные инструкции о сегментах – CODE_SEG SEGMENT и CODE_SEG ENDS -отсутствует. Что же с ними произошло?

Эти инструкции не появились и в последней, написанной на машинном языке, версии программы, так как они являются псевдооператорами, а такие операторы служат только для дополнительной информации, необходимой ассемблеру для корректного функционирования. Ассемблер заботится о большом количестве "бухгалтерии" всего для нескольких оттранслированных строк.

Контрольные вопросы

1. Составить программу сложения, вычитания чисел (числа выбрать самостоятельно).
2. Составить программу сложения, вычитания двух длинных чисел.
3. Составить программу умножения, деления чисел.
4. Составить программу управления и контроля объектами систем автоматики, телемеханики и связи.

ЛИТЕРАТУРА

1. Л. Левенталь. «Введение в микропроцессоры».М., 1995
2. П. Нортон, Д.Соухэ. «Язык ассемблера для IBM PC».М., 1996
3. Р. Тохтайм. «Микропроцессоры. Курс и упражнения». М.,1996.
4. М.М. Алиев «Микропроцессорлар ва улардан автоматика ва телемеханика курилмаларида фойдаланиш». Т., 1992.
5. Ю.Н. Казаринов, В.Н. Номонконов, В.Н. Филиппов. «Применение микропроцессоров и микро-ЭВМ в радиотехнических системах». М., 1996.
6. В.Л. Григорьев Программирование однокристальных микро-процессоров. —М., 1987.
7. http://www.Intel.com
8. http://www.Intel.ru
9. http://www.IBM.com
10. Зельдин Е.А. Цифровые интегральные микросхемы в инфор-мационно-измерительной аппаратуре. —М.: Энергоатомиздат, 1986. - 280 с.
11. Шило В.А. Популярные цифровые микросхемы: Справочник. —М.: Радио и связь, 1989. — 352 с.

СОДЕРЖАНИЕ

М.М. АЛИЕВ

ЦИФРОВАЯ ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И МИКРОПРОЦЕССОРЫ

Ташкент – «Fan va texnologiya» –2009

Редактор: Ж.Тураханов
Тех.редактор: А.Мойдинов
Корректор: Д.Вахидова
Компьютерная верстка: Ш.Миркасимова

Разрешено в печать 30.09.09. Формат 60x84 1\16. Гарнитур
«Times Uz». Печать офсетная. Усл.п.л. 10,25. Изд.п.л. 10,(
Тираж 500. Заказ № 122.

Отпечатано в типографии
«Fan va texnologiyalar Markazining bosmaxonasi».
100003, г. Ташкент, ул. Алмазар, 171.