МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
Інженерний навчально-науковий інститут
Кафедра автоматизованого управління технологічними процесами

А.М.Ніколаєнко

Мікропроцесорна техніка в автоматизованих системах

Конспект лекцій

для студентів напряму 151 "Автоматизація та комп'ютерно-інтегровані технології" денної та заочної форм навчання.

СХВАЛЕНО
на засіданні кафедри автоматизованого
управління технологічними процесами
Протокол № 1
від 30.09.2020р.

Запоріжжя

Тема 1. Історія виникнення та класифікація мікропроцесорів.

История развития микропроцессоров.

Электронные вычислительные машины широко используются с 50-х годов. Вначале это были ламповые и дорогие машины, предназначенные для административно-управленческих целей, доступные крупным предприятиям. В последние годы структура и формы вычислительных машин изменились из-за появления нового элемента — микропроцессора. Микропроцессор — это интегральная схема (ИС), обладающая такой же производительностью при переработке информации, что и большая ЭВМ. Более точно – это программно-управляемое электронное цифровое устройство, предназначенное для обработки цифровой информации и управления процессом этой обработки, выполненное на одной или нескольких интегральных схемах с высокой степенью интеграции электронных элементов.

В 1970 году Маршиан Эдвард Хофф из фирмы Intel сконструировал интегральную схему, аналогичную по своим функциям центральному процессору большой ЭВМ, – первый микропроцессор Intel-4004, который уже в 1971 году был выпущен в продажу.

15 ноября 1971 г. можно считать началом новой эры в электронике. В этот день компания приступила к поставкам первого в мире микропроцессора Intel 4004.

Это был настоящий прорыв, ибо МП Intel-4004 размером менее 3 см был производительнее гигантской машины ENIAC. Правда работал он гораздо медленнее и мог обрабатывать одновременно только 4 бита информации (процессоры больших ЭВМ обрабатывали 16 или 32 бита одновременно), но и стоил первый МП в десятки тысяч раз дешевле. Кристалл представлял собой 4-разрядный процессор с классической архитектурой ЭВМ гарвардского типа и изготавливался по передовой p-канальной МОП-технологии с проектными нормами 10 мкм. Электрическая схема прибора насчитывала 2300 транзисторов. МП работал на тактовой частоте 750 кГц при длительности цикла команд 10,8 мкс. Чип i4004 имел адресный стек (счетчик команд и три регистра стека типа LIFO), блок РОНов (регистры сверхоперативной памяти или регистровый файл – РФ), 4-разрядное параллельное АЛУ, аккумулятор, регистр команд с дешифратором команд и схемой управления, а также схему связи с внешними устройствами. Все эти функциональные узлы объединялись между собой 4-разрядной ШД. Память команд достигала 4 кб (для сравнения: объем ЗУ миниЭВМ в начале 70-х годов редко превышал 16 кб), а РФ ЦП насчитывал шестнадцать 4-разрядных регистров, которые можно было использовать и как восемь 8-разрядных. Такая организация РОНов сохранена и в последующих МП фирмы Intel. Три регистра стека обеспечивали три уровня вложения подпрограмм. МП i4004 монтировался в пластмассовый или металлокерамический корпус типа DIP (Dual In-line Package) всего с 16 выводами.

В систему его команд входило всего 46 инструкций. Вместе с тем кристалл располагал весьма ограниченными средствами ввода/вывода, а в системе команд отсутствовали операции логической обработки данных (И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ), в связи с чем их приходилось реализовывать с помощью специальных подпрограмм. Модуль i4004 не имел возможности останова (команды HALT) и обработки прерываний.

Цикл команды процессора состоял из 8 тактов задающего генератора. Была мультиплексированная ША (шина адреса)/ШД (шина данных), адрес 12-разрядный передавался по 4 разряда.

1 апреля 1972 г. фирма Intel начала поставки первого в отрасли 8-разрядного прибора i8008. Кристалл изготавливался по p-канальной МОП-технологии с проектными нормами 10 мкм и содержал 3500 транзисторов. Процессор работал на частоте 500 кГц при длительности машинного цикла 20 мкс (10 периодов задающего генератора).

В отличие от своих предшественников МП имел архитектуру ЭВМ принстонского типа, а в качестве памяти допускал применение комбинации ПЗУ и ОЗУ.

По сравнению с i4004 число РОН уменьшилось с 16 до 8, причем два регистра использовались для хранения адреса при косвенной адресации памяти (ограничение технологии – блок РОН аналогично кристаллам 4004 и 4040 в МП 8008 был реализован в виде динамической памяти). Почти вдвое сократилась длительность машинного цикла (с 8 до 5 состояний). Для синхронизации работы с медленными устройствами был введен сигнал готовности READY.

Система команд насчитывала 65 инструкций. МП мог адресовать память объемом 16 кб. Его производительность, по сравнению с четырехразрядными МП, возросла в 2,3 раза. В среднем для сопряжения процессора с памятью и устройствами ввода/вывода требовалось около 20 схем средней степени интеграции.

Возможности p-канальной технологии для создания сложных высокопроизводительных МП были почти исчерпаны, поэтому "направление главного удара" перенесли на n-канальную МОП-технологию.

1 апреля 1974 МП Intel 8080 был представлен вниманию всех заинтересованных лиц. Благодаря использованию технологии n-МОП с проектными нормами 6 мкм, на кристалле удалось разместить 6 тыс. транзисторов. Тактовая частота процессора была доведена до 2 МГц, а длительность цикла команд составила уже 2 мкс. Объем памяти, адресуемой процессором, был увеличен до 64кб. За счет использования выводного корпуса удалось разделить ША и ШД, общее число микросхем, требовавшихся для построения системы в минимальной конфигурации сократилось до 6.

В РФ были введены указатель стека, активно используемый при обработке прерываний, а также два программно недоступных регистра для внутренних пересылок. Блок регистров общего назначения (РОН) был реализован на микросхемах статической памяти. Исключение аккумулятора из РФ и введение его в состав АЛУ упростило схему управления внутренней шиной.

Новое в архитектуре МП – использование многоуровневой системы прерываний по вектору. Такое техническое решение позволило довести общее число источников прерываний до 256 (до появления БИС контроллеров прерываний схема формирования векторов прерываний требовала применения до 10 дополнительных чипов средней интеграции). В i8080 появился механизм прямого доступа в память (ПДП) (как ранее в универсальных ЭВМ IBM System 360 и др.).

ПДП открыл зеленую улицу для применения в микро-ЭВМ таких сложных устройств, как накопители на магнитных дисках и лентах дисплеи на ЭЛТ, которые и превратили микро-ЭВМ в полноценную вычислительную систему. Традицией

компании, начиная с первого кристалла, стал выпуск не отдельного чипа ЦП, а семейства БИС, рассчитанных на совместное использование.

При оценке параметров микропроцессора и выборе микропроцессорной серии наибольшую роль играет разрядность прибора, которая задает элементарный объем обрабатываемых данных. Чем больше разрядность, тем выше производительность и шире возможности адресации. В ранних приборах разрядность регистров, шин управления, а также информационных шин почти всегда была одинаковой. Сейчас такая структура встречается редко. Например, микропроцессор Motorola 6800 имеет 32-разрядную внутреннюю архитектуру, 16-разрядную шину данных и 24-разрядную адресную шину (адресует до 16 Мбайт оперативной памяти). Для удобства такую архитектуру называют 32/16/24.

В настоящее время стремятся к большей разрядности, например делают полную 32-разрядную архитектуру (32/32/32).

Если считать, что выпуск предыдущих микропроцессоров должен прекращаться при появлении кристаллов с более высокой разрядностью, то 4-разрядные производились бы всего 1год, 8-разрядные - 5 лет, 16-разрядные - 5 лет (табл.1.1).

Таблица1.1.Характеристики микропроцессоров.

Разрядность	Модель	Количество транзисторов, шт	Год выпуска	Торговая марка
4	4004	2200	1971	Intel
8	8008	2300	1972	Intel
8	8080	4800	1973	Intel
8	280	8400	1976	Zilog
8	8048	12400	1977	Intel
16	8086	29000	1978	Intel
16	68000	75000	1980	Motorola
16	80286	130000	1982	Intel
32	NR-9000	450000	1982	Hewlett-Packard

Однако следует оговориться: 4-разрядные микропроцессоры производятся и применяются до настоящего времени.

В начале развития микропроцессоры изготовлялись по p-МОП технологии, затем специалисты стали отдавать предпочтение комплементарной МОП-технологии (КМОП) (технологии, используемые при изготовлении процессоров, будут рассмотрены ниже в подразделе 3.1). Теперь применяется множество разнообразных видов технологии и технологических приемов при изготовлении микропроцессоров: n-МОП технология с обогащением и с обеднением, биполярная технология, технология И2Л и др. Например, за первые 20 лет развития микропроцессорной техники в США зарегистрированы 237 технологических нововведений, из них 67 революционных.

Классификация микропроцессоров

1. По числу больших интегральных схем (БИС) в микропроцессорном комплекте различают однокристальные, многокристальные и многокристальные секционные микропроцессоры.

Процессоры даже самых простых ЭВМ имеют сложную функциональную структуру, содержат большое количество электронных элементов и множество разветвленных связей. Изменять структуру процессора необходимо так, чтобы полная принципиальная схема или ее части имели количество элементов и связей, совместимое с возможностями БИС. При этом микропроцессоры приобретают внутреннюю магистральную архитектуру, т. е. в них к единой внутренней информационной магистрали подключаются все основные функциональные блоки

(арифметико-логический, рабочих регистров, стека, прерываний, интерфейса, управления и синхронизации и др.).

Для обоснования классификации микропроцессоров по числу БИС надо распределить все аппаратные блоки процессора между основными тремя функциональными частями: операционной, управляющей и интерфейсной. Сложность операционной и управляющей частей процессора определяется их разрядностью, системой команд и требованиями к системе прерываний; сложность интерфейсной части разрядностью и возможностями подключения других устройств ЭВМ (памяти, внешних устройств, датчиков и исполнительных механизмов и др.). Интерфейс процессора содержит несколько десятков информационных шин данных (ШД), адресов (ША) и управления (ШУ).

Однокристальные микропроцессоры получаются при реализации всех аппаратных средств процессора в виде одной БИС или СБИС (сверхбольшой интегральной схемы). По мере увеличения степени интеграции элементов в кристалле и числа выводов корпуса параметры однокристальных микропроцессоров улучшаются. Однако возможности однокристальных микропроцессоров ограничены аппаратными ресурсами кристалла и корпуса.

Для получения многокристального микропроцессора необходимо провести разбиение его логической структуры на функционально законченные части и реализовать их в виде БИС (СБИС). Функциональная законченность БИС многокристального микропроцессора означает, что его части выполняют заранее определенные функции и могут работать автономно.

На рис. 1.1,*а* показано функциональное разбиение структуры процессора при создании трехкристального микропроцессора (пунктирные линии), содержащего БИС операционного (ОП), БИС управляющего (УП) и БИС интерфейсного (ИП) процессоров.

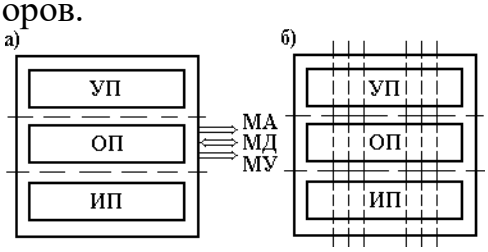


Рис. 1.1. Функциональная структура процессора (а) и ее разбиение для реализации процессора в виде комплекта секционных БИС.

Операционный процессор служит для обработки данных, управляющий процессор выполняет функции выборки, декодирования и вычисления адресов операндов и также генерирует последовательности микрокоманд. Автономность работы и большое быстродействие УП позволяет выбирать команды из памяти с большей скоростью, чем скорость их исполнения ОП. При этом в УП образуется очередь еще не исполненных команд, а также заранее подготавливаются те данные, которые потребуются ОП в следующих циклах работы. Такая опережающая выборка команд экономит время ОП на ожидание операндов, необходимых для выполнения команд программ. Интерфейсный процессор позволяет подключить память и периферийные средства к микропроцессору; он, по существу, является сложным контроллером для устройств ввода/вывода информации. Интерфейсный процессор выполняет также функции канала прямого доступа к памяти.

Выбираемые из памяти команды распознаются и выполняются каждой частью микропроцессора автономно и поэтому может быть обеспечен режим

одновременной работы всех интегральных схем МП, т. е. конвейерный поточный режим исполнения последовательности команд программы (выполнение последовательности с небольшим временным сдвигом). Такой режим работы значительно повышает производительность микропроцессора.

Многокристальные секционные микропроцессоры получаются в том случае, когда в виде БИС реализуются части (секции) логической структуры процессора при функциональном разбиении ее вертикальными плоскостями (рис. 1,б). Для построения многоразрядных микропроцессоров при параллельном включении секций БИС в них добавляются средства "стыковки".

Для создания высокопроизводительных многоразрядных микропроцессоров требуется столь много аппаратных средств, не реализуемых в доступных БИС, что может возникнуть необходимость еще и в функциональном разбиении структуры микропроцессора горизонтальными плоскостями. В результате рассмотренного функционального разделения структуры микропроцессора на функционально и конструктивно законченные части создаются условия реализации каждой из них в виде БИС. Все они образуют комплект секционных интегральных схем МП.

Таким образом, микропроцессорная секция – это БИС, предназначенная для обработки нескольких разрядов данных или выполнения определенных управляющих операций. Секционность интегральных схем МП определяет возможность "наращивания" разрядности обрабатываемых данных или усложнения устройств управления микропроцессора при "параллельном" включении большего числа БИС.

Однокристальные и трехкристальные интегральных схем МП, как правило, изготовляют на основе микроэлектронных технологий униполярных полупроводниковых приборов, а многокристальные секционные БИС МП на основе технологии биполярных полупроводниковых приборов. Использование многокристальных микропроцессорных высокоскоростных биполярных БИС, имеющих функциональную законченность при малой физической разрядности обрабатываемых данных и монтируемых в корпус с большим числом выводов, позволяет организовать разветвление связи в процессоре, а также осуществить конвейерные принципы обработки информации для повышения его производительности.

2. По назначению различают *универсальные* и *специализированные* микропроцессоры.

Универсальные микропроцессоры могут быть применены для решения широкого круга разнообразных задач. При этом их эффективная производительность слабо зависит от проблемной специфики решаемых задач. Специализация МП, т. е. его проблемная ориентация на ускоренное выполнение определенных функций позволяет резко увеличить эффективную производительность при решении только определенных задач.

Среди специализированных микропроцессоров можно выделить различные микроконтроллеры, ориентированные на выполнение сложных последовательностей логических операций, математические МП, предназначенные для повышения производительности при выполнении арифметических операций за счет, например, матричных методов их выполнения, МП для обработки данных в различных областях применений и т. д. С помощью специализированных МП можно эффективно решать новые сложные задачи параллельной обработки данных. Например, конволюция позволяет осуществить более сложную математическую обработку сигналов, чем широко используемые методы

корреляции. Последние в основном сводятся к сравнению всего двух серий данных: входных, передаваемых формой сигнала, и фиксированных опорных, а также к определению их подобия. Конволюция дает возможность в реальном масштабе времени находить соответствие для сигналов изменяющейся формы путем сравнения их с различными эталонными сигналами, что, например, может позволить эффективно выделить полезный сигнал на фоне шума. Разработанные однокристальные конвольверы используются в устройствах опознавания образов в тех случаях, когда возможности сбора данных превосходят способности системы обрабатывать эти данные.

3. *По виду обрабатываемых входных сигналов* различают *цифровые* и *аналоговые* микропроцессоры.

Сами микропроцессоры – цифровые устройства, однако, могут иметь встроенные аналого-цифровые и цифроаналоговые преобразователи. Поэтому входные аналоговые сигналы передаются в МП через преобразователь в цифровой форме, обрабатываются и после обратного преобразования в аналоговую форму поступают на выход. С архитектурной точки зрения такие микропроцессоры представляют собой аналоговые функциональные преобразователи сигналов и называются аналоговыми микропроцессорами. Они выполняют функции любой аналоговой схемы (например, производят генерацию колебаний, модуляцию, смещение, фильтрацию, кодирование и декодирование сигналов в реальном масштабе времени и т. д., заменяя сложные схемы, состоящие из операционных усилителей, катушек индуктивности, конденсаторов и т. д.). При этом применение аналогового микропроцессора значительно повышает точность обработки аналоговых сигналов и их воспроизводимость, а также расширяет функциональные возможности за счет программной "настройки" цифровой части микропроцессора на различные алгоритмы обработки сигналов.

Обычно в составе однокристальных аналоговых МП имеется несколько каналов аналого-цифрового и цифроаналогового преобразования. В аналоговом микропроцессоре разрядность обрабатываемых данных достигает 24 бит и более, большое значение уделяется увеличению скорости выполнения арифметических операций.

Отличительная черта аналоговых микропроцессоров – способность к переработке большого объема числовых данных, т. е. к выполнению операций сложения и умножения с большой скоростью при необходимости даже за счет отказа от операций прерываний и переходов. Аналоговый сигнал, преобразованный в цифровую форму, обрабатывается в реальном масштабе времени и передается на выход обычно в аналоговой форме через цифроаналоговый преобразователь. При этом, согласно теореме Котельникова, частота квантования аналогового сиг- нала должна вдвое превышать верхнюю частоту сигнала.

Сравнение цифровых микропроцессоров производится сопоставлением времени выполнения ими списков операций. Сравнение же аналоговых микропроцессоров производится по количеству эквивалентных звеньев аналого-цифровых фильтров рекурсивных фильтров второго порядка. Производительность аналогового микропроцессора определяется его способностью быстро выполнять операции умножения: чем быстрее осуществляется умножение, тем больше эквивалентное количество звеньев фильтра в аналоговом преобразователе и тем более сложный алгоритм преобразования цифровых сигналов можно задавать в микропроцессоре.

Одним из направлений дальнейшего совершенствования аналоговых микропроцессоров является повышение их универсальности и гибкости. Поэтому вместе с повышением скорости обработки большого объема цифровых данных будут развиваться средства обеспечения развитых вычислительных процессов обработки цифровой информации за счет реализации аппаратных блоков прерывания программ и программных переходов.

4. *По характеру временной организации работы* микропроцессоры делят на *синхронные* и *асинхронные*.

Синхронные микропроцессоры – микропроцессоры, в которых начало и конец выполнения операций задаются устройством управления (время выполнения операций в этом случае не зависит от вида выполняемых команд и величин операндов).

Асинхронные микропроцессоры позволяют начало выполнения каждой следующей операции определить по сигналу фактического окончания выполнения предыдущей операции. Для более эффективного использования каждого устройства микропроцессорной системы в состав асинхронно работающих устройств вводят электронные цепи, обеспечивающие автономное функционирование устройств. Закончив работу над какой-либо операцией, устройство вырабатывает сигнал запроса, означающий его готовность к выполнению следующей операции. При этом роль естественного распределителя работ принимает на себя память, которая в соответствии с заранее установленным приоритетом выполняет запросы остальных устройств по обеспечению их командной информацией и данными.

5. *По организации структуры микропроцессорных систем* различают *одно-* и *многомагистральные* микроЭВМ.

В одномагистральных микроЭВМ все устройства имеют одинаковый интерфейс и подключены к единой информационной магистрали, по которой передаются коды данных, адресов и управляющих сигналов.

В многомагистральных микроЭВМ устройства группами подключаются к своей информационной магистрали. Это позволяет осуществить одновременную передачу информационных сигналов по нескольким (или всем) магистралям. Такая организация систем усложняет их конструкцию, однако увеличивает производительность.

6. *По количеству выполняемых программ* различают *одно-* и *многопрограммные* микропроцессоры.

В однопрограммных микропроцессорах выполняется только одна программа. Переход к выполнению другой программы происходит после завершения текущей программы.

В много-или мультипрограммных микропроцессорах одновременно выполняется несколько (обычно несколько десятков) программ. Организация мультипрограммной работы микропроцессорных управляющих систем позволяет осуществить контроль за состоянием и управлением большого числа источников или приемников информации.

Тема 2. Архітектура та типи архітектур мікропроцесорних систем
Обобщённая архитектура микропроцессора

С точки зрения пользователя (разработчика автоматических систем) при выборе МП для решения конкретной задачи целесообразно располагать некоторыми обобщенными или комплексными характеристиками возможностей МП, т. е. воспринимать его как нечто цельное, имеющее вполне определенные потребительские качества (свойства и характеристики). В конечном итоге разработчик нуждается в уяснении и понимании лишь тех компонентов МП и МПС, которые явно отражаются в программах и (или) должны быть учтены при разработке и выполнении программ: число и имена программно-доступных регистров; разрядность машинного слова; система команд; доступный размер и адреса ОЗУ; быстродействие МП; схему обработки прерываний; способы адресации ОЗУ и внешних устройств. Совокупность таких сведений представляет определенную модель МП (МПС) с точки зрения пользователя (разработчика МПАС).

Указанные выше характеристики и свойства определяются понятием архитектуры МП (МПС, МЭВМ).

Архитектура МП – это его логическая организация, рассматриваемая с точки зрения пользователя; она определяет возможности МП по аппаратной, программной и микропрограммной реализации функций, необходимых для построения МПС и МПАС.

Понятие архитектуры МП отражает:

- структуру, т. е. совокупность компонентов, составляющих МП, и связей между ними;
- способы представления и форматы данных;
- способы обращения ко всем доступным для пользователя (программнодоступным) элементам структуры (адресация к регистрам, ячейкам оперативной и постоянной памяти, внешним устройствам);
- набор операций, выполняемых МП, т. е. система команд МП;
- характеристики управляющих слов и сигналов, вырабатываемых микропроцессором и поступающих в МП извне;
- реакцию на внешние сигналы (схема обработки прерываний и т. д.) и другие характеристики.

Микропроцессор характеризуется:

1) тактовой частотой, определяющей максимальное время выполнения переключения элементов в ЭВМ;

2) разрядностью, т. е. максимальным числом одновременно обрабатываемых двоичных разрядов;

3) архитектурой.

Разрядность МП обозначается m/n/k/ и включает:

m – разрядность внутренних регистров, определяет принадлежность к тому или иному классу процессоров;

n – разрядность шины данных, определяет скорость передачи информации;

k – разрядность шины адреса, определяет размер адресного пространства.

Например, МП i8088 характеризуется значениями m/n/k = 16/8/20;

Понятие архитектуры микропроцессора включает в себя систему команд и способы адресации, возможность совмещения выполнения команд во времени, наличие дополнительных устройств в составе микропроцессора, принципы и режимы его работы. Выделяют понятия микроархитектуры и макроархитектуры.

Микроархитектура микропроцессора – это аппаратная организация и

логическая структура микропроцессора, регистры, управляющие схемы, арифметико-логические устройства, запоминающие устройства и связывающие их информационные магистрали.

Макроархитектура – это система команд, типы обрабатываемых данных, режимы адресации и принципы работы микропроцессора.

В общем случае под архитектурой ЭВМ понимается абстрактное представление машины в терминах основных функциональных модулей, языка ЭВМ, структуры данных.

Типы архитектур микропроцессорых систем.

Все микропроцессоры можно разделить на следующие группы:

- МП с гарвардской архитектурой;
- МП с фоннеймановской архитектурой;
- МП типа CISC (Complex Instruction Set Computing) с полным набором команд;
- МП типа RISC (Reduced Instruction Set Computing) с сокращенным набором команд;
- МП типа MISC (Minimum Instruction Set Computing) с минимальным набором команд и весьма высоким быстродействием (в настоящее время эти модели находятся в стадии разработки).

Архитектуры микропроцессоров различаются по использованию памяти. Наибольшее распространение получили:

- гарвардская архитектура;
- архитектура фон Неймана.

Гарвардская архитектура предполагает раздельное использование памяти программ и данных. Обычно такую архитектуру используют для повышения быстродействия системы за счёт разделения путей доступа к памяти программ и данных. Большинство специализированных микропроцессоров (особенно микроконтроллеры) имеют данную архитектуру.

Антипод гарвардской – архитектура фон Неймана – предполагает хранение программ и данных в общей памяти и наиболее характерна для микропроцессоров, ориентированных на использование в компьютерах. Примером могут служить микропроцессоры семейства x86.

Термин *CISC* означает сложную систему команд и является аббревиатурой английского определения Complex Instruction Set Computer.

Большинство современных ПК типа IBM PC (International Business Machine) используют МП типа CISC.

Отметим некоторые характеристики МП:

- начиная с МП 80386 используется конвейерное выполнение команд – одновременное выполнение разных тактов последовательных команд в разных частях МП при непосредственной передаче результатов из одной части МП в другую. Конвейерное выполнение команд увеличивает эффективное быстродействие ПК в 2–3 раза;
- начиная с МП 80286 предусматривается возможность работы в вычислительной сети;
- начиная с МП 80286 имеется возможность многозадачной работы (многопрограммность) и сопутствующая ей защита памяти;
- начиная с МП 80386 обеспечивается поддержка режима системы

виртуальных машин, т. е. такого режима многозадачной работы, при котором в одном МП моделируется как бы несколько компьютеров, работающих параллельно и имеющих разные операционные системы;

- начиная с МП 80286 микропроцессоры могут работать в двух режимах: реальном (Real mode) и защищенном (Protected mode). В реальном режиме имитируется (эмулируется) работа МП 8086, естест- венно, однозадачная. В защищенном режиме возможна многозадачная работа с непосредственным доступом к расширенной памяти (см. под- разд. 4.5) и с защитой памяти, отведенной задачам, от посторонних об- ращений.

Микропроцессоры *80586* (P5) более известны по их товарной марке Pentium, которая запатентована фирмой Intel (МП 80586 других фирм имеют иные обозначения: К5 у фирмы AMD, М 1 у фирмы Cyrix и др.).

Эти микропроцессоры имеют пятиступенную конвейерную структуру, обеспечивающую многократное совмещение тактов выполнения последовательных команд, и КЭШ-буфер для команд условной передачи управления, позволяющий предсказывать направление ветвления программ; по эффективному быстродействию они приближаются к RISC МП, выполняющим каждую команду как бы за один такт. Pentium имеют 32-разрядную адресную шину и 64-разрядную шину данных. Обмен данными с системой может выполняться со скоростью 1 Гб/с.

У всех МП Pentium имеется встроенная КЭШ-память, отдельно для команд, отдельно для данных; имеются специализированные конвейерные аппаратные блоки сложения, умножения и деления, значительно ускоряющие выполнение операций с плавающей запятой.

Микропроцессоры Pentium Pro. В сентябре 1995 г. прошли презентацию и выпущены МП 80686 (P6), торговая марка Pentium Pro. Благо- даря новым схемотехническим решениям они обеспечивают для ПК более высокую производительность. Часть этих новшеств может быть объединена понятием динамическое исполнение (dynamic execution), что в первую очередь означает наличие 14-ступенной суперконвейерной структуры (superpipelining), предсказания ветвлений программы при условных передачах управления (branch prediction) и исполнение команд по предполагаемому пути ветвления (speculative execution).

КЭШ-память емкостью 256–512 кб – обязательный атрибут высокопроизводительных систем на процессорах Pentium. Однако у них встроенная КЭШ-память имеет небольшую емкость (16 кб), а основная ее часть находится вне процессора на материнской плате. Поэтому обмен данными с ней происходит не на внутренней частоте МП, а на частоте тактового генератора, которая обычно в 2–3 раза ниже, что снижает общее быстродействие компьютера. В МП Pentium Pro КЭШ-память емкостью 256–512 кб находится в самом микропроцессоре.

Микропроцессоры OverDrive. Интерес представляют также недавно разработанные МП OverDrive, по существу являющиеся своеобразными сопроцессорами, обеспечивающими для МП 80486 режимы ра- боты и эффективное быстродействие, характерные для МП Pentium. Появились МП OverDrive, улучшающие характеристики и микропро- цессоров Pentium.

Термин *RISC* означает сокращённую систему команд и происходит от английского Reduced Instruction Set Computer.

Микропроцессоры типа RISC содержат набор только простых, чаше всего встречающихся в программах команд. При необходимости выполнения более сложных команд в микропроцессоре производится их автоматическая сборка из

простых. В этих МП на выполнение каждой простой команды за счет их наложения и параллельного выполнения тратится 1 машинный такт (на выполнение даже самой короткой команды из системы CISC обычно тратится 4 такта).

Некоторые микропроцессоры типа RISC: ARM (на его основе выпускались ПК IBM PC RT) – один из первых 32-разрядных RISC микропроцессоров, имеющий 118 различных команд. Современные RISC микропроцессоры (80860, 80960, 80870, Power PC) являются 64- разрядными при быстродействии до 150 млн оп./с. Микропроцессоры Power PC (Performance Optimized With Enhanced RISC PC) весьма перспективны и уже сейчас широко применяются в машинах-серверах и в ПК типа Macintosh.

Микропроцессоры типа RISC имеют очень высокое быстродействие, но программно не совместимы с CISC-процессорами: при выполнении программ, разработанных для ПК типа IBM PC, они могут лишь эмулировать (моделировать, имитировать) МП типа CISC на программном уровне, что приводит к резкому уменьшению их эффективной производительности.

Однако, несмотря на широкую распространённость этих понятий, необходимо признать, что сами названия не отражают главного различия между системами команд CISC и RISC. *Основная идея RISC- архитектуры* – эго тщательный подбор таких комбинаций кодов операций, которые можно было бы выполнить за один такт тактового генератора. Основной выигрыш от такого подхода – резкое упрощение аппаратной реализации ЦП и возможность значительно повысить его производительность. Все команды работают с операндами и имеют одинаковый формат. Обращение к памяти выполняется с помощью специальных команд загрузки регистра и записи. Простота структуры и небольшой набор команд позволяет реализовать полностью их аппаратное выполнение и эффективный конвейер при небольшом объёме оборудования. Арифметику RISC-процессоров отличает высокая степень дробления конвейера. Этот прием позволяет увеличить тактовую частоту (значит и производительность) компьютера. Чем более элементарные действия выполняются в каждой фазе работы конвейера, тем выше частота его работы. RISC-процессоры с самого начала ориентированны на реализацию всех возможностей ускорения арифметических операций, поэтому их конвейеры обладают значительно более высоким быстродействием, чем в CISC-процессорах. Поэтому RISC-процессоры в 2–4 раза быстрее имеющих ту же тактовую частоту CISC- процессоров с обычной системой команд и высокопроизводительней, несмотря на больший объём программ, на (30 %). Дейв Паттерсон и Карло Секуин сформулировали 4 основных принципа RISC:

1. Любая операция должна выполняться за один такт, вне зависимости от ее типа.
2. Система команд должна содержать минимальное количество наиболее часто используемых простейших инструкций одинаковой длины.
3. Операции обработки данных реализуются только в формате “регистр–регистр“ (операнды выбираются из оперативных регистров процессора, и результат операции записывается также в регистр; а обмен между оперативными регистрами и памятью выполняется только с помощью команд чтения/записи).
4. Состав системы команд должен быть “ удобен “ для компиляции операторов языков высокого уровня.

Микропроцессоры с архитектурой CISC (Complex Instruction Set Computers) – архитектура вычислений с полной системой команд. Реализующие на уровне машинного языка комплексные наборы команд различной сложности (от простых,

характерных для микропроцессора первого поколения, до команд значительной сложности, характерных для современных 32-разрядных микропроцессоров типа 80486, 68040 и др.).

CISC-процессоры выполняют большой набор команд с развитыми возможностями адресации, давая разработчику возможность выбрать наиболее подходящую команду для выполнения необходимой операции. Процессор с CISC-архитектурой может иметь однобайтовый, двухбайтовый и трехбайтовый (редко четырехбайтовый) формат команд. При этом система команд, как правило, не ортогональна, то есть не все команды могут использовать любой из способов адресации применительно к любому из регистров процессора. Выборка команды на исполнение осуществляется побайтно в течение нескольких циклов работы процессора.

В связи с успехами микроэлектроники появилась возможность построения RISC-компьютеров (Reduced Instruction Set Computing), т. е. микропроцессорных систем с сокращенным набором команд.

Поэтому в RISC архитектуре основу системы команд составляют наиболее употребительные, «короткие» операции типа алгебраического сложения. Сложные операции выполняются как подпрограммы, состоящие из простых операций. Это позволяет значительно упростить внутреннюю структуру процессора, уменьшить фазы дробления конвейерной обработки и увеличить частоту работы конвейера. Но здесь необходимо отметить, что за эффект приходится расплачиваться усложнением процедур обмена данными между регистрами, кэшпамятью и оперативной памятью.

В процессорах с RISC-архитектурой набор исполняемых команд сокращен до минимума. Для реализации более сложных операций приходится комбинировать команды. При этом все команды имеют формат фиксированной длины (например, 12, 14 или 16 бит), выборка команды из памяти и ее исполнение осуществляется за один цикл (такт) синхронизации. Система команд RISC-процессора предполагает возможность равноправного использования всех регистров процессора. Это обеспечивает дополнительную гибкость при выполнении ряда операций.

Вопрос о производительности процессоров с архитектурой RISC и CISC сложен и неоднозначен. Во-первых, оценка производительности по времени выполнения команд различных систем (RISC и CISC) не совсем корректна. Обычно производительность МП и МК принято оценивать числом операций пересылки «регистр-регистр», которые могут быть выполнены в течение одной секунды. В системах с CISC-процессором время выполнения операции «регистр-регистр» составляет от 1 до 3 циклов, что, казалось бы, уступает производительности систем с RISC-процессором. Однако стремление к сокращению формата команд при сохранении ортогональности системы команд RISC-процессора приводит к вынужденному ограничению числа доступных в одной команде регистров. Так, например, системой команд МК PIC16 предусмотрена возможность пересылки результата операции только в один из двух регистров — регистр-источник операнда f или рабочий регистр W. Таким образом, операция пересылки содержимого одного из доступных регистров в другой (не источник операнда и не рабочий) потребует использования двух команд. Такая необходимость часто возникает при пересылке содержимого одного из регистров общего назначения (РОН) в один из портов МК. В то же время, в системе команд большинства CISC-процессоров присутствуют команды пересылки содержимого РОН в один из

портов ввода/вывода. То есть более сложная система команд иногда позволяет реализовать более эффективный способ выполнения операции.

Во-вторых, оценка производительности по скорости пересылки «регистр-регистр» не учитывает особенностей конкретного реализуемого алгоритма управления. Так, при разработке быстродействующих устройств автоматизированного управления основное внимание следует уделять времени выполнения операций умножения и деления при реализации уравнений различных передаточных функций. А при реализации пульта дистанционного управления бытовой техникой следует оценивать время выполнения логических функций, которые используются при опросе клавиатуры и генерации последовательной кодовой посылки управления. Поэтому в критических ситуациях, требующих высокого быстродействия, следует оценивать производительность на множестве тех операций, которые преимущественно используются в алгоритме управления и имеют ограничения по времени выполнения.

В-третьих, необходимо еще учитывать, что указанные в справочных данных частоты синхронизации обычно соответствуют или пропорциональны частоте кварцевого резонатора, в то время как длительность цикла центрального процессора определяется частотой обмена по системной магистрали. Соотношение этих частот индивидуально для каждого устройства должно быть принято в расчет при сравнении производительности различных вычислительных систем.

Другой подход к архитектуре вычислительных систем с точки зрения использования памяти в процессорах реализует один из двух принципов построения:

- архитектура с общей памятью для размещения команд и данных;
- архитектура с разделенной памятью команд и данных.

Для универсальных процессоров типичным является использование архитектуры с общей, единой шиной для данных и команд (одношинная, или принстонская, фон-неймановская архитектура). Соответственно, в составе системы в этом случае используется одна общая память, как для данных, так и для команд,рис. 2.1.

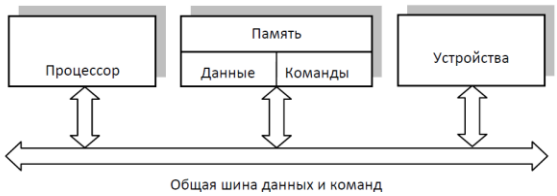


Рис. 2.1. Архитектура с общей шиной данных и команд

Другой тип архитектуры микропроцессорной системы — это архитектура с раздельными шинами данных и команд (двухшинная, или гарвардская, архитектура). Эта архитектура предполагает наличие в системе отдельной памяти для данных и отдельной памяти для команд (рис. 2.2). Обмен процессора с каждым из двух типов памяти происходит по своей шине.

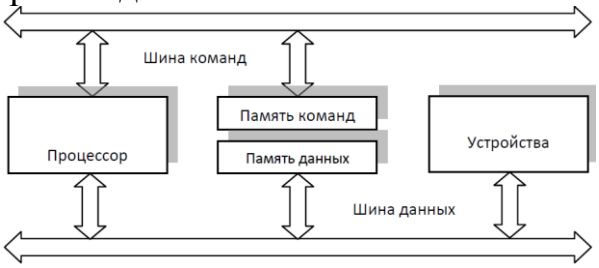


Рис. 2.2. Архитектура с раздельными шинами данных и команд

Архитектура с общей шиной распространена гораздо больше, она применяется, например, в персональных компьютерах и в сложных микрокомпьютерах. Архитектура с раздельными шинами применяется в основном в однокристальных микроконтроллерах.

Рассмотрим некоторые достоинства и недостатки обоих архитектурных решений.

Архитектура с общей шиной (принстонская, фон-неймановская) проще, она не требует от процессора одновременного обслуживания двух шин, контроля обмена по двум шинам сразу. Наличие единой памяти данных и команд позволяет гибко распределять ее объем между кодами данных и команд. Например, в некоторых случаях нужна большая и сложная программа, а данных в памяти надо хранить не слишком много. В других случаях, наоборот, программа требуется простая, но необходимы большие объемы хранимых данных. Перераспределение памяти производится оперативно, единственное условия — чтобы команды и данные в сумме помещались в памяти системы. Как правило, в системах с такой архитектурой память бывает довольно большого объема (до десятков гигабайт). Это позволяет решать самые сложные задачи.

Архитектура с раздельными шинами данных и команд сложнее, она заставляет процессор работать одновременно с двумя потоками кодов, обслуживать обмен по двум шинам одновременно. Программа может размещаться только в памяти команд, данные — только в памяти данных. Такая узкая специализация ограничивает круг задач, решаемых системой, так как не дает возможности гибкого перераспределения памяти. Память данных и память команд в этом случае имеют разный объем (память команд значительно больше памяти данных), поэтому применение систем с данной архитектурой ограничивается обычно не слишком сложными задачами.

Преимущество гарвардской архитектуры в первую очередь, в быстродействии. Дело в том, что при единственной шине команд и данных процессор вынужден по одной этой шине принимать данные (из памяти или устройства ввода/вывода) и передавать данные (в память или в устройство ввода/вывода), а также читать команды из памяти. Естественно, одновременно эти пересылки кодов по магистрали происходить не могут, они должны производиться по очереди. Современные процессоры способны совместить во времени выполнение команд и проведение циклов обмена по системной шине. Использование конвейерных технологий и быстрой кэш-памяти позволяет им ускорить процесс взаимодействия со сравнительно медленной системной памятью. Повышение тактовой частоты и совершенствование структуры процессоров дают возможность сократить время выполнения команд. Но дальнейшее увеличение быстродействия системы возможно только при совмещении пересылки данных и чтения команд, то есть при переходе к архитектуре с двумя шинами.

В случае двухшинной архитектуры обмен по обеим шинам может быть независимым, параллельным во времени. Соответственно, структуры шин (количество разрядов кода адреса и кода данных, порядок и скорость обмена информацией и т.д.) могут быть выбраны оптимально для той задачи, которая решается каждой шиной. Поэтому при прочих равных условиях переход на двухшинную архитектуру ускоряет работу микропроцессорной системы, хотя и требует дополнительных затрат на аппаратуру, усложнения структуры процессора. Память данных в этом случае имеет свое распределение адресов, а память команд — свое.

Проще всего преимущества двухшинной архитектуры реализуются внутри одной микросхемы. В этом случае можно также существенно уменьшить влияние недостатков этой архитектуры. Поэтому основное ее применение — в микроконтроллерах, от которых не требуется решения слишком сложных задач, но зато необходимо максимальное быстродействие при заданной тактовой частоте.

Таким образом, сложившаяся практика проектирования микропроцессорных систем показывает, что для решения задач управления на верхнем уровне используются универсальные микропроцессоры с принстонской, CISC архитектурой, а на нижнем – микроконтроллеры с гарвардской, и предпочтительно RISC архитектурой.

Тема 3. Архітектура 8-розрядного мікропроцесора.

Типичным представителем 8-битных однокристальных микропроцессоров является i8080, разработанный фирмой Intel в 1974 г. А также его советский аналог К580ИК80. Кристалл процессора производится по технологическим нормам 6 мкм, вмещает 6000 транзисторов и имеет тактовую частоту 2 МГц, а более поздний его вариант i8080A (КР580ВМ80А) – 2,5 МГц. Процессор снабжен 8- разрядной шиной данных, 16-разрядной шиной адреса, с помощью которой адресует 2^{16} = 64 Кбайт памяти, 256 устройств ввода и 256 устройств вывода. МП работает от трех источников питания – +5, +12 и –5 В и рассеивает мощность 1,25 Вт. Длительность такта при частоте 2 МГц составляет 0,5 мкс, при этом быстродействие – 500 000 коротких операций (регистр-регистр) в секунду.

Зная конструкцию и особенности работы этих устройств, можно изучить любой, даже самый сложный современный процессор.

В структурной схеме МП КР580, рис.3.1, можно выделить три основных составляющих: арифметическо-логический блок, блок регистров и устройство управления.

Арифметическо-логический блок, рис.3.2, представляет собой:

-арифметическо- логическое устройство (АЛУ);

-регистр временного хранения;

- аккумулятор.

АЛУ - осуществляет операции по арифметической и логической обработке данных, операции сдвига. АЛУ состоит из сумматора, регистра сдвига, регистра состояния.

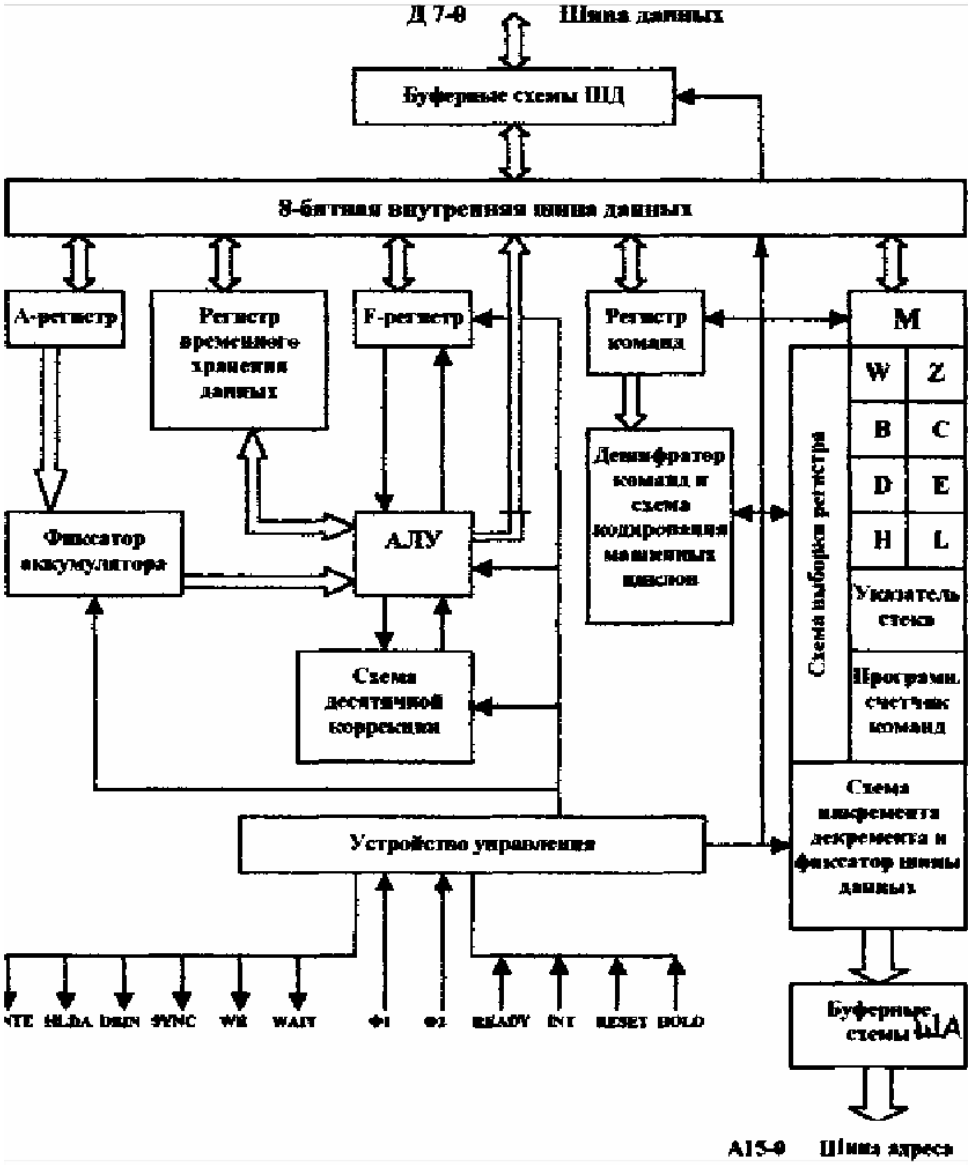


Рисунок 3.1 – Структурная схема МП

Арифметическими операциями являются: сложение, вычитание, сравнение, преобразование, увеличение, уменьшение числа на 1. Логические операции - их 16. Основные - И, ИЛИ, НЕ и их комбинации.

Более сложные операции (умножение, деление, вычисление элементарных функций и др.) выполняются по подпрограммам.

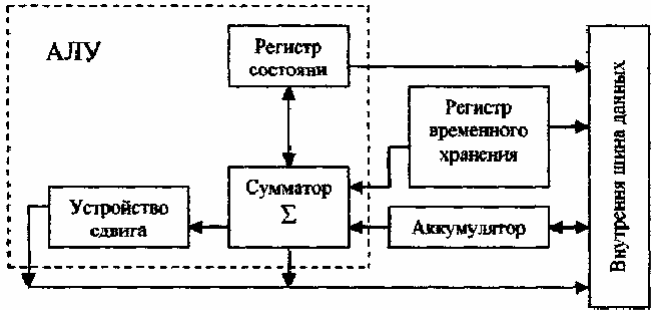


Рисунок 3.2- Арифметический логический блок

При сложении двух чисел одно из них должно находиться в аккумуляторе, который является регистром. Другое слагаемое находиться в регистре временного хранения. Результат вычисления заносится в аккумулятор поверх одного из

слагаемых. Например, рассмотрим сложение двух чисел 00001010 + 00000101. После операции сложения получим число 00001111, которое запишется в аккумулятор поверх числа 00001010 (рис.3.3).

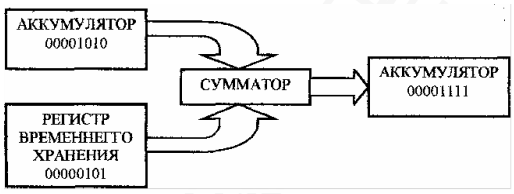


Рис. 3.3- Сложение чисел в АЛУ

Аккумулятор (А) используется в качестве источника одного из операндов и места, где фиксируется результат операции, В команде аккумулятор А в явном виде не адресуется. На использование А в операции указывает код операции команды. Можно сказать, в отношении А применяется подразумеваемая адресация, что позволяет применять одноадресные команды, имеющие короткий формат. Чтобы А мог выполнять функции регистра операнда и регистра результата операции, он строится на основе двухступенчатого триггера. Использование аккумулятора А и регистров общего назначения РОН позволяет при выполнении команд уменьшить количество обращений к памяти и тем самым повысить быстродействие МП.

Блок регистров можно разбить на следующие группы по их функциональному назначению:

-регистры общего назначения (РОН);
-регистр признаков F(флагов);
-регистры временного хранения W, Z, T;
-счетчик команд;
-указатель стека;
-регистр команд.

Регистры B, C, D, E, H, L образуют группу регистров общего назначения (РОН) (рис.3.4).

Особенностью РОН является их доступность для пользователя. С помощью определённой команды можно поместить данные в любой регистр РОН или извлечь их из регистров, кроме того, переслать из одного регистра в другой.

Регистры РОН - 8-разрядные, но они допускают попарную работу, поэтому позволяют вводить и запоминать 16- разрядные числа (BC, DE, HL). При этом из пары регистров выделяют старший и младший разряды. РОН выполняют следующие функции:

-временно хранят данные, подлежащие обработке;
-осуществляют промежуточные вычисления;
-осуществляют счет числа циклов работы МП;
-хранят указатели адресов ячеек памяти двоичных чисел.

B	**C**
D	**E**
H	**L**
Старший разряд	Младший разряд

Рисунок 3.4- Регистры общего назначения

Процесс выполнения программы можно поставить в зависимость от значения результата выполнения предыдущей операции. Для обращения к информации о результатах вычислений в состав МП введен регистр признаков с набором триггеров, которые устанавливаются в «1» или сбрасываются в «О» в зависимости от результата произведенных вычислений.

Каждый из триггеров хранит какой-то один бит условия, а в совокупности эти биты образуют регистр бит условий (рис.3.5). Программа может проверить значения четырех бит условий: переноса (CY), знака (S),

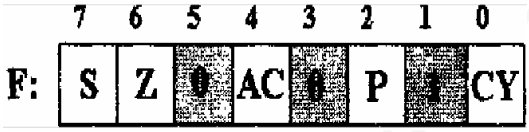


Рисунок 3.5- Регистр признаков (флагов)

нуля (Z), четности (P), используя одну из команд условного перехода, условного вызова подпрограммы или условного возврата из подпрограммы. Пятый бит - бит вспомогательного переноса (AC).

Бит переноса обычно используется для индикации переполнения при сложении или заема при вычитании, возникающих в седьмом разряде аккумулятора. Кроме того, команды, сдвигающие содержимое аккумулятора влево или вправо на один разряд, используют бит переноса как девятый бит аккумулятора. Например, при сложении двух однобайтных чисел AE_{16} и 74_{16}, возникает перенос «1» из седьмого разряда аккумулятора в бит переноса:

```
   10101110=AE
+  01110100=74
-------------
CY= 1 00100010
```

Если при выполнении операции сложения осуществляется перенос из седьмого разряда аккумулятора, то бит переноса устанавливается в «1», если переноса нет, то бит переноса устанавливается в «0».

Бит знака. Седьмой разряд результата выполнения операции в аккумуляторе может быть интерпретирован как знак результата. Команды, использующие бит знака, устанавливают его эквивалентным значению седьмого разряда аккумулятора. Ноль в седьмом разряде говорит о положительном значении результата, а единица — об отрицательном. Это значение дублируется в бит знака регистра бит условий и используется в командах условных переходов для реализации ветвления в зависимости от знака полученного результата.

Бит нуля. Ряд команд МП устанавливают бит нуля в «1», что соответствует равенству нулю всех бит аккумулятора. Если результат в аккумуляторе отличен от нуля, бит нуля будет сброшен в «0». Например, в результате сложения двух чисел $A7_{16}$ и 59_{16} будут установлены в «1» биты переноса и нуля:

```
   10100111=A7
+  01011001=59
-------------
CY=1 00000000=0
 Z=1
```

Бит четности. Четность результата выполнения команды определяется подсчетом количества единиц в аккумуляторе после выполнения команды. Бит четности устанавливается в «1» для четного количества единиц и сбрасывается в «0» для нечетного их количества. Например, если результат вычислений в

аккумуляторе представляет собой значение 01001110, то происходит установка в «1» бита четности в регистре бит условий. Обычно признак четности используют для контроля на четность данных в процессе их передачи.

Бит вспомогательного переноса указывает на наличие переноса из третьего разряда результата в аккумуляторе. Этот бит нельзя опросить в программе непосредственно и он используется лишь в команде DAA (десятичная настройка аккумулятора) для преобразования чисел из двоичной системы счисления в двоично-десятичную. Покажем, например, как бит дополнительного переноса и команда DAA позволяют выполнить сложение кодов 19_{10} и 8_{10} с получением двоично- десятичного результата. Значение 00011001_2 будет эквивалентно 19_{10}. Сложение 19_{10} с 8_{10} даст шестнадцатеричное число 21_{16}, и будет установлен бит вспомогательного переноса:

$$\begin{array}{r} +\ 00011001 = 19_{10} \\ \underline{+\ 00001000 = 8_{10}} \\ \text{AC=1}\ \ 00100001 = 21_{16} \end{array}$$

8-разрядные регистры временного хранения данных T, W, Z недоступны программисту. Используются для запоминания двухбайтовых и трёхбайтовых команд перехода, передаваемых с внутренней ШД в счётчик команд.

Программный счётчик команд - это программно- доступный регистр и состоит из 16 двоичных разрядов и позволяет адресовать память объемом 2^{16} = 65536 бит. Это средство, с помощью которого МП переходит от одной команды к другой при хранении их в памяти.

Счетчик команд содержит адрес байта команды, которая будет следующей считана из памяти (ПЗУ или ОЗУ) для выпол- нения. После считывания очередной команды счетчик команд увеличивает свое значение на 1, 2 или 3 в зависимости от длины команды. Этот процесс длится до тех пор, пока выполнение команд происходит по последовательной ветви команд.

Если текущая команда (перехода или вызова подпрограммы) изменит последовательность выполнения программы, МП занесет в счетчик команд не адрес следующей команды, а адрес команды, выполняемой в настоящий момент. Счетчик команд доступен программно и всегда содержит 16-разрядный адрес. Он может быть сброшен устройством управления, инкрементирован, изменен командой перехода.

Указатель стека представляет собой 16- разрядный регистр и служит для адресации стековой памяти.

Стек—это группа последовательно пронумерованных регистров или ячеек памяти, снабжённых указателем стека, в котором автоматически при записи и считывании устраняется адрес последней занятой ячейки стека (вершина стека).

При записи в стек слово помещается в следующую по порядку свободную ячейку стека, а при считывании из стека извлекается последнее из него слово.

Стековая память является производственной зоной в ОЗУ и осуществляет режим обработки прерываний.

Действия МП при выполнении прерываний следующие:

-обслуживание подпрограммы прерывания;

-запоминание результата незавершённых операций;

-восстановление своёго состояния;

-доведение до конца прерванной операции.

Регистр команд — это 8-разрядный регистр, содержащий первый байт команды, принимает и хранит код очередной команды, адрес которой был установлен в программном счетчике команд.

Устройство управления и синхронизации (УУС) выполняет следующие функции:

-получает сигналы с ШД и определяет природу выполняемой команды;

-передаёт сигналы во все устройства системы для координации выполнения команд;

-вырабатывает серию управляющих сигналов, которые передаются на узлы, блоки и элементы МП для координации их работы и согласования действий с внешними устройствами, например, с памятью.

Всё что происходит в МП, подчинено УУС.

На рис.3.1 показаны 10 управляющих выводов. К ним подключается шина управления, по которой сигналы передаются из МП и в МП для согласования совместных действий с внешними устройствами и изменения режима их работы.

Двунаправленная шина данных предназначена для организации связи между отдельными блоками микропроцессора, для связи с другими микросхемами и микроЭВМ. Она включает в себя внутреннюю шину данных и буфер данных (БД), соединенный с внешней шиной данных Д7– Д0. Двунаправленный с тремя состояниями БД состоит из буферного регистра, формирователей и предназначен для развязки внутренней и внешней шин данных (в процессе ввода или выполнения операций, не связанных с пересылкой данных, БД отключается).

Выбор регистра, участвующего в операции, осуществляется схемой выбора регистра (СВР).

Адресная логика обеспечивает выдачу на адресную шину адресов, команд и реализуется в виде буферных схем, которые включает в себя адресный буфер (АБ), буферный регистр адреса (БРА) и схему инкрементации и декрементации (СИД).

Адресный буфер представляет собой 16 выходных формирователей с тремя состояниями и предназначен для выдачи адреса на выводы адресной шины А15–АО. Третье (отключающее) состояние позволяет подключать микропроцессор непосредственно к общей системной адресной шине микроЭВМ.

Буферный регистр адреса принимает и хранит адрес с любого 16-разрядного регистра. Его выход связан со входами адресного буфера и со входами схемы СИД.

Схема СИД - схема быстрого переноса/заема. С ее помощью содержимое БРА может быть передано с изменением на единицу или без изменения через 16-разрядный мультиплексор на вход любого 16-битового регистра (пары) BC, DE, HL, SP или PC.

Схема десятичной коррекции (СДК) предназначена для преобразования двоичного кода в двоично-десятичный при обработке двоично-десятичных чисел.

Сравнивая структурную схему МП Intel 8080 (рис.3.6) со схемой на рис 3.1, можно сказать, что она аналогична структуре МП К580.

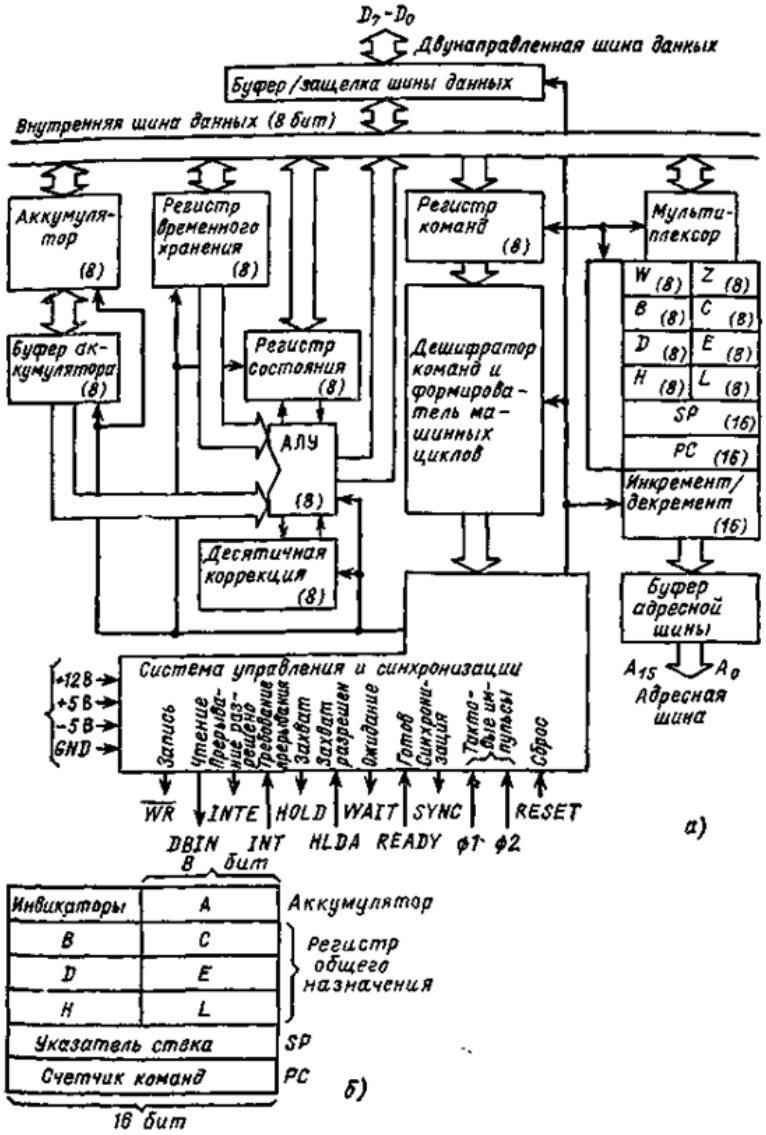


Рисунок 3.6-Структурная схема МП Intel 8080 (а), регистры, доступные программисту (б).

Таким образом, основными особенностями организации МП К580 являются:

-трёхшинная структура с шинами данных,адреса и управления;

-магистральный принцип связей, реализованный в виде связывающей основные узлы МП двунаправленной шины данных;

-наличие регистровой памяти, организованной регистрами общего назначения и специальными регистрами (программный счётчик команд, указатель стека, указатель данных), а также регистрами временного хранения;

-наличие 16-разрядной шины адреса, обеспечивающей возможность прямой адресации любого байта в памяти ёмкостью 64 Кбайт;

-наличие операций над двухбайтными словами;

-использование трёх форматов команд (однобайтного, двухбайтного и трёхбайтного) и разнообразных способов адресации;

-реализация векторного многоуровневого приоритетного прерывания путём использования контроллера прерываний;

-реализация в МП режима прямого доступа к памяти путём подключения контроллера прямого доступа;

-наличие эффективных средств работы с подпрограммами и обработки запросов прерываний (стековая память, специальные команды вызова подпрограмм и возврата из подпрограмм).

Тема 4. Пам’ять мікропроцесорних систем

Классификация запоминающих устройств

Памятью называется совокупность технических средств, предназначенных для записи, хранения и считывания информации в виде цифрового кода. Отдельные элементы памяти получили название запоминающих устройств (ЗУ). Основная память микропроцессорной системы состоит из ЗУ двух видов: – оперативного – ОЗУ (RAM, Random Access Memory) и постоянного – ПЗУ (ROM, Read Only Memory) (рис. 4.1).

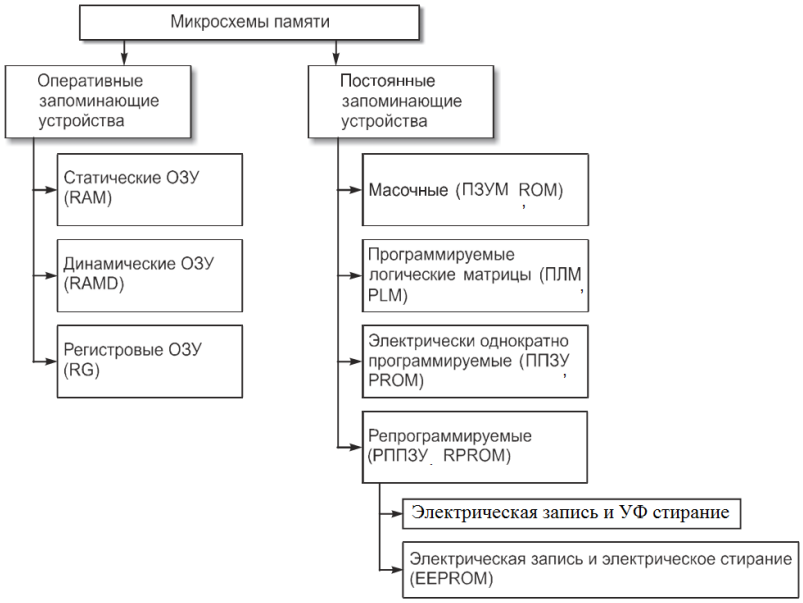


Рис. 4.1. Классификация микросхем памяти

ОЗУ предназначено для хранения переменной информации, оно допускает изменение своего содержимого в ходе выполнения процессором вычислительных операций с данными. Это значит, что процессор может выбрать из ОЗУ код команды и данные (режим считывания) и после обработки поместить в ОЗУ полученный результат (режим записи). Причем возможно размещение в ОЗУ новых данных на месте прежних, которые в этом случае перестают существовать. При этом различают статические и динамические ОЗУ.

В микросхемах *статических ОЗУ* информация хранится в виде устойчивого состояния триггера, который способен при наличии напряжения питания сохранять свое состояние неограниченное время. Достоинством таких ОЗУ является максимальное быстродействие, а недостатком – высокая стоимость и значительное энергопотребление.

В микросхемах *динамических ОЗУ* элементы памяти выполнены на основе конденсаторов, сформированных внутри полупроводникового кристалла. Такие элементы памяти не могут долгое время сохранять свое состояние, определяемое наличием или отсутствием электрического заряда, и поэтому нуждаются в периодическом обновлении (*регенерации*). Микросхемы динамических ОЗУ отличаются от статических гораздо большей информационной емкостью, что обусловлено меньшим числом компонентов в одном элементе памяти и, следовательно, более плотным их размещением в полупроводниковом кристалле. Однако динамические ОЗУ сложнее в применении, поскольку нуждаются в организации принудительной регенерации и в усложнении устройств управления. Динамическая память имеет среднее быстродействие и невысокую стоимость.

Таким образом, ОЗУ может работать в режимах записи, считывания и хранения информации.

ПЗУ содержит информацию, которая не изменяется в ходе выполнения процессором программы и должна храниться при выключенном источнике питания. Такую информацию составляют стандартные подпрограммы, табличные данные, коды физических констант, постоянных коэффициентов и т.п. Эта информация заносится в ПЗУ предварительно, например, путем пережигания легкоплавких металлических перемычек в структуре ПЗУ, и в ходе работы, процессора может только считываться.

Существует разновидность ПЗУ, допускающая неоднократное (сотни тысяч циклов) перепрограммирование (репрограммирование). Элементом памяти в репрограммируемых ПЗУ (РПЗУ, PROM) является МДП-транзистор, обладающий свойством переходить в состояние проводимости под воздействием импульса программирующего напряжения и сохранять это состояние длительное время. Данный эффект обусловлен накоплением электрического заряда в подзатворном диэлектрике. Для стирания информации перед новым циклом программирования необходимо вытеснить накопленный под затвором заряд. В зависимости от способа выполнения этой операции микросхемы РПЗУ разделяют на два вида: со стиранием ультрафиолетовым светом (УФ РПЗУ) и со стиранием электрическим сигналом (ЭС РПЗУ, EEPROM, или *Flash-память*). Флэш-технология позволяет оснастить системную память уникальными свойствами. Подобно ОЗУ, флэш-память модифицируется электрически внутрисистемно, но, подобно ПЗУ, флэш энергонезависима и хранит данные даже после отключения питания. Однако в отличие от ОЗУ флэш нельзя переписывать побайтно: ее нужно стереть перед записью новых данных.

Микросхемы флэш-памяти в последнее время получили большое распространение ввиду высоких потребительских качеств – простоты программирования, высокой скорости чтения и значительной емкости. Параметрические блоки флэш-памяти используются для хранения телефонных номеров, учета времени использования и идентификатора пользователя (SIM-карта) в сотовых телефонах. Производители автомобилей используют флэш-память в системах управления двигателями для хранения кодов ошибок и параметров оптимальных режимов работы. В каждом из подобных примеров изготовители экономят на расходах, связанных с необходимостью содержания складского запаса «прошитых» разными программами ПЗУ, используя флэш-память не только для хранения прикладных программ, но и параметров.

Следует отметить, что существует две разновидности флэш-памяти. Первая используется для хранения программ и имеет емкость порядка 1 Мбайт; вторая, – NAND EEPROM используется, в основном, в качестве мобильного носителя данных, имеет последовательный доступ к данным и емкость до 4 Гбайт. Таким образом, ПЗУ работает в режимах хранения и считывания.

Запоминающее устройство, реализующее функции основной памяти, размещают рядом с процессором в одном блоке, и такое ЗУ в этом смысле является внутрисистемным. Быстродействие внутреннего ЗУ должно быть соизмеримо с быстродействием процессора. Однако практически это требование не всегда удается выполнить: по временн***ы***м параметрам ОЗУ и ПЗУ отстают от процессора. Поэтому внутри ЭВМ обычно размещают еще и вспомогательную (буферную) память на быстродействующих регистрах, которая используется в

качестве *сверхоперативного ЗУ* (СОЗУ или cash) с небольшой информационной емкостью для кратковременного хранения текущих команд, адресов и данных.

Важнейшими характеристиками ЗУ являются:

- емкость, удельная емкость;
- быстродействие;
- энергопотребление;
- способность сохранять информацию при отключении питания.

Информационная емкость определяет число единиц информации в битах или байтах, которое БИС памяти может хранить одновременно. Она выражается через число ячеек *N* с указанием разрядности *n* в виде *M = N*n*. *Удельная емкость* – отношение информационной емкости к ее физическому объему. *Быстродействие*, как правило, характеризуется двумя параметрами:

1. Время выборки ($t_в$) – представляет интервал времени между передачей сигнала «выборка кристалла» (CS) при считывании информации и появлением информации на шине данных,
2. Время цикла записи ($t_{цз}$), которое определяется минимально возможным временем с момента подачи сигнала CS при записи и повторном обращении к памяти.

В качестве характеристики быстродействия памяти выбирается максимальное из $t_в$ и $t_{цз}$. В табл. 4.2 приведены типичные скоростные параметры для разных классов памяти.

Таблица 4.2 Временные параметры основных типов памяти, мкс

Технология Изготовления	Статические ОЗУ SRAM	ППЗУ PROM	ПЗУ ROM	Динамические ОЗУ DRAM
Биполярная МОП	30–100 200–500	50–150 300	50–150 350–1800	- 500

Память как функциональный узел

Рассмотрим микросхему памяти как «черный ящик», обратив основное внимание на назначение ее выводов, внешние и внутренние характеристики. На рис. 4.2 приведены графические изображения ОЗУ и ПЗУ.

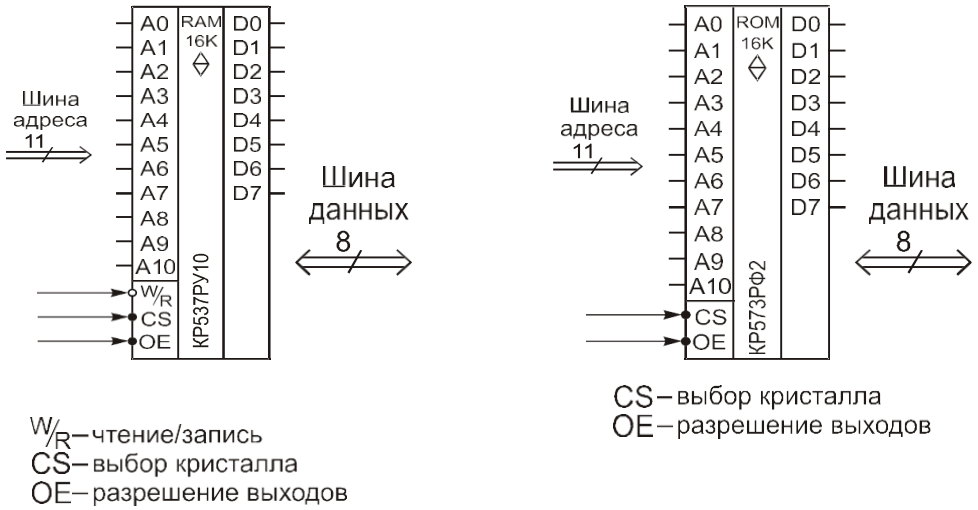


Рис. 4.2. Условные обозначения микросхем ОЗУ и ПЗУ

Сигналы и соответствующие выводы можно подразделить на *адресные*, *управляющие* и *информационные*. Число адресных входов A0 – A11 позволяет определить информационную емкость микросхемы: $2^{11} = 2048$ адресуемых ячеек памяти. Наличие восьми информационных выводов указывает на

восьмиразрядную организацию каждой ячейки. Поэтому общий объем памяти составляет 2048 × 8 бит = 2 Кбайт. Для управления режимом работы предусмотрены три сигнала: W/R – чтение/запись (Write/Read), CS – выбор микросхемы (Chip Select) и OE – разрешение выходов (Output Enable). Для обращения к микросхеме для записи или считывания одного байта информации необходимо подать сигнал CS с нулевым уровнем (разрешающий обращение) и сигнал W/R с соответствующим режиму уровнем: при записи – 1, при считывании – 0. Для упрощения дешифрации микросхемы памяти могут иметь несколько входов CS. Входы-выходы D совмещены, поэтому они обладают свойством двунаправленной проводимости. Отметим, что все операции чтения записи возможны только при низком активном уровне на входе OE. В противном случае шина данных переключается в высокоомное состояние, что равносильно ее отключению.

Многомодульная организация памяти

Механизм взаимодействия процессора с памятью основан на классической архитектуре вычислительной системы, состоящей из трех основных компонентов: процессора, памяти и устройств ввода/вывода, объединенных шинами данных, адреса и управления (рис. 4.3).

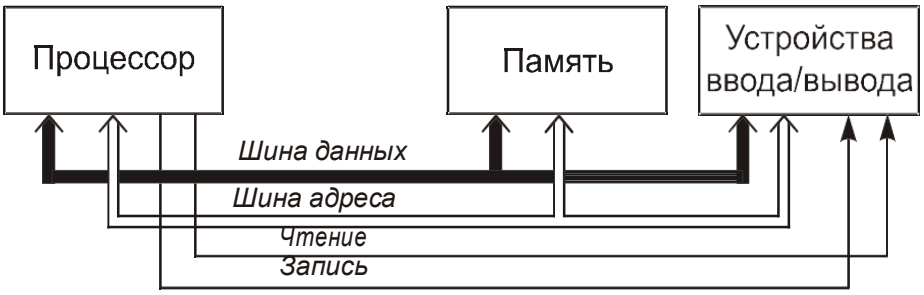


Рис. 4.3. Взаимодействие процессора и памяти

В простейшем случае в процессе работы программы МП выдает на адресную шину адрес требуемой ячейки памяти, сопровождаемый сигналом чтение/запись шины управления, а память извлекает или сохраняет информацию, находящуюся на шине данных. При обмене информацией с памятью ведущим является микропроцессор либо контроллер прямого доступа к памяти. При этом возникает проблема согласования быстродействия МП и памяти. Обычно процессор более быстродействующий, чем остальные компоненты, поэтому, чтобы согласовать временные параметры всех подсистем, применяются два способа:

1) синхронный, при котором между выдачей адреса и передачей данных производится фиксированная задержка (целое число тактов);

2) асинхронный, при котором после приема адреса и его дешифрации память отвечает процессору сигналом «Готовность».

На рис. 4.4 представлено построение и взаимодействие микропроцессора и памяти, состоящей из ОЗУ и ПЗУ одинаковой емкости и одинаковой организации 16 К×8. Известно, что процессор КР580ВМ80 с помощью шестнадцати адресных линий может адресовать 2^{16}= 64 Кбайт памяти. Организуем ее таким образом, чтобы ее первая половина (32 Кбайт) была отведена под ОЗУ, а вторая (32 Кбайт) – под ПЗУ. При этом все адресное пространство оказывается разделенным на четыре банка (две микросхемы ОЗУ и две микросхемы ПЗУ по

16 Кбайт каждая).

При обмене информацией процессор выставляет на адресной шине 16-разрядный адрес ячейки памяти, который сопровождается сигналом

«чтение/запись». Четырнадцать младших разрядов адреса (A0-A13) непосредственно подключены к соответствующим входам микросхем памяти, а две старшие адресные линии A14 и A15 определяют номер банка. Микросхема-дешифратор посредством сигнала CS (выбор кристалла) позволяет выбрать положение микросхемы ЗУ в адресном пространстве. Для данного случая это адреса 0000h-7FFFh для ОЗУ и 8000h-FFFFh для ПЗУ. Напомним, что запись информации в ППЗУ возможна только вне микропроцессорной системы в специальном программаторе.

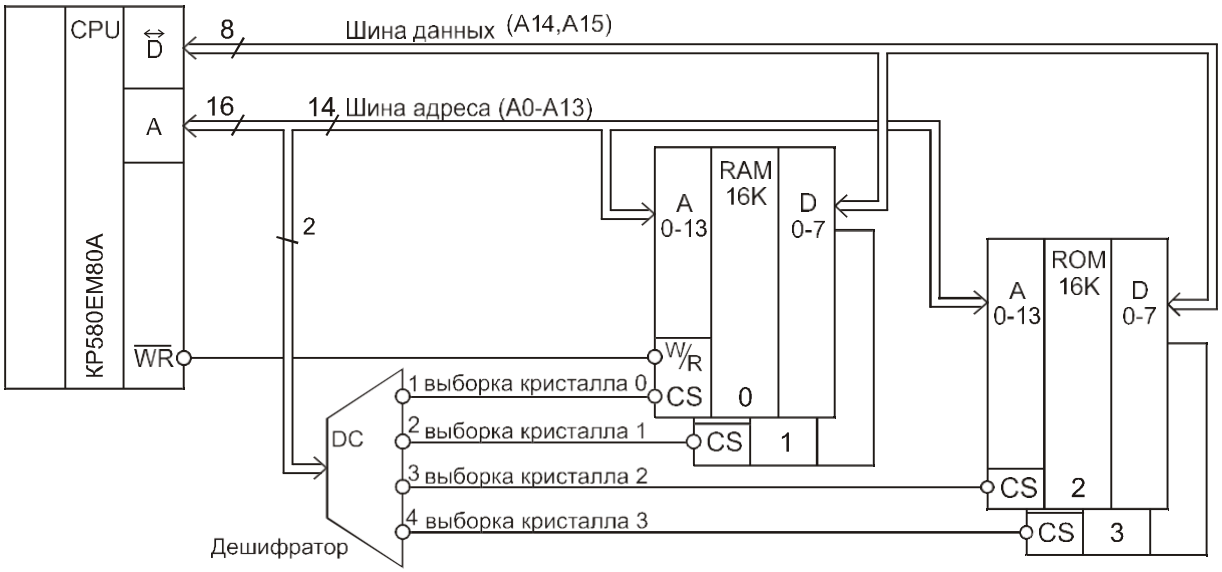


Рис. 4.4. Многомодульная организация памяти

В некоторых микропроцессорах (рис. 4.5) шина адреса и шина данных совмещается (мультиплексируется). В этом случае шина сначала используется для передачи адреса, а затем по ней передаются данные. При этом адрес запоминается во внешнем регистре, который стробируется специальным сигналом разрешения захвата адреса ALE.

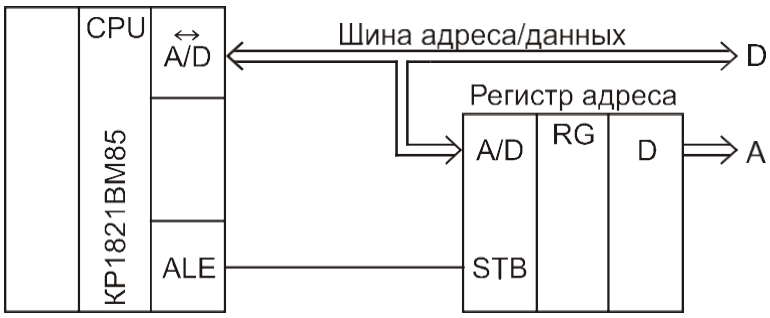


Рис. 4.5. Мультиплексирование шин

Аналогичный прием (совмещение шин адреса и данных) используется в большинстве современных микроконтроллеров.

Организация стековой памяти

Стеком называют безадресный способ организации памяти, доступ к которой организован по принципу: «*последним пришел, первым ушел*» (Last Input First Output – LIFO). Использование принципа доступа к памяти на основе

механизма LIFO началось с больших ЭВМ. В малых ЭВМ она стала широко использоваться в связи с удобствами реализации процедур вызова подпрограмм и при обработке прерываний. В микропроцессорных системах стековый принцип доступа к памяти стал широко использоваться из-за короткой длины машинного слова. Стек позволяет сохранять адреса возврата и флаги при обработке прерываний и вызове подпрограмм, а также передавать параметры в подпрограммы.

Принцип работы стековой памяти состоит в следующем (рис. 4.6). Когда слово A помещается в стек, оно располагается в первой свободной ячейке памяти. Следующее записываемое слово перемещает предыдущее на одну ячейку вверх и занимает его место и т.д. Запись 8-го кода, после H, приводит к переполнению стека и потере кода A. Считывание слов из стека осуществляется в обратном порядке, начиная с кода H, который был записан последним. Заметим, что выборка, например кода E, невозможна до выборки кода F, что определяется механизмом обращения при записи и чтении типа LIFO. Для фиксации переполнения стека желательно формировать признак переполнения.

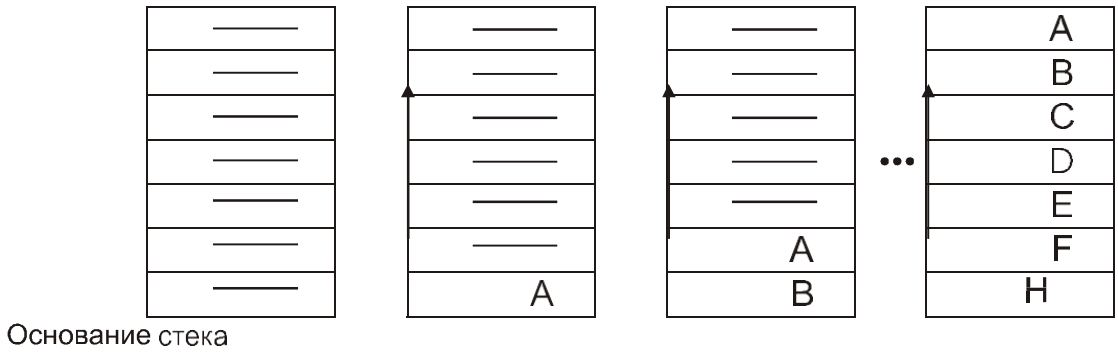


Рис. 4.6. Стек LIFO

Стек можно организовать двумя способами: на основе регистра сдвига и на основе ОЗУ и указателя стека (используется в процессоре КР580ВМ80) (рис. 4.7).

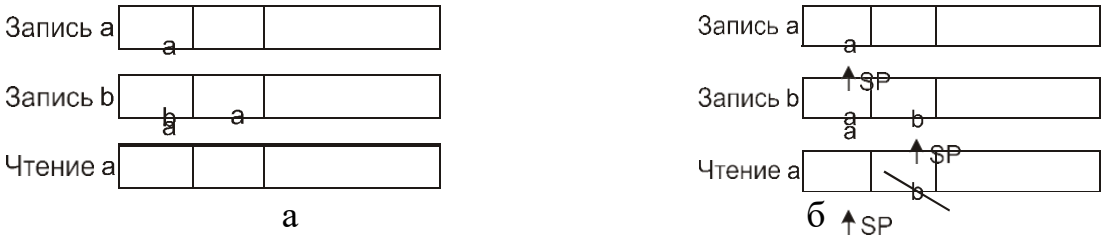


Рис. 4.7. Способы организации стека:
а – на основе регистра сдвига; б – на основе указателя стека и ОЗУ

Существует два типа стека:

- встроенный (указатель стека находится на кристалле МП);
- автономный (указатель стека находится на кристалле МП, а ОЗУ – на внешней микросхеме).

Встроенный стек имеет небольшой объем (несколько десятков килобайт), но обеспечивает максимальное быстродействие, так как не требует обращения к внешней памяти.

Чтение и запись в стек производится по следующему алгоритму:

PUSH a – запись в стек	**POP a** – чтение из стека
`SP:=SP+1`	`a:=M(SP)`
`M(SP):=a`	`SP:=SP-1`

В различных МП используются и другие алгоритмы записи/чтения. Так, стек может расти вниз, а не вверх, и его указатель может изменяться как до записи, так и после. Начальное значение указателя стека определяется программистом или операционной системой, а в некоторых МП, например MCS51, устанавливается при сбросе.

В сложных МП при вызове подпрограмм и прерываниях флаги сохраняются в стеке аппаратно. Если указатель попал в область, отведенную для векторов прерываний, команд или данных, то произойдет автоматическое переполнение стека. В простых МП за сохранением регистра флагов должен следить сам программист.

Тема 5. Організація роботи мікропроцесорної системи.

Обмен информацией в микропроцессорах и микропроцессорных системах

Обмен информацией в микропроцессорных системах происходит в циклах обмена информацией. Под циклом обмена информацией понимается временной интервал, в течение которого происходит выполнение одной элементарной операции обмена по шине. Например, пересылка данных из процессора в память или же пересылка данных из устройства ввода/вывода в процессор. В пределах одного цикла также может передаваться и несколько кодов данных.

Циклы обмена информацией делятся на два основных типа:

• Цикл записи (вывода), в котором процессор записывает (выводит) информацию;

• Цикл чтения (ввода), в котором процессор читает (вводит) информацию.

Во время каждого цикла устройства, участвующие в обмене информацией, передают друг другу информационные и управляющие сигналы в строго установленном порядке или, как еще говорят, в соответствии с принятым протоколом обмена информацией.

Длительность цикла обмена может быть постоянной или переменной, но она всегда включает в себя несколько периодов сигнала тактовой частоты системы. То есть даже в идеальном случае частота чтения информации процессором и частота записи информации оказываются в несколько раз меньше тактовой частоты системы.

Обмен информацией происходит с использованием системной магистрали (системной шины, интерфейса) микропроцессорной системы. Системный интерфейс содержит три основные информационные шины: адреса, данных и управления. Шина данных — это основная шина по которой передается информация. Количество ее разрядов (линий связи) определяет скорость и эффективность обмена, а также максимально возможное количество команд. Шина данных всегда двунаправленная, так как предполагает передачу информации в обоих направлениях. Наиболее часто встречающийся тип выходного каскада для линий этой шины — выход с тремя состояниями. Обычно шина данных имеет 8, 16, 32 или 64 разряда. За один цикл обмена по 64-разрядной шине может передаваться 8 байт информации, а по 8-разрядной — только один байт. Разрядность шины данных определяет и разрядность всей магистрали. Например, когда говорят о 32-разрядном системном интерфейсе, подразумевается, что он имеет 32-разрядную шину данных.

Шина адреса — вторая по важности шина, которая определяет максимально возможную сложность микропроцессорной системы, то есть допустимый объем

памяти и, следовательно, максимально возможный размер программы и максимально возможный объем запоминаемых данных. Количество адресов, обеспечиваемых шиной адреса, определяется как 2N, где N — количество разрядов. Например, 16-разрядная шина адреса обеспечивает 65536 адресов. Разрядность шины адреса обычно кратна 4 и достигает 64 разрядов.

Шина адреса может быть однонаправленной (когда магистралью всегда управляет только процессор) или двунаправленной (когда процессор может временно передавать управление магистралью другому устройству). Наиболее часто используются типы выходных каскадов с тремя состояниями или обычные.

Для снижения общего количества линий связи процессора применяется мультиплексирование шин адреса и данных. То есть одни и те же линии связи используются в разные моменты времени для передачи как адреса, так и данных (сначала — адрес, затем — данные). Для фиксации этих моментов (стробирования) служат специальные сигналы на шине управления. Мультиплексированная шина адреса/данных обеспечивает меньшую скорость обмена, требует более длительного цикла обмена.

Шина управления — это вспомогательная шина, управляющие сигналы на которой определяют тип текущего цикла и фиксируют моменты времени, соответствующие разным частям или стадиям цикла. Кроме того, управляющие сигналы обеспечивают согласование работы процессора с работой памяти или устройства ввода/вывода. Управляющие сигналы также обслуживают запрос и предоставление прерываний, запрос и предоставление прямого доступа к памяти.

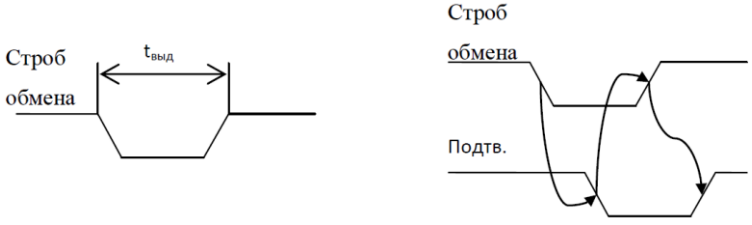
Линии шины управления могут быть как однонаправленными, так и двунаправленными.

Основные управляющие сигналы — это стробы обмена, то есть сигналы, формируемые процессором и определяющие моменты времени, в которые производится пересылка данных по шине данных, обмен данными. Чаще всего в магистрали используются два различных строба обмена:

• Строб записи (вывода), который определяет момент времени, когда устройство-исполнитель может принимать данные, выставленные процессором на шину данных;

• Строб чтения (ввода), который определяет момент времени, когда устройство-исполнитель должно выдать на шину данных код данных, который будет прочитан процессором.

При этом большое значение имеет то, как процессор заканчивает обмен в пределах цикла, в какой момент он снимает свой строб обмена. Возможны два пути решения (рис. 5.1):



а) синхронный обмен б) асинхронный обмен

Рис. 5.1. Синхронный обмен и асинхронный обмен

• при синхронном обмене процессор заканчивает обмен данными самостоятельно, через раз и навсегда установленный временной интервал

выдержки ($t_{выд}$), то есть без учета реального быстродействия остальных устройств участвующих в обмене;

• при асинхронном обмене процессор заканчивает обмен только тогда, когда устройство-исполнитель подтверждает выполнение операции специальным сигналом ответным сигналом.

Достоинства синхронного обмена — более простой протокол обмена, меньшее количество управляющих сигналов. Недостатки — отсутствие гарантии, что исполнитель выполнил требуемую операцию, а также высокие требования к быстродействию исполнителя.

Достоинства асинхронного обмена — более надежная пересылка данных, возможность работы с самыми разными по быстродействию исполнителями.

Недостаток — необходимость формирования сигнала подтверждения всеми исполнителями, то есть дополнительные аппаратурные затраты.

Синхронный и асинхронный типы обмена обеспечивают разную производительность работы микропроцессорной системы. С одной стороны, при асинхронном обмене требуется какое-то время на выработку, передачу дополнительного сигнала и на его обработку процессором. С другой стороны, при синхронном обмене приходится искусственно увеличивать длительность строба обмена для соответствия требованиям большего числа исполнителей, чтобы они успевали обмениваться информацией в темпе процессора. Поэтому иногда в магистрали предусматривают возможность как синхронного, так и асинхронного обмена, причем синхронный обмен является основным и довольно быстрым, а асинхронный применяется только для медленных исполнителей.

Обмен информацией в контроллерах устроен принципиально по-другому без использования внешнего системного интерфейса. Каждый МК имеет некоторое количество линий ввода/вывода, которые объединены в многоразрядные (чаще 8-разрядные) параллельные порты ввода/вывода. В памяти МК каждому порту ввода/вывода соответствует свой адрес регистра данных. Обращение к регистру данных порта ввода/ вывода производится теми же командами, что и обращение к памяти данных. Кроме того, во многих МК отдельные разряды портов могут быть опрошены или установлены командами битового процессора.

В зависимости от реализуемых функций различают следующие типы параллельных портов:

• однонаправленные порты, предназначенные только для ввода или только для вывода информации;

• двунаправленные порты, направление передачи которых (ввод или вывод) определяется в процессе инициализации МК;

• порты с альтернативной функцией (мультиплексированные порты). Отдельные линии этих портов используются совместно со встроенными периферийными устройствами МК, такими как таймеры, АЦП, контроллеры последовательных интерфейсов;

• порты с программно управляемой схемотехникой входного/выходного буфера.

Порты выполняют роль устройств временного согласования функционирования МК и объекта управления, которые в общем случае работают асинхронно. Различают три типа алгоритмов обмена информацией между МК и внешним устройством через параллельные порты ввода/вывода:

• режим простого программного ввода/вывода;

• режим ввода/вывода со стробированием;

• режим ввода/вывода с полным набором сигналов подтверждения обмена.

В современных МК, как правило, обеспечивается индивидуальный доступ к портам, что позволяет использовать каждую линию независимо в режиме ввода или вывода.

Необходимо обратить особое внимание на то, что при вводе данных с порта считывается значение сигнала, поступающее на внешний вывод. При выводе информации на порт поступает содержимое регистра данных порта ассоциированное с одним из регистров оперативной памяти.

Использование регистра адреса/данных

Использование пары регистров *HL*в качестве указателя адреса является интересным свойством типового МП. Обычно рассматривают ее использование в качестве указателя адреса, когда она временно берет на себя роль основного счетчика команд, указывая адрес ячейки памяти или УВВ. Многие широко распространенные МП (например, Intel 8080/8085, Z 80) содержат регистры такого типа. Регистры адреса/данных в рассматриваемом типовом МП называются также парой *HL*.-регистров, регистром адреса, счетчиком данных или указателем адреса.

Рассмотрим простую задачу сложения содержимого трех последовательных ячеек памяти с размещением суммы в следующей ячейке памяти (выполнение ее показано на рис. 5.2).

Программа загружена в ячейки памяти 2000H — 200AH, а три слагаемых (0CH + 0AH + 07H) — в ячейки памяти 2100H—2102H. Программа содержит 6 команд, записанных справа на рис. 5.2. Не следует забывать, что текущая сумма будет всегда помещаться в аккумулятор, содержащий вначале первое слагаемое (0CH).

Команда *1* имеет КОП 3AH (рис. 5.2) и приказывает МП ЗАГРУЗИТЬ (LOAD) в аккумулятор содержимое ячейки памяти 2100H. Выполнение этой команды прямой загрузки аккумулятора показано на рис. 5.3, *а.* После выполнения команды аккумулятор будет содержать первое слагаемое (0CH).

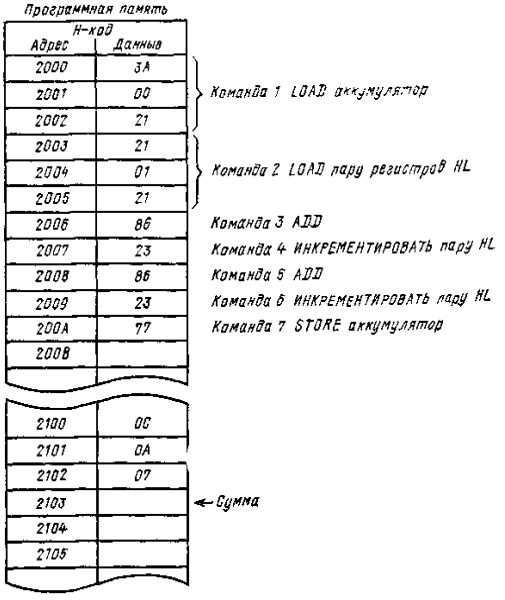


Рис. 5.2. Воображаемая память и команды в примере сложения

Команда *2* приказывает МП загрузить (LOAD) 2101H в пару регистров

HL,емкость которых 16 бит. Это число представляет собой адрес памяти данных. Более точно команду *2* можно сформулировать так: загрузить пару регистров *HL*непосредственно следующими за КОП данными, ее выполнение приведено на рис. 5.3, *б.* Заметим, что содержимое первой ячейки памяти (2004H) загружается в младший байт L, следующей за ней — в старший байт *H* пары регистров *HL*.

Команда *3* приказывает МП выполнить операцию сложить (ADD) содержимое аккумулятора с содержимым ячейки памяти, адрес которой содержится в паре регистров *HL*. Ее выполнение приведено на рис. 5.3, *в* (команда ADD). Пара регистров *HL*указывает на ячейку памяти 2101H, и АЛУ складывает свое содержимое ($0000\ 1010_2$) с содержимым аккумулятора ($0000\ 1100_2$), что дает сумму ($0001\ 0110_2$), помещаемую в аккумулятор.

Команда *4* приказывает МП инкрементировать (увеличить на 1) содержимое пары регистров *HL*(см. рис. 5.3, *г).* Заметим, что изменился только младший байт пары регистров *HL*.

Команда *5* снова приказывает МП сложить (ADD) содержимое аккумулятора и ячейки памяти с адресом 2102H, на которую указывает пара регистров *HL*(см. рис. 4.3,*б*) .Оба содержимых складываются, что дает сумму ($0001\ 1101_2$), помещаемую в аккумулятор.

По команде *6* содержимое пары регистров *HL*снова инкрементируется (см. рис. 5.3, *е).*

Команда *7* приказывает МП поместить (STORE) содержимое аккумулятора, т. е. окончательную сумму ($0001\ 1101_2$) в ячейку памяти, на которую указывает пара регистров *HL* (см. рис. 5.3, *ж), т.е.* по адресу 2103H.

Команды *3, 5, 7,* взаимодействующие с парой регистров *HL* как с указателем адреса, используют косвенно-регистровый способ адресации. Его мы изучим в следующей главе.

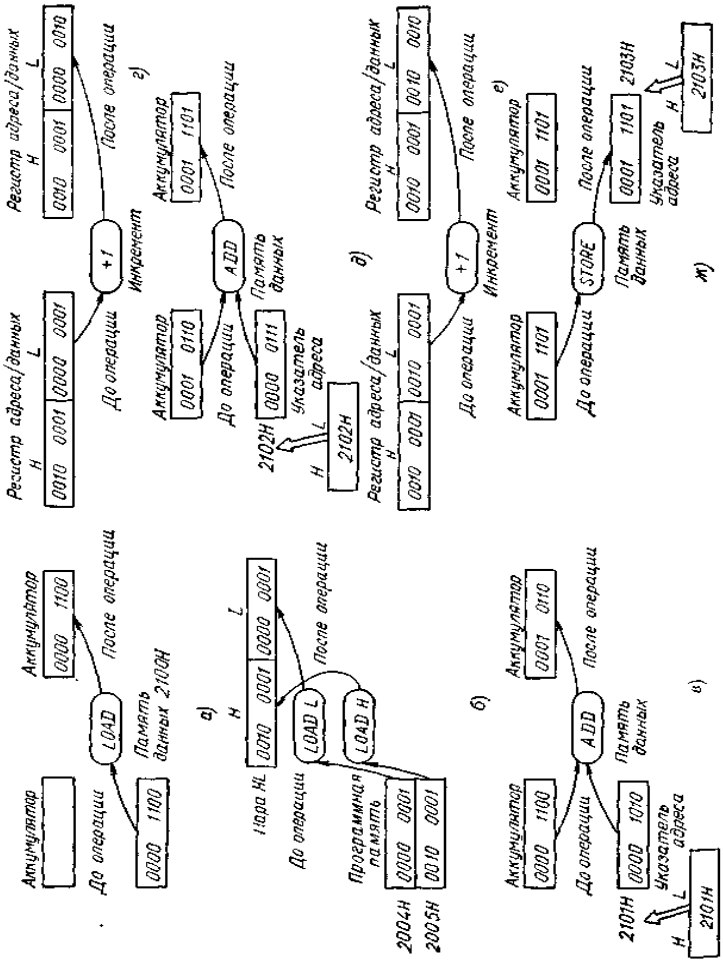


Рис. 5.3. Выполнение операций в соответствии с программой рис. 5.2:

а — команда 1 загрузки аккумулятора; б — команда 2 загрузки пары регистровHL\ в — команда 3 сложения, г — команда 4 инкремента пары регистров; д —команда 5 сложения; е — команда 6 инкремента пары регистров HL\ ж — команда 7 размещения в памяти содержимого акумулятора.

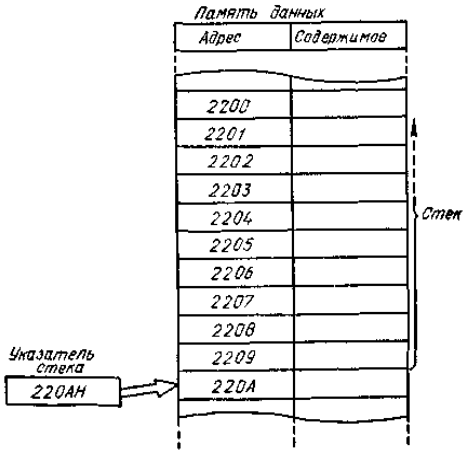
Использование указателя стека

Типовый микропроцессор содержит указатель стека — специализированный 16-разрядный регистр-счетчик, содержимым которого всегда является адрес. Этот адрес принадлежит особой группе ячеек памяти данных, которая называется *стеком.* В некоторых МП стек может быть составлен из группы физически локализованных на кристалле МП ячеек памяти. Когда микропроцессором выполнялась подпрограмма обслуживания прерывания, текущие данные во всех регистрах МП должны были временно сохраняться. Эта сохранность обеспечена стеком. А когда подпрограмма полностью выполнена, содержимое счетчика команд должно быть сохранено таким образом, чтобы МП мог возвратиться в соответствующее место в программной памяти. Зона временной памяти является стеком. Напомним, что подпрограмма является короткой, часто используемой, специализированной программой (например, умножить).

Стек типового микропроцессора будет содержаться в ОЗУ, и его положение определяется программистом. Указатель стека загружается старшим адресом, представляющим собой вершину стека (рис. 5.4). В этом случае указательстека содержит 220АН, что на единицу старше первой ячейки памяти стека 2209Н.

Данные можно записать в стек, используя команды PUSH (поместить) или CALL (вызвать). Они могут быть считаны из стека по командам POP (извлечь)

или RETURN (возврат). Стек функционирует как память с последовательным доступом по типу: данные, поступившие последними, извлекаются первыми (тип LIFO от *Last In — First Out* — последний входит — первый выходит, или FILO от *First In — Last Out* —первый входит — последний выходит).



Рнс. 5.4. Расположение стека в ОЗУ

Команда загрузки в стек (PUSH) приводит к результату, показанному на рис. 4.5, *а.* Содержимое пары регистров *HL*помещается в стек. Отметим, что двухбайтовая пararегистров *HL*должна быть размещена в двух ячейках памяти стека. Последовательность событий может быть описана в соответствии с номерами, показанными в кружках на рис. 5.5.

1. Указатель стека МП декрементируется от 220АН до 2209Н.
2. Указатель стека показывает на ячейку памяти 2209Н по адресной шине и старший байт ($0000\ 0000_2$) помещается в стек.
3. Указатель стека снова декрементируется от 2209Н до 2208Н.
4. Указатель стека указывает на ячейку памяти 2208Н (по адресной шине системы) и младший байт из регистра данных ($0000\ 1111_2$) загружается в стек.

На рис. 5.5, *б* показано выполнение другой операции загрузки. На этот раз в стек загружается содержимое аккумулятора и регистра состояния. Проследим снова за событиями, отмеченными цифрами в кружках.

5. До операции указатель стека указывает на последнюю ячейку стека. Ее называют вершиной стека. Затем указатель стека декрементируется до 2207Н.
6. Указатель стека указывает на ячейку памяти 2207Н, и содержимое аккумулятора ($0101\ 0101_2$) загружается в стек по этому адресу.
7. Указатель стека декрементируется от 2207Н до 2206Н.
8. Указатель стека указывает на ячейку памяти 2206Н. Содержимое регистра состояния ($1111\ 1111_2$) загружается по этому адресу.

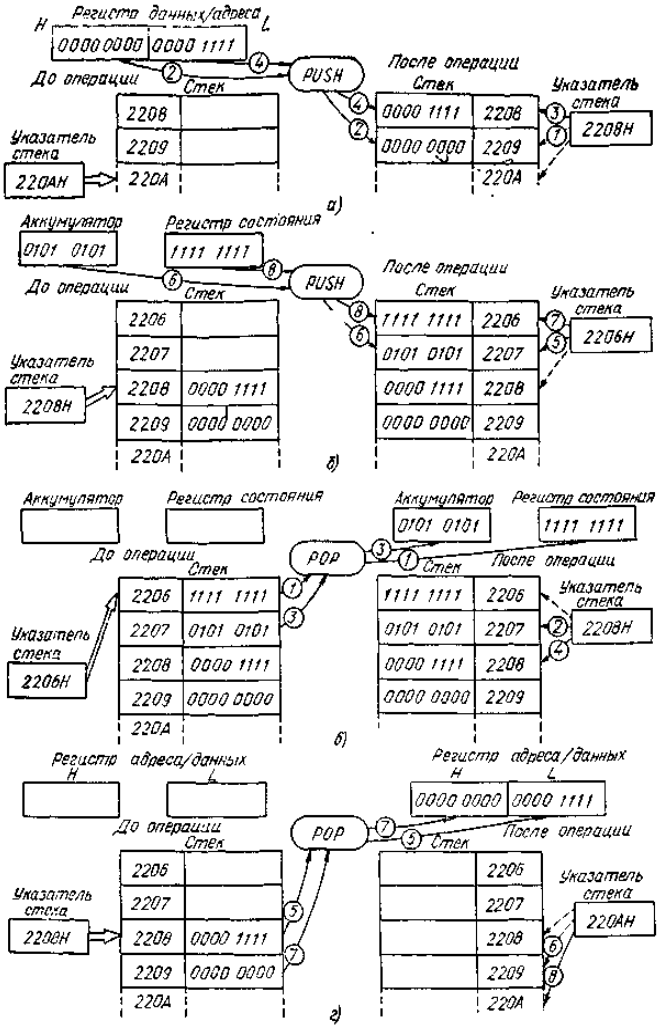


Рис. 5.5. Операции со стеком:

а — помещение в стек содержимого регистра адреса/данных; б — помещение в стек содержимого аккумулятора и регистра состояния; в — извлечение из стека содержимого аккумулятора н регистра состояния; г ~ извлечение из стека содержимого регистра адреса/данных

Стек может продолжать расти, пока длится процесс загрузки в него (на рис. 5.4 показано, что стек растет вверх).Стек не имеет ограничений, за исключением тех, которые обусловлены наличием других программ в ОЗУ.

Обычно каждой команде загрузки в стек (PUSH) позже будет соответствовать команда извлечения из стека (POP), по которой данные берутся из вершины стека. Поскольку стек является памятью типа LIFO (последний вошел — первый вышел), данные должны извлекаться из стека в порядке, обратном загрузке. На рис. 5.5, *в* и *г* подробно показаны операции извлечения из стека.

Рассмотрим команду POP на рис.5.5, *в.* Аккумулятор и регистр состояния свободны до операции извлечения из стека. Следуем последовательности, указанной цифрами в кружках.

1. Указатель стека указывает на вершину стека, т. е. на адрес 2206H. Содержимое регистра состояния ($1111\ 1111_2$) извлечено из стека и переслано в АЛУ.

2. Указатель стека инкрементирован с 2206H до 2207H.

3. Указатель стека указывает на адрес 2007H стека. Вершина стека извлекается, и ее содержимое пересылается в аккумулятор АЛУ.

4. Указатель стека инкрементирован до 2208H и указывает теперь на следующий адрес извлечения из стека.

Содержимое аккумулятора и регистра состояния было восстановлено до тех значений, которые были до операции PUSH, показанной на рис. 5.5, *б.*

Затем (на рис. 5.6, *г)* содержимое регистра адреса/данных в свою очередь извлекается из стека. Снова последуем согласно заключенным в кружки цифрам:

5. Указатель стека указывает на вершину стека (адрес 2208H). Содержимое этой ячейки памяти стека извлекается и пересылается в младший байт пары регистров *HL*.

6. Указатель стека инкрементируется до 2209H.

7. Указатель стека показывает на вершину стека, т. е. теперь адрес 2209H, содержимое которого ($0000\ 0000_2$), передается в старший байт пары регистров *HL*.

8. Указатель стека инкрементируется от 2209H до 220AH для последующей операции загрузки в стек (PUSH) или извлечения из стека (POP).

Извлечение данных из стека и их восстановление в регистре адреса/данных является действием, обратным операции загрузки в стек (PUSH), выполненной на рис. 5.5, *а.* Команды PUSH и POP используются всегда совместно, однако между ними располагаются другие команды, которые меняют данные, содержащиеся в регистрах МП.

Наш типовой МП загружает в стек и извлекает содержимое пары регистров. Некоторые МП осуществляют эти операции только с однобайтовыми регистрами, единственной командой. В других МП указатель стека может указывать на пустую ячейку памяти по ближайшему большему по отношению к вершине стека адресу вместо указания на вершину стека, как в предыдущем случае. При этих условиях имеется возможность сохранить в памяти то, что программист загрузил в начале (адрес 220AH в нашем случае) в указатель стека для определения его адреса.

Тема 6. Системи числення та перетворення чисел з однієї системи числення в іншу.

Краткие теоретические сведения

В ЭВМ арифметические и логические действия производятся над числами, представленными в виде специальных (машинных) кодов в принятой для данной машины системе счисления.

Под *системой счисления* понимается способ наименования и изображения чисел с помощью символов, имеющих определенные количественные значения.

Символы, применяемые для изображения чисел, называются *цифрами.*

В зависимости от способа изображения чисел с помощью цифр системы счисления делятся на позиционные и непозиционные.

Позиционной называется система счисления, в которой количественное значение каждой цифры зависит от ее места (позиции) в числе.

Примером такой системы может служить общепринятая в настоящее время арабская (десятичная) система счисления.

В *непозиционной* системе счисления цифры не меняют своего количественного значения при изменении положения в записи числа. К таким системам, например, относится римская система счисления, которая, однако, из-за сложности записи в ней многозначных чисел для вычислений не применяется.

В позиционной системе счисления числа записываются в виде последовательности цифр

$$A = a_{m-1}a_{m-2}\ldots a_k\ldots a_1a_0$$

Позиции, пронумерованные индексами k (в данном случае в пределах ($0<k<m$-1), называются разрядами числа. Индекс m соответствует количеству разрядов.

Каждая цифра a_k в записанной последовательности может принимать одно из некоторого количества N возможных значений, т.е. $N\text{-}1 \geq a_k \geq 0$.

Количество *(N)* различных цифр, используемых для изображения чисел в позиционной системе счисления, называется *основанием системы счисления.*

Поскольку цифра a_k соответствует количеству единиц k-го разряда, содержащихся в числе, то основание *(N)* позиционной системы счисления указывает, во сколько раз единица *(k+1)* -го разряда больше единицы младшего k-го разряда. Следовательно, записанную выше последовательность цифр, соответствующих целому числу, можно представить в виде:

$A_m = a_{m-1}N^{m-1}+a_{m-2}N^{m-2}+... +a_1N^{m-2}+a_0N^0$.

В общем случае выражение для любого числа, состоящего из целой и дробной частей (неправильная дробь), будет представлять собой ряд:

$$A_{(N)}=\pm[\, a_{m-1}N^{m-1}+a_{m-2}N^{m-2}+... +a_1N^1+a_0N^0+ a_{-1}N^{-1}+... +a_{-l}N^{-l}\,], \qquad (1)$$

где m и l — число разрядов соответственно целой и дробной частей числа, N^i-вес i-го разряда. Следует заметить, что в записи вида (1) основание N может быть разным для различных разрядов, например, запись угловых величин в градусах, минутах и секундах или запись величин, характеризующих время. Такие системы называются *неоднородными,* в отличии от *однородных* систем с равными основаниями для всех разрядов. В настоящее время в вычислительных машинах используются только однородные системы счисления.

Основание N позиционной системы счисления определяет и ее название. Так, например, общепринятая десятичная система счисления имеет основание N=10. Любое число в этой системе записывается с помощью различных цифр:

a_k=0,1,2,3,4,5,6,7,8,9

В качестве основания можно выбирать любое другое число. Такими числами, а следовательно и основаниями, могу быть: два, три, четыре, пять и т.д.

Исторически так сложилось, что именно десятичная система оказалась общепринятой и широко применяемой при ручном счете и в электромеханических вычислительных устройствах, Однако с точки зрения простоты конструктивного выполнения отдельных устройств для ЭВМ оказываются удобными также другие системы счисления с основаниями два -двоичная, восемь- восьмеричная и шестнадцать — шестнадцатиричная.

Двоичная система счисления

Цифровые вычислительные машины работают с двоичными числами. Двоичная система счисления или система с основанием 2 использует только цифры 0 и 1. Эти двоичные числа названы битами (от binary digit). Физически в цифровых электронных системах бит 0 представлен напряжением LOW (низким), а бит 1 — напряжением HIGH (высоким).

Человеческая деятельность предполагает использование десятичной системы счисления. Десятичная система, или система с основанием 10, содержит 10 цифр (от 0 до 9). Она также характеризуется значением позиции (или весом). В табл. 6.1 показано, например, что десятичное число 1327 равно одной тысяче, плюс три сотни, плюс два десятка, плюс семь единиц (1000+300 + 20+7 = 1327).

Таблица 6.1. *Значения позиций десятичных чисел*

Степень основания	10^3	10^2	10^1	10^0
Значения позиций	1000	100	10	1
Двоичные	1	3	2	7
Десятичные	1000 +	300 +	20 +	7 = 1327

Двоичная система обладает также свойством уравновешивания. В табл. 6.2 приведены десятичные значения первых четырех двоичных позиций. Двоичное число 1001 (произносится: один, нуль, нуль, один) преобразовано, таким образом, в свой десятичный эквивалент 9. Бит единицы двоичного числа в табл. 6.2 называется младшим битом (МБ), бит восьмерки — старшим битом (СБ).

Таблица 6.2. *Значения позиций двоичных чисел*

Степень снования	2^3	2^2	2^1	2^0
Значение позиции	8 СБ	4	2	1 МБ
Двоичные	1	0	0	1
Десятичные	8 +	0 +	0 +	1 = 9

В табл. 6.3 приведены десятичные числа от 0 до 15, а также их двоичные эквиваленты. Те, кто работает в области использования ЭВМ, должны, по меньшей мере, запомнить эти двоичные числа.

Таблица 6.3. *Десятичные числа и их двоичные эквиваленты*

Десятичные		Двоичные				Десятичные		Двоичные			
10	1	8	4	2	1	10	1	8	4	2	1
	0				0		8	1	0	0	0
	1				1		9	1	0	0	1
	2			1	0	1	0	1	0	1	0
	3			1	1	1	1	1	0	1	1
	4		1	0	0	1	2	1	1	0	0
	5		1	0	1	1	3	1	1	0	1
	6		1	1	0	1	4	1	1	1	0
	7		1	1	1	1	5	1	1	1	1

Как преобразовать двоичное число 1011 0110 (т. е. один, нуль, один, один, нуль, один, один, нуль) в его десятичный эквивалент? Процедура преобразования выполняется в соответствии с табл. 6.4. Десятичные значения каждой позиции записаны под каждым битом, затем десятичные числа суммируются (128+32+16+4+2=182), что дает 182.

Таблица 6.4. *Двоично-десятичные преобразования*

Степень основания	2^7	2^6	2^5		2^4	2^3	2^2		2^1	2^0	
Значение позиций	128	64	32		16	8	4		2	1	
Двоичный	1		1		1	0	1		1	0	
Десятичный	128	+	32	+	16	+	4	+	2	==	182

Обычно основание системы счисления указывается индексами. Таким образом, число 1011 0110_2 является двоичным (или основания 2), а число 182_{10} — десятичным: 1011 $0110_2 = 182_{10}$.

Как преобразовать десятичное 155 в его двоичный эквивалент? Процедура преобразования приведена на рис. 6.5.

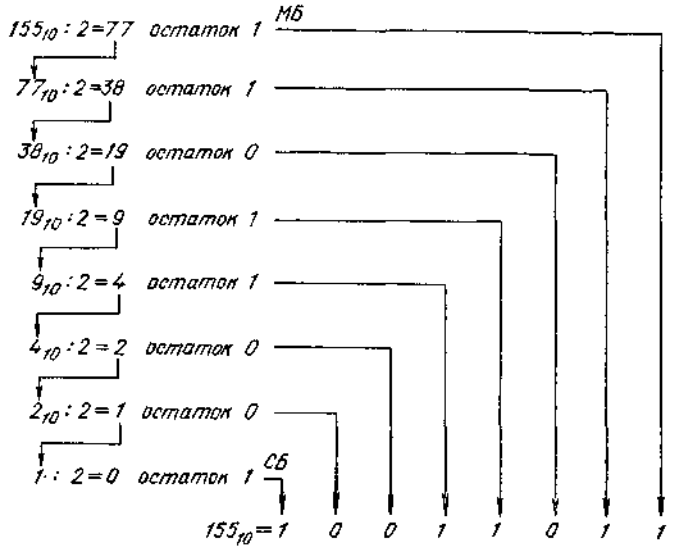


Рис. 6.5. Двоично-десятичные преобразования

Десятичное 155 сначала делится на 2, что дает нам частное 77 и остаток *1*. Этот остаток становится МБ двоичного числа и помещается в эту позицию (см. рис. 5.5). Затем частное (77) перемещается, как показывает стрелка, и становится следующим делимым. Затем каждое частное последовательно делится на 2 до тех пор, пока не получится частное, равное 0, и остаток, равный 1 (см. предпоследнюю строку на рис. 6.5). Последняя строка на рис. 5.5 дает нам результат $155_{10} = 1001\ 1011_2$.

Процедура преобразования целых десятичных чисел в двоичные - это частный случай процедуры перевода чисел из одной системы счисления в другую. Предположим, что необходимо преобразовать десятичное число 10 в двоичное. Для этого проделаем следующее.

1.Разделим подлежащее преобразованию число на основание системы счисления, в которой число должно быть представлено. В данном случае 10 следует поделить на 2. При делении на 2 остаток может быть равен 1 или 0. Значение остатка присваивается младшему значащему разряду (МЗР) искомого числа. Для рассматриваемого примера частное равно 5, а остаток - нулю, т.е. 1 - й разряд равен нулю.

2.Результат деления па первом шаге необходимо разделить еще раз на 2. Остаток (0 или 1) используется в качестве значения следующего по значимости разряда. В данном случае частное от деления 5 на 2 равно 2, а остаток, т. е. значение 2-го разряда, равно 1.

3.Результат деления на предыдущем шаге необходимо разделить па 2, а значение остатка присвоить очередному разряду. В данном случае частное равно 1, а остаток равен нулю, т.е. 3-й разряд равен нулю.

4.Шаги описанной процедуры повторяются до тех пор, пока частное, полученное в результате очередной операции деления, не станет равным нулю. Тогда остаток от последнего деления используется в качестве значения

старшего значащего разряда (СЗР). В данном случае частное от деления 1 па 2 составляет нуль, а остаток равен 1, поэтому значение 4-го разряда равно 1.

Итак, получено целое двоичное число 1010. Рассмотрим еще два примера преобразования десятичных чисел в двоичные.

Пример 1. Преобразованиедесятичного числа 57_{10} в двоичное число:

Шаг	Деление	Частное	Остаток
1	57/2	28	1 (МЗР)
2	28/2	14	0
3	14/2	7	0
4	7/2	3	1
5	3/2	1	1
6	1/2	0	1 (СЗР)

Результат: $57_{10} = 1000\ 0110_2$,

Пример 2. Преобразование десятичного числа 134_{10} в двоичное число:

Шаг	Деление	Частное	Остаток
1	134/2	67	0 (МЗР)
2	67/2	33	1
3	33/2	16	1
4	16/2	8	0
5	8/2	4	0
6	4/2	2	0
7	2/2	1	0
8	1/2	0	1 (СЭР)

Результат $-134_{10} = 1000\ 0110_2$

Изложенная процедура применима к преобразованию целых (или целой части) десятичных чисел в двоичные. Для дробных чисел (или дробных частей вещественных чисел) требуется отдельная, хотя и похожая, процедура. Если преобразовать выполнено отдельно для целой и дробной частей числа, то результат получают путем записи двоичных эквивалентов этих частей соответственно слева и справа от двоичной точки.

Процедуру преобразования десятичной дроби в двоичную рассмотрим на примере преобразования числа 0.375.

1. Преобразование осуществляется умножением дроби на основание системы счисления, в которой дробь должна быть представлена. В данном случае умножаем на 2: 2 x 0.375 = 0.75.

2. Если результат умножения меньше 1, то старшему значащему разряду присваивается значение 0; если больше 1, то присваивается 1. Поскольку 0.75 < 1, то СЗР = 0.

3. Результат предыдущей операции умножения опять умножается на 2. Заметим, что если бы результат предыдущей операции умножения был больше 1, то в данной операции умножения участвовала лишь его дробная часть. В

данном случае 2 x 0.75 = 1.5.

4. Если полученный результат меньше 1, то следующему по значимости (ближайшему справа) разряду присваивается значение 0; если равен или больше 1, то присваивается 1. В рассматриваемом примере 1.5 > 1, поэтому значение разряда 2 равно 1.

5 Шаги описанной процедуры повторяются до тех пор, пока либо результат умножения не будет точно равен 1, либо не будет достигнута требуемая точность. В данном случае после выполнения очередного шага результат равен (2 x 0.5=1.0). Поэтому очередному разряду, являющемуся младшим значащим разрядом, присваивается значение 1.

Следовательно, получена двоичная дробь 0.011.

Следует отметить, что не всегда путем повторения операций умножения можно достичь результата умножения, точно равного 1. В таком случае процесс повторения останавливают по достижении необходимой точности, а целую часть результата последней операции умножения используют в качестве значения младшего значащего разряда.

Рассмотрим еще два примера преобразования десятичных дробей в двоичные.

Пример 1. Преобразование десятичного числа 0.34375_{10} в двоичное:

Умножение	Результат в целочисленной форме
2 x 0.34375 = 0.6875	0 (СЗР)
2x0.6875= 1.375	1
2 x 0.375 = 0.75	0
2 × 0,75 = 1,5	1
2 × 0,5 = 1,0	1
2 × 0 = 0	0 (МЗР)

Результат: $0{,}01011_2$.

Пример 2. Преобразование десятичного числа $0{,}3_{10}$ в двоичное:

Умножение	Результат в целочисленной форме
2 × 0,3 = 0,6	0
2 × 0,6 = 1,2	1
2 × 0,2 = 0,4	0
2 × 0,4 = 0,8	0
2 × 0,8 = 1,6	1
2 × 0,6 = 1,2	1
2 × 0,2 = 0,4	0
2 × 0,4 = 0,8	0
2 × 0,8 = 1,6	1
2 × 0,6 = 1,2	1
2 × 0,2 = 0,4	0

Процедура преобразование в примере 2 носит характер бесконечного построения группы одинаковых операций и результатов. Поэтому ограничимся восемью разрядами. Тогда получим $0{,}3_{10} = 0{,}01001100_2$.

Шестнадцатеричная система счисления

Ячейка памяти типичной микро-ЭВМ может содержать двоичное число 1001 1110. Такая длинная цепь нулей и единиц сложна для запоминания и неудобна для ввода с клавиатуры. Число 1001 1110 могло бы быть преобразовано в десятичное, что дало бы 158_{10}, но процесс преобразований занял бы много времени. Большая часть систем микроинформатики использует шестнадцатеричную форму записи, чтобы упростить запоминание и использование таких двоичных чисел, как 1001 1110.

Шестнадцатеричная система счисления *(hexadecimal)* [1], или система с основанием 16, использует 16 символов от 0 до 9 и A, B, C, D, E, F. В табл. 6.5 приведены эквиваленты десятичных, двоичных и шестнадцатеричных чисел.

Заметим из табл. 6.5, что каждый шестнадцатеричный символ может быть представлен единственным сочетанием четырех бит. Таким образом, представлением двоичного числа 1001 1110 в шестнадцатеричном коде является число 9E. Эго значит, что часть 1001 двоичного числа равна 9, а часть 1110 равна E (конечно, в шестнадцатеричном коде). Следовательно, $1001\ 1110_2=9E_{16}$. (Не следует забывать, что индексы означают основание системы счисления.)

Как преобразовать двоичное число 111010 в шестнадцатеричное? Надо начать с МБ и разделить двоичное число на группы из 4 бит. Затем надо заменить каждую группу из 4 бит эквивалентной шестнадцатеричной цифрой: 1010_2 = A, 0011_2 = 3, следовательно, $111010_2 = 3A_{16}$.

Как преобразовать шестнадцатеричное число 7F в двоичное? В этом случае каждая шестнадцатеричная цифра должна быть заменена своим двоичным эквивалентом из 4 бит. В примере двоичное число 0111 заменено шестнадцатеричной цифрой 7, а 1111_2 заменяет F_{16}, откуда $7F_{16}= 1111\ 0111_2$.

Таблица 6.6. *Десятичные, шестнадцатеричные и двоичные эквиваленты*

Десятичные	Шестнадцатеричные	Двоичные			
		8	4	2	1
0	0	0	0	0	0
1	1	0	0	0	1
2	2	0	0	1	0
3	3	0	0	1	1
4	4	0	1	0	0
5	5	0	1	0	1
6	6	0	1	1	0
7	7	0	1	1	1
8	8	1	0	0	0
9	9	1	0	0	1
10	A	1	0	1	0
11	B	1	0	1	1
12	C	1	1	0	0
13	D	1	1	0	1
14	E	1	1	1	0
15	F	1	1	1	1

Шестнадцатеричная запись широко используется для представления двоичных чисел, поэтому необходимо табл. 6.7 также запомнить.

Таблица 6.7. *Преобразование шестнадцатеричного числа в десятичное*

Степень шестнадцати	16^8		16^2		16^1		16^0	
Значение позиции	4096		256		16		1	
Шестнадцатеричное число	2 4096 x 2		C 256 x 12		6 16 x 6		E 1 x 14	
Десятичное	8192	+	3072	+	96	+	14	= 11374_{10}

Преобразуем шестнадцатеричное число 2C6E в десятичное. Процедура действий соответствует табл. 2.6. Значениями позиций первых четырех шестнадцатеричных цифр являются соответственно слева направо 4096, 256, 16 и 1, Десятичное число содержит 14 (E_{16}) единиц, 6 чисел 16, 12 (C_{16}) чисел 256 и 2 числа 4096. Каждая цифра умножается на соответствующий ей вес, получается сумма, которая и дает нам десятичное число 11374.

Преобразуем десятичное число 15797 в шестнадцатеричное. На рис.6.6 показана процедура действий. В первой строке 15797_{10} разделено на 16,что дает частное 987_{10} и остаток 5_{10}, который преобразуется затем в свой шестнадцатеричный эквивалент ($5_{10} = 5_{16}$) и становится цифрой младшего разряда (МР) шестнадцатеричного числа. Первое частное (987) становится делимым во второй строке и снова делится на 16, что дает частное 61 и остаток 11_{10} или шестнадцатеричное B. В третьей строке 61 делится на 16, дает частное 3 и остаток 13_{10} или D_{16}, а в четвертой строке делимое 3 делится на 16, дает частное 0 и остаток 3_{10} или 3_{16}. Когда частное равно 0, как в четвертой строке, преобразование заканчивается. 3_{16} становится цифрой старшего разряда (СР) результата, т. е. $3DB5_{16}$.

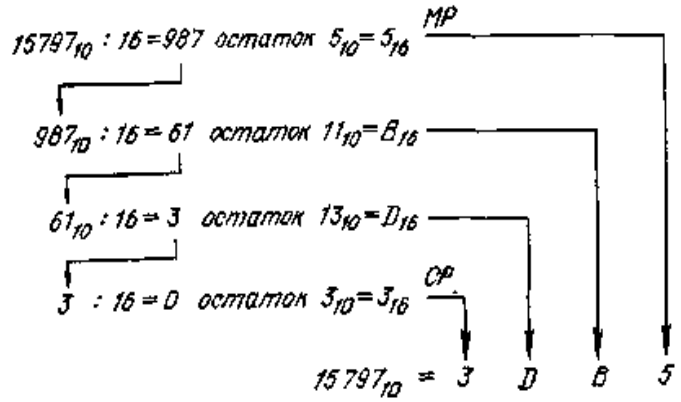


Рис. 6.6. Десятично-шестнадцатеричное преобразование

Восьмеричная система счисления

Восьмеричная запись, как и шестнадцатеричная, используется для представления двоичных чисел. Восьмеричная система содержит 8 цифр от 0 до 7 и является соответственно системой с основанием 8. В табл. 6.8 представлено несколько десятичных, восьмеричных и двоичных чисел.

Преобразуем двоичное число 11111000100 в его восьмеричный эквивалент. Процедура действий в этом случае следующая. Начиная с МБ двоичного числа, делим его на группы из 3 бит. Затем, используя табл. 6.8, преобразуем каждую

триаду (группу из 3 бит) в эквивалентную восьмеричную цифру. Таким образом, мы заменим двоичное число 11111000100 его восьмеричным эквивалентом 3704_8:

Двоичное число 011111000 100

Восьмеричное число 3704

Таблица 6.8.Десятичные, восьмеричные и двоичные эквиваленты

Десятичные	Восьмеричные	Двоичные		
		4	2	1
0	0	0	0	0
1	1	0	0	1
2	2	0	1	0
3	3	0	1	1
4	4	1	0	0
5	5	1	0	1
6	6	1	1	0
7	7	1	1	I

Преобразуем теперь восьмеричное число 6521 в его двоичный эквивалент. Каждая восьмеричная цифра заменяется двоичной триадой и получится, что $6521_8 = 110101010001_2$:

Восьмеричное число 6 5 2 1

Двоичное число 110 101010001

Запишем восьмеричное число 2357 в десятичной форме Классическая процедура выполняется согласно табл. 6.9. Здесь 512, 64, 8 и 1 есть веса четырех первых восьмеричных позиций. Заметим, что в этом примере содержится 7 единиц, 5 восьмерок, 4 числа 64 и два числа 512. Мы их складываем и получаем результат: $1024 + 192 + 40 + 7 = 1263_{10}$.

Таблица 6.9. *Восьмерично-десятичное преобразование*

Степень восьми	8^4		8^2		8^1		8^0	
Значения позицій	512		64		8		1	
Восьмеричное число	2 512 x 2		3 64 x 3		5 8 x 5		7 1 x 7	
Десятичное число	1024	+	192	+	40	+	7	$= 1263_{10}$

Наконец, преобразуем десятичное число 3336 в его восьмеричный эквивалент. Процедура показана на рис. 6.7. В первую очередь 3336 разделено на 8, что дает частное 417 и остаток 0_{10}, причем $0_{10} = 0_8$, восьмеричный 0 становится значением МР восьмеричного числа. Первое частное (417) становится делимым и снова делится на 8 (вторая строка),что дает частное 52 и остаток $1_{10} = 1_8$, который становится второй цифрой восьмеричного числа. В третьей строке частное (52) становится делимым и деление его на 8 дает частное 6 и остаток $4_{10} = 4_8$. В четвертой строке последнее частное 6 разделено на 8 с частным 0 и остатком $6_{10} = 6_8$.

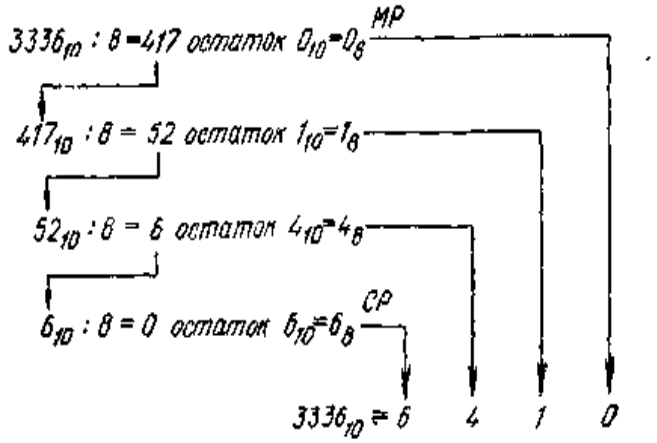


Рис 6.7. Десятично-восьмеричное преобразование

Теперь счет закончен последним частным 0. Цифра 6_8 становится значением СР восьмеричного числа, и мы можем видеть на рис. 5.7, что $3336_{10} = 6410_8$.

Большинство микропроцессоров и микро-ЭВМ обрабатывают группы из 4, 8 или 16 бит. Отсюда следует, что обычно чаще используется шестнадцатеричная запись, чем восьмеричная. Однако восьмеричная запись более удобна, когда группы бит делятся на 3 (например, группы из 12 бит).

Тема 7. Двійкова арифметика і додатковий код.

Двоичная арифметика

Сложение, вычитание или умножение двоичных чисел выполняются так же, как и в арифметике десятичных чисел. Большинство микропроцессоров владеет командами сложения и вычитания двоичных чисел, однако некоторые, менее многочисленные выполняют команды умножения и деления (например, микропроцессоры Intel 8086 и Intel 8088).

На рис. 7.1, а представлены простые правила двоичного сложения. Два первых (слева) правила очевидны, третье показывает, что 1 + 1 = 10, т. е. наиболее значимая 1 пере носится в ближайший старший разряд. Четвертое правило, наконец, показывает, что 1 + 1 + 1 = 11. В этом случае пер вое, второе слагаемые и запоминаемое в результате сложения в младшем разряде число — все 1. Результатом является сумма — 1 с переносом 1.

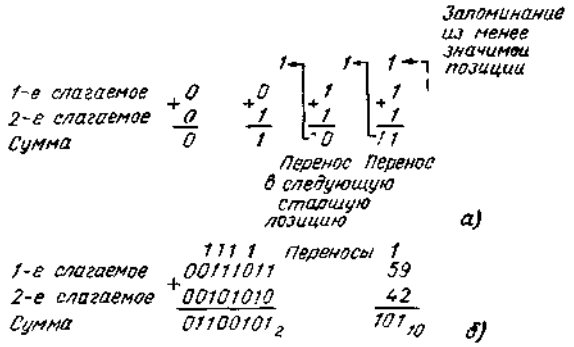


Рис. 7.1. Двоичное сложение: а — правила: б — пример

Сложим двоичные числа 0011 1011 и 0010 1010 (операция показана на рис. 6.1,б). Для большей ясности действия с десятичными эквивалентами обрабатываемых чисел показаны на рисунке справа. Суммой двух чисел 0011 1011 и 0010 1010 будет $0110\ 0101_2$.

На рис. 6.2, а приведены правила двоичного вычитания. Первые три

аналогичны десятичному вычитанию. Последнее требует заема из более значимого предшествующего разряда (в этом случае вес 2). Уменьшаемым является двоичное число 10, вычитаемым 1, разностью—1.

Вычтем двоичное число 0011 1001 из 0101 0101. Этот пример приведен на рис. 7.2, б. Разряды весов 1, 2 и 4 это го двоичного вычитания просты для выполнения и относятся к первым трем правилам на рис. 7.2, а. В колонке веса 8 имеет место вычитание 1 из 0. Тогда 1 занимается из колонки веса 16. Единица вычитается из 10_2, что дает разность 1 согласно четвертому правилу на рис. 7.2, а. После этого заема в колонке веса 16 имеет место вычитание 1 из нового вычитаемого 0. Согласно четвертому правилу 1 должна быть занята из следующей, более значимой позиции (колонка веса 32), но в колонке 32 имеем 0; поэтому колонка 32 должна сделать заем из колонки веса 64, что и вы полнено. Окончательно колонка 16 делает заем из колонки 32, уменьшаемым в колонке 16 становится 10_2, вычитаемым 1, разностью 1. В колонке 32 имеем 1 - 1 = 0, в колонке 64 — 0 - 0 = 0, в колонке 128 — 0 - 0 = 0. Таким образом, рис. 7.2, б иллюстрирует операцию вычитания $0011\ 1001_2$цз $0101\ 0101_2$ (справа эта задача решена в десятичной записи) .

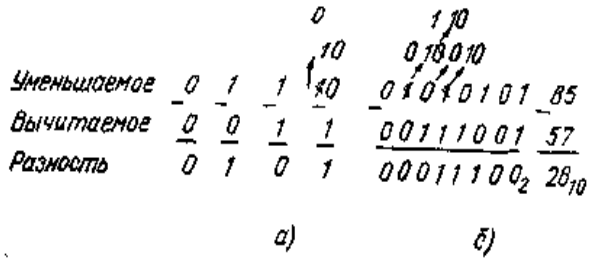


Рис. 7.2. Двоичное вычитание: *а* — правила; *б* — пример

Приведем правила десятичного умножения:

Множимые	0	1	0	1
	×	×	×	×
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 в случае умножения десятичных чисел, множимое сначала умножается на число, стоящее в младшем разряде (в рассматриваемом случае — бит в колонке веса 1).

Множимое	1101	13
	×	×
Множитель	101	5
1-е частичное произведение	1101	65_{10}
2-е частичное произведение	0000	
3-е частичное произведение	1101	
Конечное произведение	1000001_2	

Поскольку бит множителя в разряде веса 1 является 1, множимое копируется и составляет первое частичное про изведение. 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 X 101_2 = 1000001_2 соответствует произведению десятичных чисел $13_{lo}X5_{10}=65_{10}$.

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. 7.3, *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.

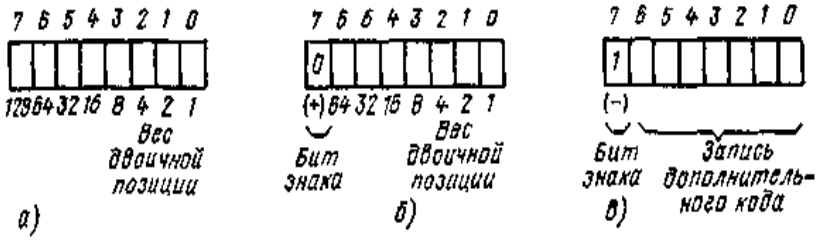


Рис. 7.3. Изображение регистра МП или ячейки памяти:
а — расположение двоичных позиций; б — идентификация положительных чисел нулем в знаковом бите; в — идентификация отрицательных чисел единицей в знаковом бите

На рис.7.3, 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.

Если, как показано на рис. 7.3, б, число положительно, оставшиеся ячейки памяти (6—0) содержат двоичное 7- разрядное число. Например, если регистр на рис. 7.3, 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. 7.3, а, регистр содержит то же число со знаком, но отрицательное, он будет содержать дополнительный код этого числа. В табл. 7.1 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. 7.1: 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.

Таблица 7.1. Десятичные числа со знаком и их представление в дополнительном коде

Десятичные	Представление чисел со знаком	Примечания
+127	0111 1111	
.	.	
.	.	
.	.	
+8	0000 1111	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	Отрицательные числа представлены в форме дополнительного кода
—2	1111 1110	
—3	1111 1101	
—4	1111 1100	
—5	1111 1011	
—6	1111 1010	
—7	1111 1001	
—8	1111 1000	
.	.	
.	.	
.	.	
—128	1000 0000	

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

Десятичное число	9	Этап *1*.	Запись десятичного числа без знака (9)
Двоичное число	0000 1001	Этап *2*.	Преобразование десятичного числа в двоичный код (0000 1001)
Дополнение до 1 (обратный или инверсный код)	1111 0110	Этап *3*.	Получить обратный код двоичного числа заменой нулей единицами, а единиц — нулями (1111 0110)
Дополнение до 2 (дополнительный код)	+1 1111 0111	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 приме ре дополнительным кодом числа 9 является 1111 0111. За метим, 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	Этап *1*.	Запись дополнительного кода (1111 0000).
Дополнение до 1	0000 1111	Этап *2*.	Получается обратный код дополнительного кода заменой нулей единицами, а единиц — нулями (0000 1111)
Двоичное число	+1 0001 0000=16	Этап *3*.	Добавить 1

Таким образом, формирование обратного кода и добавление 1 являются

теми же процедурами, которые мы про водили при преобразовании двоичного числа в дополнительный код. Однако следует отметить, что, хотя мы получили двоичное число 0001 0000 = 16_{10}, исходная запись дополни тельного кода 1111 0000 = -16, т. е. имеем отрицательное число, поскольку старший бит в дополнительном коде является 1.

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. Микро процессор 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: Согласно табл. 2.10 +5 = 0000 0101 в дополнительном коде аналогично +3 = 0000 0011. Тогда числа в дополни тельном коде 0000 0101 и 0000 0011 складываются, как обычные двоичные числа, давая сумму 0000 1000 в дополнительном коде, т. е. 0000 1000 = $+8_{10}$.

1-е число	(+5)	0000 0101
2-е число	(+3)	0000 0011
	(+8)	0000 1000

Пусть надо сложить десятичные числа +7 и -3. Согласно табл. 7.1 +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)	0000 0111
2-е число	(−3)	1111 1101
	(+4)	1 0000 0100

Пренебречь переполнением.

Старший бит является переполнением 8-разрядного регистра, и им можно пренебречь. Получаем сумму 0000 0100 или 4-410.

Сложим десятичные числа +3 и -8. Согласно все той же табл. 7.1 +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)	0000 0011
2-е число	(−8)	1111 1000
	(−5)	1111 1011

Сложим десятичные числа -2 и -5. В дополнительном коде согласно табл. 2.10 -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) 1111 1110

\+ +

2-е число (—5) 1111 1011

(—7) 1 1111 1001

Пренебречь переполнением.

Старший бит результата является переполнением 8-раз рядного регистра, и им пренебрегаем. Таким образом, суммой двух чисел 1111 1110 и 1111 1011 в дополни тельном коде будет 1111 1001. Согласно табл. 2.10 сумма 1111 1001= -$7_{|0}$.

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 0000 0101, что дает 1111 1011. Затем первое число 0000 1000 складывается с дополнительным кодом второго 1111 1011, как с обычным двоичным числом, что дает 1 0000 0011:

1-е число (+8) 0000 1000

\—

2-е число (+5) = 0000 0101 —Дополнительный код→ 1111 1011

(+3) 1 0000 0011

Пренебречь переполнением.

Старший бит является переполнением регистра, им пренебрегаем, что дает результат 0000 0011 = $+3_{10}$- Заметим, что второе число было представлено в дополнительном ко де, 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) 0000 0010

\—

2-е число (+6) = 0000 0110 —Дополнительный код→ \+ 1111 1010

(—4) 1111 1100

Дополнительный код первого числа +2 = 0000 0010, второе число +6 = 0000 0110, его дополнительный код (ин версия 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}.

Тема 8. Машинна мова і Асемблер. Простий склад команд мікропроцесора.

Общие понятия об Ассемблере

Все языки программирования можно подразделить на три группы:проблемно-ориентированные, машинно-ориентированные и машинные.

Машинные языки - это языки цифр. Они занимают самый нижний уровень в иерархии языков программирования и весьма неудобны для программиста при написании и отладке программ.

К проблемно-ориентированным языкам, или языкам высокого уровня, относятся БЭЙСИК, ФОРТРАН, ПАСКАЛЬ, Си и др. Они обладают относительной машинной независимостью, и программа для них пишется,

исходя из существа задачи. Проблемно-ориентированными языками для микроЭВМ являются БЭЙСИК и Си.

Машинно-ориентированный язык мнкроЭВМ называется языком Ассемблера. Поскольку микроЭВМ, производимые различными изготовителями, не похожи друг на друга, отличаются и языки Ассемблера, но незначительно. Свое название языки Ассемблера получили от имени программы транслятора» преобразующей исходную программу (написанную на языке Ассемблера) в объектную (программу на машинном языке» понятную ЭВМ). Каждый язык Ассемблера является машинно-зависимым и отражает аппаратурные особенности (в частности, состав программно* доступных регистров) той микроЭВМ, для которой он создан.

На своем рабочем уровне микропроцессор реагирует на список операций, называемый машинной программой. На рис. 8.1, *а* приведено содержимое памяти, являющееся программой на машинном языке. Эта программа начинается с адреса 2000Н с содержимым КОП 0011 1110_2 и оканчивается адресом 2006Н с содержимым 0111 0110_2. Человеку практически невозможно понять программу, представленную в такой форме.

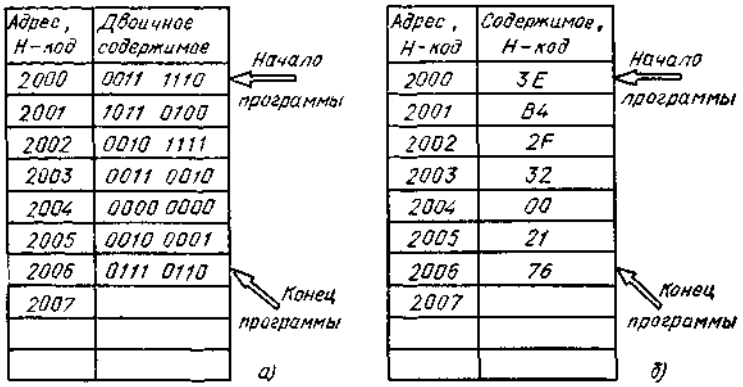


Рис. 8.1. Программы:
а — в двоичном машинном коде; *б* —в шестнадцатеричном машинном коде

Программа на машинном языке на рис. 8.1, *а* становится несколькопроще для восприятия, когда она представлена в шестнадцатеричном коде (Н-коде), как показано на рис. 8.1,б. Однако, хотя двоичные данные приведены в шестнадцатеричном коде, эта часть программы всегда рассматривается как заданная на машинном языке и оказывается трудной для понимания.

В более приемлемой форме записанная на машинном языке она могла бы выглядеть так:

1. Загрузить двоичное число (1011 0100) в аккумулятор.
2. Инвертировать каждый двоичный бит содержимого аккумулятора.
3. Поместить результаты инверсии в ячейку памяти данных 2100Н.

В этой части осуществляется перевод двоичного 8-разрядного числа в его эквивалент в инверсной форме.

Возникает вопрос: как перейти от этой формы человеческого языка, иногда длинной и сложной, к машинному языку? Ответ состоит в использовании языка простого программирования — от самого высокого уровня до машинного, представленного на рис. 8.1.

Ассемблер использует слова и фразы, преобразуя их в машинный код микропроцессора.

Обычно фраза или заданная величина на ассемблере будет соответствовать

выражению длиной от одного до трех байт машинного языка. Суть и процедура ассемблирования показаны на рис. 8.2, где, например, вторая команда программы представлена единственной *мнемоникой* из трех букв CMA (инвертировать содержимое аккумулятора). Сначала три буквы переведены в их эквивалент в коде ASCII, затем три кода ASCII преобразованы в определенный порядок специальной программой ассемблера, которая выдает код инверсии содержимого аккумулятора на машинном языке, т. е. 0010 1111_2 в данном случае или 2FH. Мнемоника преобразована в один единственный байт машинного языка.

Рис. 8.2 Трансляция мнемоники ассемблера в машинный код программы

Программа на языке ассемблер, записанная человеком, могла бы быть представлена в виде табл. 8.1.

Обычным является деление объявлений на машинномязыке на четыре поля: метка; мнемоника; операнд и комментарий. *Поле метки* используется не всегда и в этой программе остается пустым. *Поле мнемоники* содержит точную мнемонику, установленную разработчиком, которая обычно указывает программе ассемблера операцию для выполнения. *Поле операнда* содержит информацию о регистрах, данных и адресах, объединенных соответствующей операцией. Используя информацию только полей мнемоники и операнда, ассемблер может выдать соответствующий код на машинном языке. Можно также назначить ячейки памяти списку команд. Поле комментариев не учитывается ассемблером и ограничивается его перепечаткой. Это поле очень важно, поскольку позволяет понять события в программе.

Таблица 8.1. Программа на языке ассемблер

Метка	Мнемоника	Операнд	Комментарии
	MVl *A,*	B4H	Загрузить в аккумулятор данные, Следующие непосредственно; B4H
	CMA	2100H	Инвертировать содержимое аккумулятора
	STA		Разместить содержимое аккумулятора
	HLT		Остановить МП

Как только программа составлена (см. табл. 8.1), она представляется затем в виде табл. 8.2. Таким образом, задача ассемблирования (или составление программы на ассемблере) состоит из этапов: 1) перевод мнемоники иоперандов на машинный язык; 2) назначение последовательно ячейки памяти каждому КОП и операнду. Переход от версии табл. 6.1 к ассемблированной версии табл. 8.2 может быть выполнен либо вручную, либо на машине при помощи специальной программы ассемблера.

Программа, состоящая из символических команд (фрагмент показан в табл.

8.1), иногда называется *исходной программой,* а переведенная однажды на машинный язык — уже *объектной программой.*

Программирование на языке ассемблер является способом «очеловечивания» действий микропроцессора. Языки высокого уровня (БЕЙСИК, ФОРТРАН и т. д.) при их использовании делают программирование более удобным.

Таблица 8 2. Программа в машинных кодах и на языке ассемблер

Адрес, Н-код	Содержимое, Н-код	Метка	Мнемоника	Операнд	Комментарий
2000	3E		MV1 *A,*	B4H	Загрузить аккумулятор данными, следующими непосредственно за КОП, B4H
2001	B4				
2002	2F		CMA		Инвертировать содержимое аккумулятора
2003	32		STA	2100H	Поместить содержимое аккумулятора в ячейку памяти 2100H
2004	00				
2005	21				
2006	76		HLT		Остановить МП

Система команд микропроцессора

Система команд микропроцессорного устройства служит для связи между микропроцессором, аппаратурой и программным обеспечением и представляет ту часть системы, которая видна программисту.

Все команды процессора можно разделить на несколько основных групп команд. Не смотря на большое число разновидностей МП и МК, на самом низком уровне системы их команд имеют много общего. Любая микропроцессорная система содержит следующие группы команд:

- команды передачи данных (копирования), копирующие информацию из одного места в другое;
- арифметические команды, производящие арифметические преобразования операндов;
- логические команды, позволяющие компьютеру производить анализ получаемой информации. Примерами могут служить сравнение, логические операции И, ИЛИ, НЕ, а так же анализ отдельных битов кода, их сброс и установка. Сдвиги двоичного кода влево и вправо. Операции сдвига используются, например, при выполнении умножения и деления чисел;
- команды ввода и вывода информации для обмена с внешними устройствами;
- команды управления, к которым следует отнести все виды переходов. Сюда же включают операции по управлению процессором.

Команды процессора обычно включают две части – операционную и адресную. Операционная часть (её называют кодом операции – КОП) указывает, какое действие необходимо выполнить с информацией. Адресная часть (адрес) указывает, где хранится используемая в операции информация и куда поместить

результат. У некоторых команд управления работой процессора адресная часть может отсутствовать. Операционная часть команды имеется всегда. По количеству адресов, записываемых в команде, команды делятся на одно-, двух-, трех- и четырехадресные. Существует связь таких параметров процессора, как длина адресного пространства, адресность, разрядность. Увеличение разрядности позволяет увеличить адресность команды и длину адреса (т. е. объем памяти, доступной данной команде). Увеличение адресности, в свою очередь, приводит к повышению быстродействия обработки (за счет снижения числа требуемых команд).

Наибольшее количество адресов было в четырехадресных ЭВМ. В командах принята следующая последовательность указания частей – код операции – КОП, адрес первого операнда – А1, адрес второго операнда – А2, адрес результата – А3 и адрес следующей команды – А4. Такие команды применялись в первых ЭВМ (рис.8.5).

КОП – код операции	А1 – адрес первого операнда	А2 – адрес второго операнда	А3 – адрес результата	А4 – адрес следующей команды

Рис. 8.5. Типовая структура четырехадресной команды

В микропроцессорах большое количество адресов снижает производительность. Поэтому, сначала отказались от использования адреса следующей команды, заменив его счетчиком команд. Затем результат начали помещать по адресу пер-вого операнда. Это привело к настоящему времени к использованию двух-, одно- и безадресных команд (рис 8.6).

а)

КОП – код операции	А1 – адрес первого операнда	А2 – адрес второго операнда

б)

КОП – код операции	А1 – адрес первого операнда

в)

КОП – код операции

а) двухадресная команда; б) одноадресная команда; в) безадресная команда.

Рис 8.6. Типовая структура современных команд

Таким образом, программирование в машинных командах требует знания системы команд конкретной ЭВМ и их адресности. При этом реализация даже довольно несложных вычислений требует разложения их на простые операции, что значительно увеличивает общий объем программы и затрудняет ее чтение и отладку.

Рассмотрим особенности команд используемых в универсальных микропроцессорах и микроконтроллерах.

Например, система команд одного из первых универсальных микропроцессоров i8080 состоит из 78 базовых команд, которые можно разделить на пять групп:

 команды передачи данных — используются для передачи данных из регистра в регистр, из памяти в регистр, из регистра в память;

 арифметические команды — используются для сложения, вычитания, инкре-мента или декремента содержимого регистров или ячейки памяти;

 логические команды: И, ИЛИ, исключающее ИЛИ, сравнение, сдвиги;

 команды переходов — используются для условных и безусловных перехо-дов, вызова подпрограмм и возврата из них;

 команды управления, ввода/вывода и работы со стеком — используются для управления прерыванием, регистром признаков, ввода и вывода информации.

В микропроцессоре принят формат информационного слова, представляющего собой 8-разрядное двоичное слово (байт). Формат информационного слова (данных):

7	6	5	4	3	2	1	0

где D7 — старший разряд слова, D0 — младший разряд.

Формат команды зависит от типа операции и может быть одно-, двух-, или трехбайтовым. Байты двух- и трехбайтовых команд должны храниться в ячейках памяти, следующих одна за другой. Адрес первого байта всегда является адресом кода операции.

Формат команд процессора:

Однобайтовая команда

Байт В1	D7	D6	D5	D4	D3	D2	D1	D0	Код операции

Двухбайтовая команда

Байт В1	D7	D6	D5	D4	D3	D2	D1	D0	Код операции
Байт В2	D7	D6	D5	D4	D3	D2	D1	D0	Данные или адрес

Трехбайтовая команда

Байт В1	D7	D6	D5	D4	D3	D2	D1	D0	Код операции
Байт В2	D7	D6	D5	D4	D3	D2	D1	D0	Данные или адрес
Байт В3	D7	D6	D5	D4	D3	D2	D1	D0	Данные или адрес

Как видно из формата команды универсального процессора, команды имею фиксированный размер кода операции и переменную длину команды.

Тема 9. Команди передавання даних.

Состав команд операций передачи данных

Команды передачи данных выполняют передачу данных из регистра в регистр, размещение данных в памяти, размещение извлеченных из памяти данных в УВВ и установление индикатора переноса (табл. 9.1). Почти все команды передачи не влияют на индикаторы МП. Эта группа содержит множество команд, данные могут быть переданы из любой ячейки памяти в любой регистр или наоборот. Каждая команда передачи содержит адреса источника и назначения данных. Способы адресации ориентированы на то, где и как осуществляется поиск данных.

Рассмотрим первую команду передачи, записанную в табл. 9.1: необходимо их передать из регистра *L*в регистр*A* (аккумулятор). Команда ассемблера (MOV *A, L)*означает перемещение. Следующая за мнемоникой буква (A - в нашем примере) указывает назначение, тогда как последняя (L) идентифицирует источник данных. Это покажется немного несовременно, но эта условность принята фирмой Intel для команд микропроцессоров Intel 8080/ /8085.

Таблица 9.1. *Команды передачи данных типового МП*

Операция	Адресация	Мнемоника	КОП	Байты	Формат команд	Символика
Передать L в A	Регистровая	MOV A, L	7D	1	коп	(A) ←(L)
Передать H в A	»	MOV A, H	7C	1	коп	(A) ← (H)
Передать A в L	»	MOV L, A	6F	1	коп	(L) ← (A)
Передать A в H	»	MOV H, A	67	1	коп	(H) ← (A)
Передать HL в PC	»	PCHL	E9	1	коп	(PC) ←(HL)
Передать HL в SP	»	SPHL	F9	1	коп	(SP) ←(HL)
Загрузить A данными	Непосредственная	MVI A	3E	2	КОП данные	(A) ← (байт 2)
Загрузить L данными	»	MVI L	2E	2	КОП данные	(L) ← (байт 2)
Загрузить H данными	»	MVI H	26	2	КОП данные	(H) ← (байт 2)
Загрузить LOC (HL) в A	Косвенная регистровая	MOV A, M	7E	1	коп	(A) ← ((H)(L))
Загрузить HL данными	Непосредственная	LXI H	21	3	КОП мл.байт ст. байт	(L) ← (байт 2) (H) ← (байт 3)
Загрузить SP данными	»	LXI P	31	3	КОП мл.байтст. байт	(SP) ← (байт 2 + 3)
“ Загрузить HL из LOC *	Прямая	LHLD	2A	3	КОП мл.адрес ст. адрес	(L) ← (байт 2 +3) (H) ← ((байт 2 + 3) + 1)
Загрузить A из LOC	»	LDA	3A	3	КОП мл.адрес ст. адрес	(A) ← ((байт 2 +3))
Поместить A в LOCaa	»	STA	32	3	КОП мл.адрес ст. адрес	(адрес) ← (A)
Поместить HL в LOC	»	SHLD	22	3	КОП мл.адрес ст. адрес	(адрес) ←(L) (адрес + 1) ← (H)
Поместить A в LOCaa(HL)	Косвенная регистровая	MOVM, A	77	1	коп	((H)(L)) ← (A)
Поместить L в LOC (HL)	То же	MOV M, L	75	1	коп	((H)(L)) ← (L)
Поместить H в LOC (HL)	»	MOV M, L	74	1	коп	((H)(L)) ← (H)
Ввести в A из порта в LOCa	Прямая	IN	DB	2	КОП адр. порта	(A) ← (адрес порта)
Вывести A в порт в LOCa	»	OUT	D3	2	КОП адр. порта	(адрес порта) ← (A)
Установить индикатор переноса	Неявная	STC	37	1	коп	(CY) ← 1

Обратимся к примеру на рис.9.1. Источником данных является регистр*A* (аккумулятор), приемником — регистр *L*.Выполненная один раз команда не меняет содержимое аккумулятора.

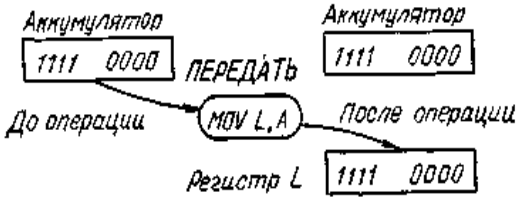


Рис. 9.1. Команда передачи данных из *A*в *L*

На рис. 9.2 приведен другой пример. Здесь источником данных является пара регистров *HL*,приемником — 16- разрядный указатель стека. Если рассмотреть внимательно

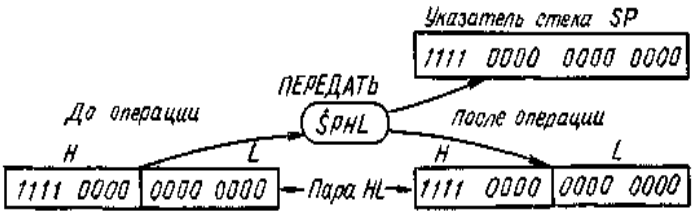


Рис. 9.2. Команда передачи данных из пары регистров *HL*в указатель стека *SP*

мнемонику команды передачи содержимого пары регистров *HL* в указатель стека *SP*, подтвердится, что назначение указывается первым *(SP)*, а источник — последним *(HL)*,что дает нам мнемонику SPHL.

Существует пять команд непосредственной загрузки данных. Эти команды используются очень часто для помещения начального значения в регистр МП в заданный момент, предшествующий программе. На рис. 9.3 приведен пример

такой команды. Пара регистров *HL* емкостью 16 бит должна быть загружена данными, следующими непосредственно за КОП в памяти. Заметим, что согласно табл. 9.1 команда ЗАГРУЗИТЬ *HL* данными является трехбайтовой командой. Первый байт — это КОП (21Н в нашем примере), тогда как два последующих байта являются байтами данных. Второй байт содержит МБ данных(в штриховой рамке), который служит указателем адреса памяти (LOC) и загружается в регистр *L*. *Содержимое* следующего байта памяти [адрес (LOC + 1) в нашем примере] загружается в регистр *H*.

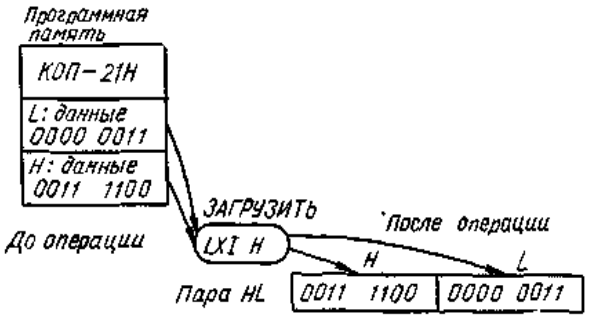


Рис. 9.3 Команда загрузки пары регистров *HL*с непосредственной адресацией

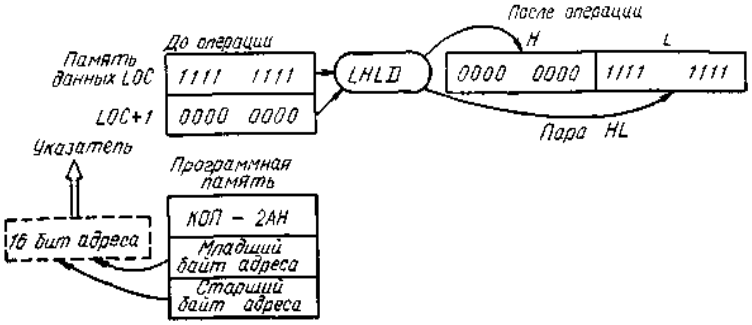


Рис 9.4. Команда загрузки пары регистров *HL*с прямой адресацией

Типовой МП снабжен пятью командами размещения, которые пояснены в табл. 9.1. Рассмотрим пример использования команды прямого размещения содержимого *A,* приведенный на рис. 9.5. Содержимое аккумулятора (регистр*A*) помещается в память (LOC), на которую указывает 16-разрядный адрес, составленный вторым и третьим байтом команды. После выполнения ячейка памяти (LOC) и аккумулятор содержат те же данные, т. е. 1100 0000.

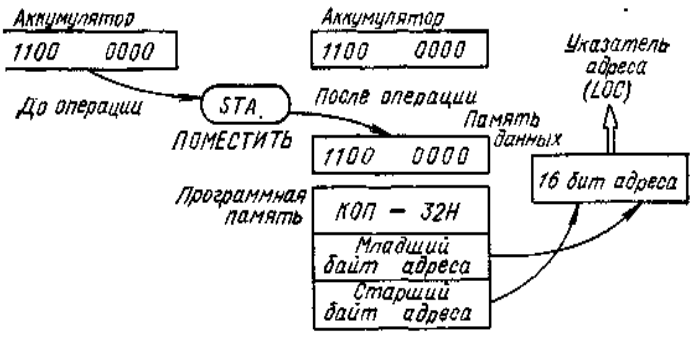


Рис. 9.5. Команда загрузки данных аккумулятора в память с прямой адресацией

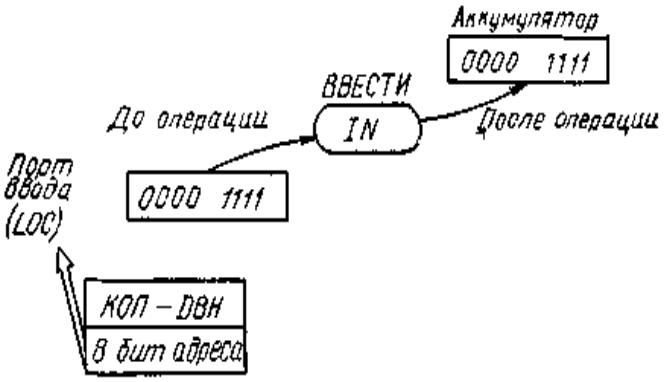


Рис. 9.6. Команда ввода с прямой адресацией

Команда ввода (третья строка снизу в табл. 9.1) идентична команде загрузки. Источник данных передачи является портом ввода, идентифицированным двоичным 8-разрядным числом ($0—255_{10}$), назначение этих данных — аккумулятор МП. Пример выполнения команды представленна рис. 9.6. Данные порта ввода 0000 1111, на который указывает второй байт команды, передаются в аккумулятор, исходя из порта ввода, идентифицированного LOC.

Таким образом, команды передачи данных предписывают микропроцессору пересылку данных из одного блока в другой, например, из одного регистра в другой. Эти команды всегда определяют источник и приемник данных, причем в команде сначала следует адрес приемника, а потом адрес источника.

Содержимое источника при этом не изменяется.

Команда MOV

MOV A,B; B→A

MOV A,M; M→A,

M адрес ячейки памяти, который предварительно надо поместить в H-пару.

Команда MVI

Команда MVI отличается от команды MOV тем, что в качестве источника данных используется восьмиразрядная константа, которая следует непосредственно за кодом операции. Приемником данных является регистр r или ячейка памяти M.

MV1 C,data8; data8→C

Команда LXI

Команда LXI может быть использована для загрузки регистровых пар (B-пары, D-пары, H-пары) и указателя стека SP шестнадцатиразрядным числом, которое непосредственно следует за кодом операции.

LXI D,data16; data16→(D,E)

Команды LDA, STA

По команде LDA в аккумулятор загружается содержимое ячейки памяти, адрес которой следует за кодом операции. По команде STA содержимое аккумулятора записывается в ячейку памяти, адрес которой следует за кодом операции.

LDA addr16; addr16 →A

STA addr16; A→addr16

Команды LDA и STA являются трехбайтными. В первом байте содержится код операции, во втором байте < B2 > - младший байт адреса (addr мл), а в третьем байте < B3 > - старший байт адреса (addr ст).

Команда LDAX.

По команде LDAX в аккумулятор загружается содержимое ячейки памяти, адресуемой регистровой парой (B,C) или (D,E).

Команда STAX осуществляет передачу содержимого аккумулятора в ячейку памяти, адресуемой регистровой парой (B,C) или (D,E).

LDAX B; косвенная адресация. Предварительно в B-пару загружается адрес ячейки, содержимое которой загружается в аккумулятор. Операции с портом.

Команды IN и OUT управляют обменом информации между аккумулятором A и портами ввода-вывода. В команде IN (ввод) источником является порт ввода port, а приемником - аккумулятор A. В команде OUT (вывод) источником является аккумулятор A, а приемником порт вывода port. Адрес порта непосредственно следует за кодом операции IN или OUT.

IN addrport;

OUT addrport;

Работа со стеком.

Команда LXI

LXI SP,data16; data16 → SP. С этого момента известно, где в памяти находится стек.

Команды пересылок PUSH (Поместить в стек) и POP (Вытолкнуть из стека) всегда оперирует с регистровой парой (B,C), (D,E), (H,L) или парой регистров (A,РП), образующей слово состояния программы PSW. Напомним, что указатель стека SP содержит адрес той ячейки в стеке, в которую в последний раз была записана информация, т.е адрес верха стека. По команде PUSH (Поместить в стек) выполняются действия: а) содержимое SP сначала уменьшается на 1 (декрементируется); б) старший байт загружается в стек; в) содержимое SP вновь декрементируется; г) младший байт загружается в стек. PUSH B; Записать в стек B-пары. По команде POP (Вытолкнуть из стека) выполняются обратные действия. Сначала младший байт выталкивается из стека в МП и содержимое SP увеличивается на 1 (инкрементируется). Затем старший байт выталкивается из стека в МП и содержимое SP инкрементируется. POP B; вытолкнуть из стека в B-пару.

Использование регистра адреса/данных

Использование пары регистров *HL* в качестве указателя адреса является интересным свойством типового МП. Обычно рассматривают ее использование в качестве указателя адреса, когда она временно берет на себя роль основного счетчика команд, указывая адрес ячейки памяти или УВВ. Многие широко распространенные МП (например, Intel 8080/8085, Z 80) содержат регистры такого типа. Регистры адреса/данных в рассматриваемом типовом МП называются также парой *HL*.-регистров, регистром адреса, счетчиком данных или указателем адреса.

Рассмотрим простую задачу сложения содержимого трех последовательных ячеек памяти с размещением суммы в следующей ячейке памяти (выполнение ее показано на рис. 9.7).

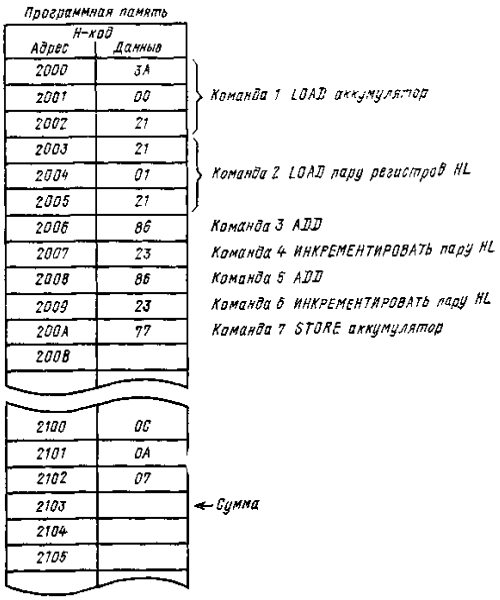


Рис. 9.7. Воображаемая память и команды в примере сложения

Программа загружена в ячейки памяти 2000Н — 200АН, а три слагаемых (0СН + 0АН + 07Н) — в ячейки памяти 2100Н—2102Н. Программа содержит 6 команд, записанных справа на рис. 9.7. Не следует забывать, что текущая сумма будет всегда помещаться в аккумулятор, содержащий вначале первое слагаемое (0СН).

Команда *1* имеет КОП 3АН (рис. 9.7) и приказывает МП ЗАГРУЗИТЬ (LOAD) в аккумулятор содержимое ячейки памяти 2100Н. Выполнение этой команды прямой загрузки аккумулятора показано на рис. 9.8, *а.* После выполнения команды аккумулятор будет содержать первое слагаемое (0СН).

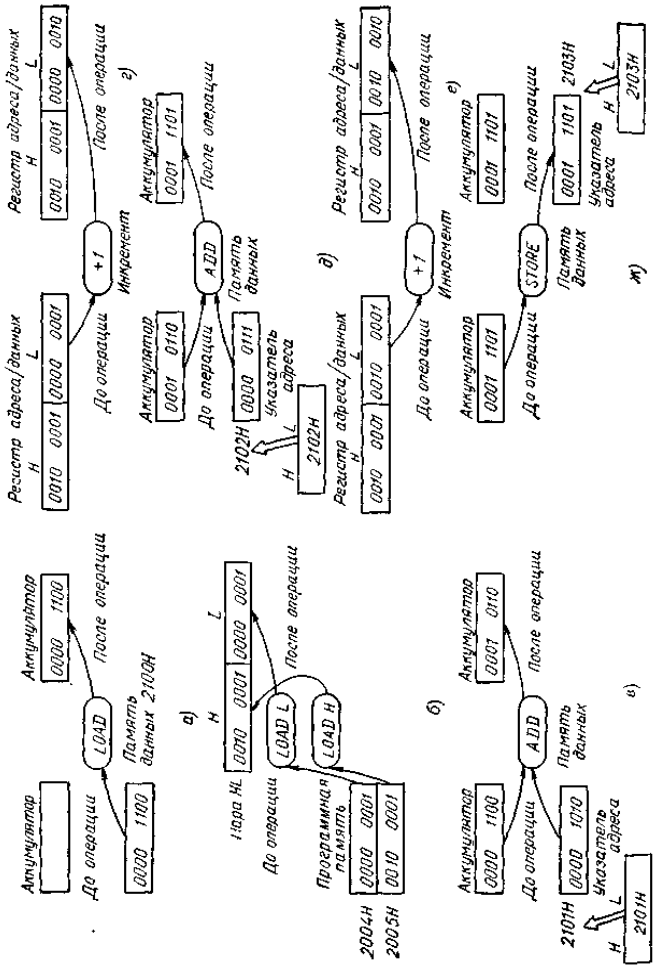


Рис. 9.8. Выполнение операций в соответствии с программой рис. 9.7:
а — команда 1 загрузки аккумулятора; б — команда 2 загрузки пары регистровHL\ в — команда 3 сложения, г — команда 4 инкремента пары регистров; д —команда 5 сложения; е — команда 6 инкремента пары регистров HL\ ж — команда 7 размещения в памяти содержимого аккумулятора

Команда *2* приказывает МП загрузить (LOAD) 2101H в пару регистров *HL*,емкость которых 16 бит. Это число представляет собой адрес памяти данных. Более точно команду *2* можно сформулировать так: загрузить пару регистров *HL*непосредственно следующими за КОП данными, ее выполнение приведено на рис. 9.8, *б.* Заметим, что содержимое первой ячейки памяти (2004H) загружается в младший байт L, следующей за ней — в старший байт *H* пары регистров *HL*.

Команда *3* приказывает МП выполнить операцию сложить (ADD) содержимое аккумулятора с содержимым ячейки памяти, адрес которой содержится в паре регистров *HL*. Ее выполнение приведено на рис. 9.8, *в* (команда ADD). Пара регистров *HL*указывает на ячейку памяти 2101H, и АЛУ складывает свое содержимое ($0000\ 1010_2$) с содержимым аккумулятора ($0000\ 1100_2$), что дает сумму ($0001\ 0110_2$), помещаемую в аккумулятор.

Команда *4* приказывает МП инкрементировать (увеличить на 1) содержимое пары регистров *HL*(см. рис. 9.8, *г).* Заметим, что изменился только младший байт пары регистров *HL*.

Команда *5* снова приказывает МП сложить (ADD) содержимое аккумулятора и ячейки памяти с адресом 2102H, на которую указывает пара регистров *HL*(см. рис. 9.8,*б*) .Оба содержимых складываются, что дает сумму ($0001\ 1101_2$), помещаемую в аккумулятор.

По команде *6* содержимое пары регистров *HL*снова инкрементируется (см. рис. 9.8, *е).*

Команда *7* приказывает МП поместить (STORE) содержимое аккумулятора, т. е. окончательную сумму ($0001\ 1101_2$) в ячейку памяти, на которую указывает пара регистров *HL* (см. рис. 9.8, *ж), т.е.* по адресу 2103H.

Команды *3, 5, 7,* взаимодействующие с парой регистров *HL*как с указателем адреса, используют косвенно-регистровый способ адресации. Его мы изучим в следующей главе.

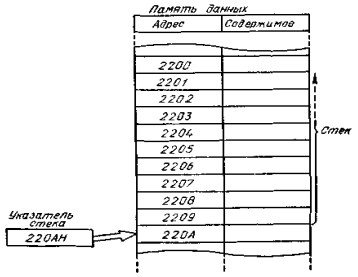
Использование указателя стека

Типовый микропроцессор содержит указатель стека — специализированный 16-разрядный регистр-счетчик, содержимым которого всегда является адрес. Этот адрес принадлежит особой группе ячеек памяти данных, которая называется *стеком.* В некоторых МП стек может быть составлен из группы физически локализованных на кристалле МП ячеек памяти. Когда микропроцессором выполнялась подпрограмма обслуживания прерывания, текущие данные во всех регистрах МП должны были временно сохраняться. Эта сохранность обеспечена стеком. А когда подпрограмма полностью выполнена, содержимое счетчика команд должно быть сохранено таким образом, чтобы МП мог возвратиться в соответствующее место в программной памяти. Зона временной памяти является стеком. Напомним, что подпрограмма является короткой, часто используемой, специализированной программой (например, умножить).

Стек типового микропроцессора будет содержаться в ОЗУ, и его положение определяется программистом. Указатель стека загружается старшим

адресом, представляющим собой вершину стека (рис. 9.9). В этом случае указательстека содержит 220АН, что на единицу старше первой ячейки памяти стека 2209Н.

Данные можно записать в стек, используя команды PUSH (поместить) или CALL (вызвать). Они могут быть считаны из стека по командам POP (извлечь) или RETURN (возврат). Стек функционирует как память с последовательным доступом по типу: данные, поступившие последними, извлекаются первыми (тип LIFO от *Last In — First Out* — последний входит — первый выходит, или FILO от *First In — Last Out* —первый входит — последний выходит).



Рнс. 9.9. Расположение стека в ОЗУ

Команда загрузки в стек (PUSH) приводит к результату, показанному на рис. 9.10, *а.* Содержимое пары регистров *HL*помещается в стек. Отметим, что двухбайтовая пararегистров *HL*должна быть размещена в двух ячейках памяти стека. Последовательность событий может быть описана в соответствии с номерами, показанными в кружках на рис. 9.10.

1. Указатель стека МП декрементируется от 220АН до 2209Н.
2. Указатель стека показывает на ячейку памяти 2209Н по адресной шине и старший байт ($0000\ 0000_2$) помещается в стек.
3. Указатель стека снова декрементируется от 2209Н до 2208Н.
4. Указатель стека указывает на ячейку памяти 2208Н (по адресной шине системы) и младший байт из регистра данных ($0000\ 1111_2$) загружается в стек.

На рис. 9.10, *б* показано выполнение другой операции загрузки. На этот раз в стек загружается содержимое аккумулятора и регистра состояния. Проследим снова за событиями, отмеченными цифрами в кружках.

5. До операции указатель стека указывает на последнюю ячейку стека. Ее называют вершиной стека. Затем указатель стека декрементируется до 2207Н.
6. Указатель стека указывает на ячейку памяти 2207Н, и содержимое аккумулятора ($0101\ 0101_2$) загружается в стек по этому адресу.
7. Указатель стека декрементируется от 2207Н до 2206Н.
8. Указатель стека указывает на ячейку памяти 2206Н. Содержимое регистра состояния ($1111\ 1111_2$) загружается по этому адресу.

Стек может продолжать расти, пока длится процесс загрузки в него (на рис. 9.9 показано, что стек растет вверх).Стек не имеет ограничений, за исключением тех, которые обусловлены наличием других программ в ОЗУ.

Обычно каждой команде загрузки в стек (PUSH) позже будет соответствовать команда извлечения из стека (POP), по которой данные берутся из вершины стека. Поскольку стек является памятью типа LIFO (последний вошел — первый вышел), данные должны извлекаться из стека в порядке, обратном загрузке. На рис. 9.10, *в* и *г* подробно показаны операции извлечения из стека.

Рассмотрим команду POP на рис. 9.10, *в.* Аккумулятор и регистр состояния свободны до операции извлечения из стека. Следуем последовательности, указанной цифрами в кружках.

1. Указатель стека указывает на вершину стека, т. е. на адрес 2206H. Содержимое регистра состояния ($1111\ 1111_2$) извлечено из стека и переслано в АЛУ.
2. Указатель стека инкрементирован с 2206H до 2207H.
3. Указатель стека указывает на адрес 2007H стека. Вершина стека извлекается, и ее содержимое пересылается в аккумулятор АЛУ.
4. Указатель стека инкрементирован до 2208H и указывает теперь на следующий адрес извлечения из стека.

Содержимое аккумулятора и регистра состояния было восстановлено до тех значений, которые были до операции PUSH, показанной на рис. 9.10, *б.*

Затем (на рис. 9.10, *г)* содержимое регистра адреса/данных в свою очередь извлекается из стека. Снова последуем согласно заключенным в кружки цифрам:

5. Указатель стека указывает на вершину стека (адрес 2208H). Содержимое этой ячейки памяти стека извлекается и пересылается в младший байт пары регистров *HL.*
6. Указатель стека инкрементируется до 2209H.

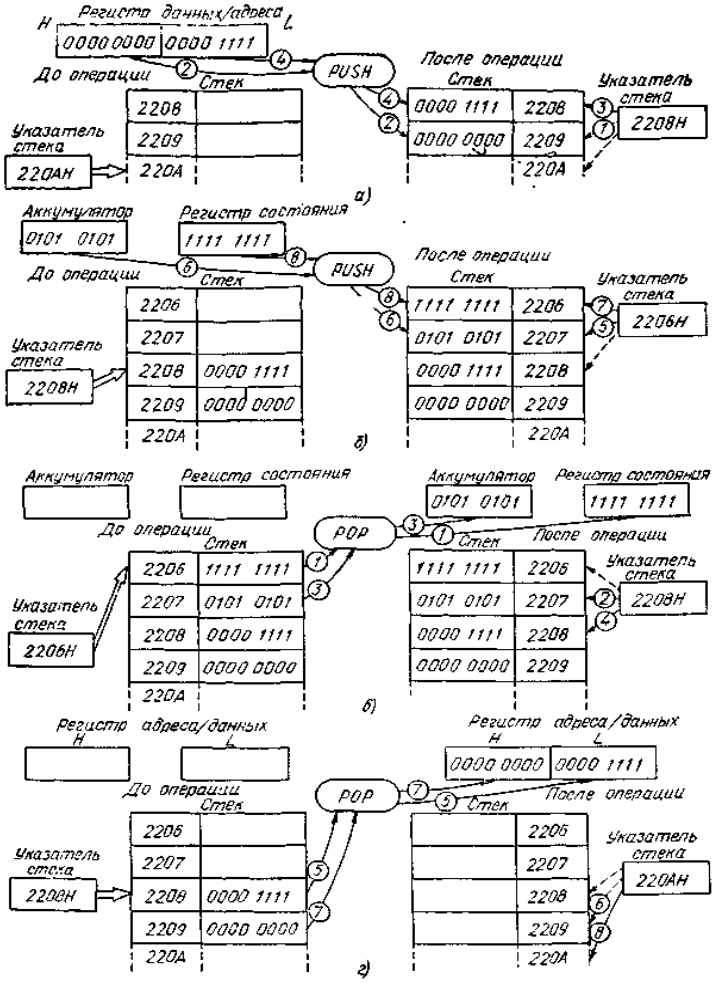


Рис. 9.10. Операции со стеком

а — помещение в стек содержимого регистра адреса/данных; б — помещение в стек содержимого аккумулятора и регистра состояния; в — извлечение из стека содержимого аккумулятора н регистра состояния; г ~ извлечение из стека содержимого регистра адреса/даннях

7. Указатель стека показывает на вершину стека, т. е. теперь адрес 2209H, содержимое которого (0000 0000_2), передается в старший байт пары регистров *HL*.
8. Указатель стека инкрементируется от 2209H до 220AH для последующей операции загрузки в стек (PUSH) или извлечения из стека (POP).

Извлечение данных из стека и их восстановление в регистре адреса/данных является действием, обратным операции загрузки в стек (PUSH), выполненной на рис. 5.15, *а.* Команды PUSH и POP используются всегда совместно, однако между ними располагаются другие команды, которые меняют данные, содержащиеся в регистрах МП.

Наш типовой МП загружает в стек и извлекает содержимое пары регистров. Некоторые МП осуществляют эти операции только с однобайтовыми регистрами, единственной командой. В других МП указатель стека может указывать на пустую ячейку памяти по ближайшему большему по отношению к вершине стека адресу вместо указания на вершину стека, как в предыдущем случае. При этих условиях имеется возможность сохранить в памяти то, что программист загрузил в начале (адрес 220AH в нашем случае) в указатель стека для определения его адреса.

Рассмотрим простейшие примеры программ пересылки данных.

Пример 1. Переслать данные, занимающие 2 байта, из памяти в регистровую пару *DE*.

Программа:

```
Адрес  Ассемблерный код      Комментарий
00      LXI   H, ADR    ; Установить непосредственный
01                      ; адрес, по которому хранится
02                      ; 2-й байт данных, в
                        ; регистровую     пару HL
03     MOV E ,M         ; Переслать в E содержимое
                        ; ячейки памяти, адресуе- мой парой HL
04      INX   H         ; Увеличить на 1 адрес, находящийся в HL
05     M0V   D ,M       ; Переслать в D содержимое
                          ячейки памяти, адрес ко- торой находится в HL
06      END             ; Конец.
```

Пример 2 . Переслать данные, занимающие 2 байта, из регистровой пары *DE* в память.

Программа:

```
Адрес  Ассемблерный код   Комментарий
00      LKI   H, ADR    ; Установить непосредст-
                          венный
01                      ;адрес      в       регистровую
02                      ; пару HL
03      M0V  M, E       ; Переслать содержимое  E
                          в ячейку памяти с адресом, находящимся в HL
04        INX   H       ; Увеличить на 1 адрес в HL
05      MOV  M, D       ; Переслать содержимое D в
                            ячейку памяти, адресуе- мую парой HL
06         END          ; Конец.
```

Пример 3. Переслать данные длиной 2 байта из памяти в пару *HL*.

Программа:

Адрес	Ассемблерный код	Комментарий
00	*LXI H, ADR*	; Установить непосредственный
01		; адрес в регистровую
02		; пару *DE*
03	*LDAX D*	; Загрузить *A* содержимым ячейки памяти с адресом, находящимся в *DE*
04	*MOV L, A*	; Переслать содержимое *A* в *L*
05	*INX D*	; Увеличить на 1 адрес, содержащийся в *DE*
06	*LDAX D*	; Загрузить *A* содержимым ячейки памяти, адресуемой парой *DE*
07	*MOV H, A*	; Переслать содержимое *A* в *H*
08	*E ND*	; Конец.

Пример 4. Переслать данные длиной 2 бай та из пары *HL* в память.

Программа:

Адрес	Ассемблерный код	Комментарий
00	*LXI D, ADR*	; Установить непосредствен- 01 ; ный адрес в регистровую
02		; пару *DE*
03	*MOV A, L*	; Переслать содержимое регистра *L* в *A*
04	*STAX D*	; Записать в ячейку памяти, адресуемую парой *DE*, содержимое *A*
05	*INX D*	; Увеличить на 1 адрес в *DE*
06	*MOV A, H*	; Переслать содержимое регистра *H* в *A*
07	*STAX D*	; Записать в ячейку памяти, адресуемую парой *DE*, содержимое *A*
08	*END*	; Конец.

Тема 10. Склад команд арифметичних дій.

Команды сложения

Первыми рассмотрим команды арифметических действий. Они сведены в табл. 9.1 и содержат команды сложить, вычесть, инкрементировать, декрементировать, сравнить. Заметим по данным табл. 10.1, что существуют четыре команды сложения. Аккумулятор (регистр А) содержит одно из слагаемых. Каждая команда точно оговаривает различные источники другого слагаемого.

Рассмотрим первую из команд сложения в табл. 10.1. Команда сложить непосредственно является двухбайтовой. Ее формат КОП (в этом случае С6Н) содержится в первом байте команды, непосредственно за ним — во второмбайте — находятся данные для сложения с содержимым аккумулятора. Команда ADI выполняется по схеме, приведенной на рис. 10.1, *а.* Данные, находящиеся в памяти непосредственно за КОП, складываются с содержимым аккумулятора ($0000\ 1111_2$). Сумма ($0001\ 1111_2$) помещается в аккумулятор.

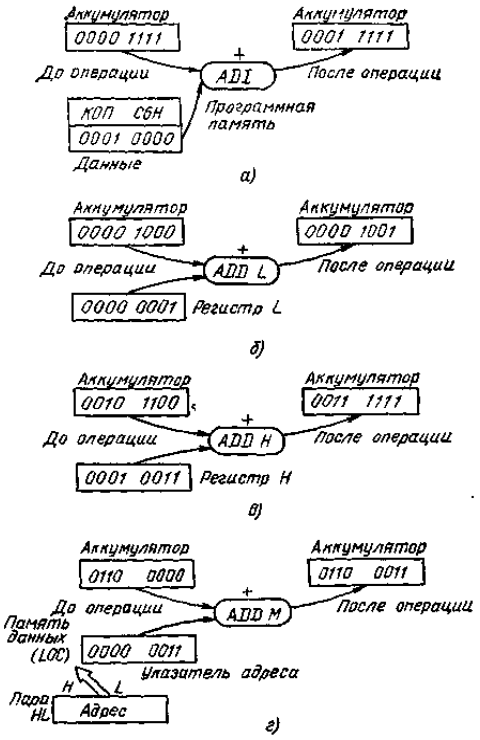


Рис. 10.1. Интерпретация операции ADD *М в* кодах табл. 10.1:
а—сложение (А) с данными, следующими непосредственно за КОП; *б* — сложение (L) и (Д); *в* — сложение (Н) и (А); *г* —сложение с косвенной адресацией

Таблица 10.1, Команды арифметических операций типового МП

Операция	Адресация	Мнемоника	К.ОП	Байты	формат команды	Символика	Индикаторы
СложитьА с данными	Непосредственная	AD1	С6	2	КОП данные	(А) ← (А) + (байт 2)	Z, CY
Сложить LcA	Регистровая	ADD L	85	1	КОП	(А) ← (А) + (L)	Z, CY
Сложить Н с А	»	ADD H	84	1	коп	(А) ← (А) + (Н)	Z, CY
Сложить LOC (HL) cA	Косвенная регистровая	ADD M	86	1	коп	(А) ← (А) + ((Н)(L))	Z, CY
Вычесть данные изА	Непосредственная	SUI	6	2	КОП данные	(А) ← (А) - (байт 2)	Z, CY
Вычесть LизА	Регистровая	SUB L	95	1	коп	(А) ← (А) - (L)	Z, CY
Вычесть Н изА	Регистровая	SUB H	94	1	коп	(А) ← (А) - (Н)	Z, CY
Вычесть LOC (HL) изА	Косвенная регистровая	SUB M	96	1	коп	(А) ← (А) - ((Н)(L))	Z, CY
ИнкрементА	Регистровая	INR A	3С	1	КОП	(А) ← (А) + 1	Z
Инкремент HL	»	INX H	23	1	КОП	(HL) ← (HL) + 1	
ДекрементА	»	DCR A	3D	1	КОП	(А) ← (А) - 1	Z
Декремент HL	»	DCX H	2B	1	КОП	(HL) ← (HL) - 1	
СравнитьА с данными	Непосредственная	CPI	FE	2	КОП данные	(А) ← (байт 2)	Z= 1, если (А) = (байт 2); CY =1, если (А) < (байт 2)
СравнитьА с L	| Регистровая	CMP L	BD	1	коп	(А) ← (L)	Z = 1, если (А) = (L); CY = 1, если (А) < (L)
СравнитьА с Н	»	CMP H		1	КОП	(А) ← (Н)	Z = 1, если (А) = (Н); CY= 1, если (А) < (Н)
СравнитьА cLOC (HL)	Косвенная регистровая	CMP M		1	коп	(А) ← ((Н)(L))	Z = 1, если (А)=((Н)(L)); CY=1,если (А)<((Н)(L))

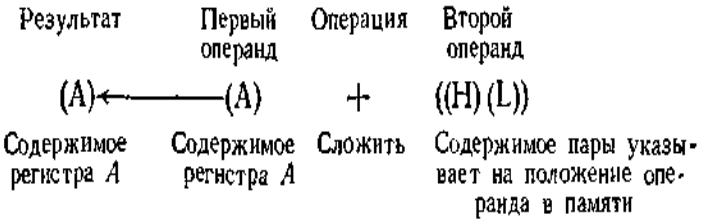
Второй является команда сложить содержимое регистров *L*и *A* (мнемоника ADD L), на рис. 10.1, *б* показано ее выполнение. Содержимое аккумулятора (0000 1000_2) складывается с содержимым регистра *L*(0000 0001_2), получающаяся в результате выполнения команды ADD *L*сумма (0000 1001_2) помещается в аккумулятор.

Третья команда в табл. 10.1 представляет собой однобайтовую команду сложить содержимое регистров *Н* и*А* (мнемоника ADD *Н).* Это другая команда сложения содержимого одного регистра с содержимым другого, и она выполняется в последовательности, приведенной на рис. 10.1, *в.* Содержимое регистра *А* (0010 1100_2) сложено с содержимым регистра *Н* (00010011_2), сумма (0011 1111_2) помещена в аккумулятор.

Четвертая строка таблицы представляет собой однобайтовую команду сложить с косвенным регистром (мнемоника ADD *М).* Адрес данного слагаемого числа задан в более сложной форме с использованием способа косвенной регистровой адресации. Рисунок 10.1, *г* является примером выполнения команды ADD *М.* Пара *HL*указывает 16-разрядный адрес памяти, т. е. LOC. Содержимое LOC (0000 0011_2) сложено с содержимым аккумулятора (0110 0000_2), сумма (0110 0011_2) помещена в аккумулятор. Команды косвенного сложения используют в качестве указателя адреса 16-разрядный регистр (обычно пару *HL).*

Рассмотрим снова команду сложить с косвенным регистром (ADD М) в табл. 10.1. Предписание символики указывает: сложить LOC (Н + L) с*А,* которое мы прочтем как сложить содержимое памяти, на которую указывает пара регистров *HL,*с содержимым регистра *А.* Следуя соответствующей строке в табл. 10.1, мы найдем, что способом адресации является косвенная регистровая и соответствующей мнемоникой будет ADD *М* с КОП 86Н. Эта команда является однобайтовой. Представленный формат ее показывает, что единственным байтом является КОП. Следуя предписанию, данному фирмой Intel (и, очевидно, обратному обычной записи), символическую запись читают справа налево.

Справа стрелки первый операнд идентифицирован как содержимое *A* (акумулятора). Скобки в рамках этой записи означают содержимое. Знак «+» означает выполняемую операцию, а двойная скобка (()) указывает команду косвенной адресации. Выражение((H)(L)) означает: содержимое пары *HL*указывает адрес расположения в памяти второго операнда. Иначе говоря, второй операнд расположен в ячейке памяти, на которую указывает пара регистров *HL*.После того как операция выполнена, ее результат помещается в аккумулятор. Все действия выполняются по такой схеме:



В последней правой колонке табл. 10.1 приведен список устанавливаемых по результатам операции индикаторов. Пример сложения двоичных чисел 1111 1111 и 0000 0001 приведен на рис. 10.2. Выполнение такой операции обычным способом дает следующий результат:

Решение этой задачи микропроцессор выполняет в соответствии с рис.10.2 (операция ADD *M.*

Аккумулятор содержит 1111 1111_2, тогда как ячейка памяти, на которую указывает пара регистров *HL,*содержит другое слагаемое (0000 0001_2). Выполнив операцию сложения, аккумулятор содержит 8 младших бит суммы,т. е. 0000 0000_2. Индикатор переноса регистра состояния установлен в 1, что показывает наличие переноса старшего бита результата. Индикатор нуля проверяет содержимое аккумулятора после операции и находит 0000 0000_2. Так как это нуль, индикатор нуля становится 1, показывая, что содержимое аккумулятора 0000 0000_2 после операции сложения.

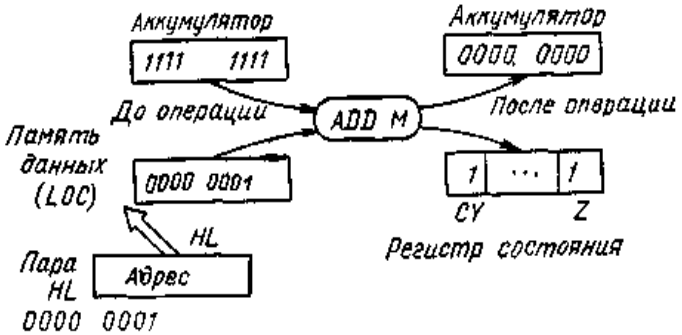


Рис. 10.2. Влияние результата операции ADD *M* на содержимое аккумулятора и индикаторы

Пример 1. Сложить два числа 10100011 и 00011110.

Далее приводится два варианта решения задачи. 1-й вариант составления

программы:

Адрес Ассемблерный код Комментарий

00 *MVI A 10100011 B* ;Пересылка непосредственных
01 данных в *A*
02 *ADI A 00011110 B* ; Сложение непосредственных
03 данных с содержимым *A*
04 *END* ; Конец.

2-й вариант составления программы:

Адрес Ассемблерный код Комментарий

00 *MVI B 10100011 B* ;Переслать непосредственные
01 данные в регистр *B*
02 *MOV A, B* ; Переслать данные из *B* в *A*
03 *MVI B 10100011 B* ;Переслать непосредственные
04 данные в регистр *B*
05 *ADD B* ;Сложить содержимое A и *B*
06 *END* ;Конец.

Первый вариант по сравнение со вторым более предпочтителен, так как программа в первом случае занимает меньший объем памяти.

Пример 2 Сложить два числа 10100011 и 00011110 и результат записать в ячейку памяти 1101.

Программа:

Адрес Ассемблерный код Комментарий

00 *MVI B 10100011 B* ;Переслать непосредственные
01 данные в регистр B
02 *MOV A, B* ;Переслать содержимое *B* в *A* 03 *ADI B 10100011*
B ; Сложить непосредственные 04 данные с
содержимым *A*
05 *IXI B 00000000* ;Загрузить пару регистров *BC*
06 *00001101B* ;непосредственными данными
07
08 *STAX B* ; Записать содержимое *A* в ячейку памяти,
адресуемую содержимым регистром *BC*
09 *END* ; Конец.

Пример 3 Сложить два числа 10100011 и 00011110 и результат записать по адресу, находящемуся в двойном регистре *HL* .

Программа:

Адрес Ассемблерный код Комментарий

00 *MVI 10100011 B*;Пересылка непосредственных
01 данных в *A*
02 *ADI A 00011110 B* ;Сложение непосредственных
03 данных с содержимым *A*
04 *HCHG* ; Обмен содержимым регистров *H* и *D* , *E* и *L*.
05 *STAX D* ; Запись содержимого *A* в
ячейку памяти, адресуемую содержимым регистров
D и E
06 *END* ; Конец.

Пример 4 Сложить содержимое ячеек 40*H* и 41*H*. Результат поместить в ячейку 42*H*.

Программа

Адрес	Ассемблерный код	Комментарий
00	*LDA 40H*	; Загрузка *A* содержимым
01		; ячейки *40H*
02		;
03	*MOV B, A*	; Пересылка в регистр *B* содержимого *A*
04	*LDA 41H*	; Загрузка *A* содержимым
05		; ячейки *41H*
06		
07	*ADD B*	; Сложение содержимого *B* с содержимым *A*
08	*STA 42H*	; Пересылка содержимого *A* в
09		; ячейку *42 H*
0A		;
0B	*END*	; Конец.

Пример 5 Сложить число *IEH* с числами, находящимися в ячейках памяти с адресами 2000 *H* и 2005 *H* .

Программа:

Адрес	Ассемблерный код	Комментарий
00	*LXI H, 2000H*	; Установить адрес 2000*H* 01 ; в *HL*
02		;
03	*MOV A, M*	;Переслать содержимое ячейки памяти, адрес которой находится в *HL* , в *A*
04	*ADI IEH*	;Сложить число, содержащееся
05		; в *A*, с числом *IEH*
06	*LXI H,2005H*	; Загрузить адрес 2005 *H*
07		; в *HL*
08		;
09	*MOV B, M*	; Переслать число из ячейки памяти с адресом, находящимся в *HL* , в *B* .
0A	*ADD B*	; Сложить содержимое *A* с числом, находящимся в *B*
0B	*LXI H REZ*	; Установить в *HL* адрес,
0C		; определяемый меткой *REZ.*
0D		;
0E	*MOV M, A*	; Переслать результат из *A* в ячейку памяти с адресом в *HL*
0F	*REZ: DS OIH*	; Зарезервировать 1 байт памяти для хранения результата
10	*END*	; Конец.

Команды вычитания

Рассмотрим теперь четыре операции вычитания, приведенные в табл. 10.1, относящиеся к составу команд типового микропроцессора. Каждая команда вычитает содержимое некоторого регистра или ячейки памяти из содержимого аккумулятора. В силу внутренних особенностей АЛУ не обладает возможностями вычитания, оно осуществляет сложение, представляя вычитаемое в форме дополнительного кода и затем складывая его.

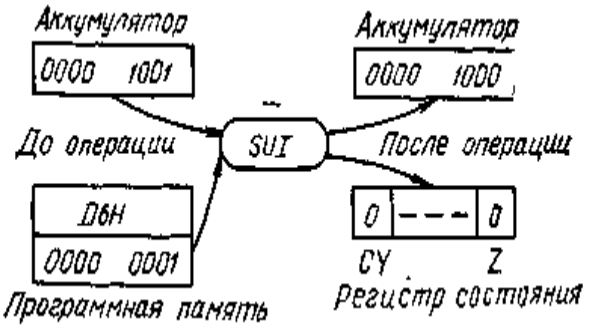
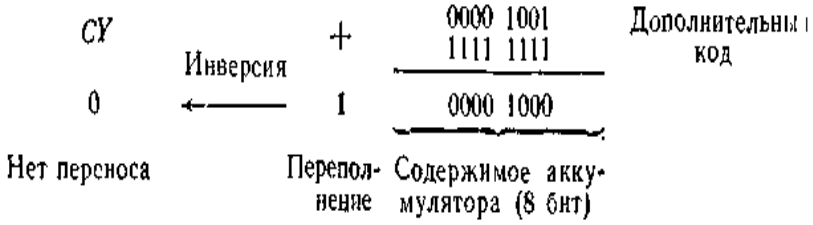
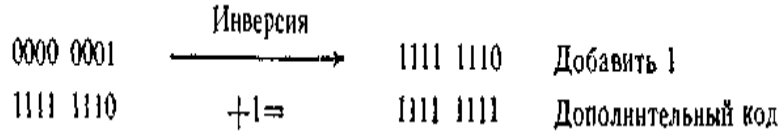


Рис. 10.3. Команда вычитания данных с непосредственной адресацией

В качестве примера рассмотрим процесс вычитания, представленный на рис. 10.3. Двоичное число 0000 0001 вычитается из 0000 1001 (09H — 01H = 08H). Выполнение этой операции обычным способом дает следующий результат:



Напомним, что вычитание двоичных чисел осуществляют сложением первого числа со вторым, представленным в форме дополнительного кода, пренебрегая переполнением. В этой задаче вторым числом будет 0000 0001, которое будет преобразовано в дополнительный код и даст:

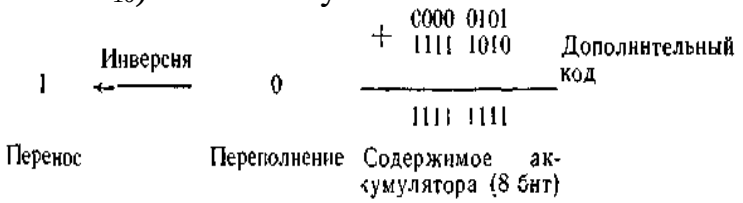


Дополнительный код 1111 1111 другого числа складывается тогда с первым числом, что дает сумму 1 0000 1000. В старшем бите суммы 1 является переполнением и не принадлежит разности $0000\ 1000_2$. Микропроцессор использует это переполнение для установления индикатора переноса. Вычитая, МП инвертирует переполнение, и результат становится содержимым индикатора переноса *CY*. Переполнение 1 инвертируется и сбрасывает индикатор переноса в 0. Когда в ходе вычитания индикатор переноса сбрасывается, это значит, что переноса не было и что первое число больше второго.

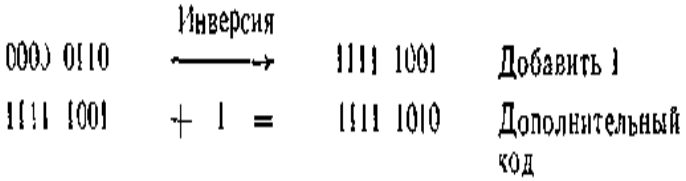
Команда вычесть непосредственные данные (мнемоника SUI)

используется в примере на рис. 10.3. Следующие непосредственно заКОП данные второго байта памяти 0000 0001_2 вычитаются из содержимого аккумулятора 0000 1001_2, разность передается в аккумулятор, после чего операция завершается. Индикатор переноса сбрасывается в 0. Это означает, что переноса не было или содержащееся в аккумуляторе число до операции было больше числа в памяти. Индикатор нуля проверяет содержимое аккумулятора. Операция вычитания завершается, содержимое 0000 1000— не нуль, индикатор нуля также сброшен в 0.

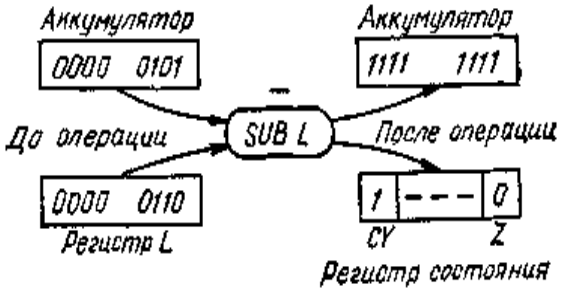
Рассмотрим другой пример вычитания, когда уменьшаемое меньше вычитаемого. Пусть надо вычесть двоичное число 0000 0110 из двоичного 0000 0101 (05H—06H = FFH = -1_{10}). В этом случае:



Вычитаемое здесь представлено в дополнительном коде



Уменьшаемое 0000 0101_2 сложено с результатом преобразования вычитаемого в дополнительный код 1111 1010, что дает 1111 1111. Как можно увидеть, результат 1111 1111 является представлением в дополнительном коде числа -1_{10}. Сложение не вызывает переполнения тогда, когда имеем 0 в регистре переполнения. Этот результат инвертируется (так как речь идет о вычитании), что дает 1 в регистре переноса *CY*. Когда индикатор *CY* устанавливается в 1 после вычитания, это означает, что число, содержащееся в аккумуляторе, младше числа в памяти или в регистре. Индикатор переноса в состоянии 1 показывает, что число, содержащееся в аккумуляторе после того, как вычитание было выполнено, является дополнительным кодом, представляя отрицательное число. 1111 1111 в аккумуляторе на рис. 10.4 представляет собой -1_{10}.



Рнс. 10.4. Команда вычитания (L) из (A)

Пример 1 Из числа *A3H* вычесть число *1EH*.
Результат занести в ячейку памяти с адресом *1030H.*
Программа:

Адрес	Ассемблерный код	Комментарий
00	*MVI A, A3H*	; Загрузить в *A* число *A3H*

```
02        SUI  I E H      ; Вычесть из числа A3H
03                        ; число IEH
04     LXI  H, 1030H      ;Установить адрес I030H
05                          в HL
06
07        MOV   M, A      ; Переслать результат из A
                            в ячейку памяти, адресуемую парой HL
08     END                ; Конец.
```

Таким образом, под управлением команд рассмотренной группы микропроцессор может выполнять в АЛУ арифметические операции. Поскольку МП является одноадресным, то один из операндов в арифметических операциях всегда помещается в аккумулятор, неявно адресуемый самим кодом операции. Вслед за кодом операции необходимо указывать местонахождение второго операнда. Результат (сумма или разность) помещается в аккумулятор. Операнд, который перед арифметической операцией находился в аккумуляторе, после выполнения операции уничтожается.

Команда сложения ADD, ADC

ADD r; A+B→A

ADC r; A+B+CY→A

Команда ADD обеспечивает сложение содержимого аккумулятора с содержимым другого регистра, при этом результат операции остаётся в аккумуляторе.

Команда ADC является разновидностью команды ADD. По команде ADC происходит не только сложение двух операндов, но и сложение с признаком переноса CY, оставшимся от предыдущей операции. Результат сохраняется в аккумуляторе.

Команда сложения ADI, ACI

Команда ADI осуществляет сложение операнда, который непосредственно следует за кодом операции, с содержимым аккумулятора. По команде ACI непосредственный операнд суммируется с содержимым аккумулятора и с признаком переноса CY

ADI data8; A+ data8 →A

ACI data8; A+ data8 +CY→A

Команда вычитания SUB, SBB

Команда SUB позволяет микропроцессору непосредственно вычесть содержимое одного из регистров общего назначения или ячейки памяти M из содержимого аккумулятора. Команда SBB является разновидностью команды SUB. По этой команде осуществляется вычитание с заемом.

SUB r A – r →A

SUB M A – M →A

SBB r A – r - CY →A

SBB M A – M - CY →A

Команда вычитания SUI, SBI

По команде SUI из содержимого аккумулятора вычитается операнд, который непосредственно следует за кодом операции. По команде SBI из

содержимого аккумулятора вычитается и непосредственный операнд, и признак заёма CY.

SUI data8; A+ data8 →A

SBI data8; A+ data8 +CY→A

Внимание! Все описанные арифметические команды сложения и вычитания изменяют (модифицируют) содержимое всех признаков регистра признаков.

Команды INR, DCR

Команда INR является разновидностью команды ADD. По этой команде МП увеличивает на 1 содержимое одного из регистров РОН, аккумулятора или ячейки памяти М.

Команда DCR является разновидностью команды вычитания SUB. По этой команде МП уменьшает на 1 содержимое одного из регистров РОН, аккумулятора или ячейки памяти М.

INR r r + 1 → r

INR M M + 1 → M

DCR r r – 1 → r

DCR M M – 1 → M

Внимание! Эти команды модифицируют все признаки за исключением признака переноса CY.

Команды INX, DCX

Команда инкремента INX и декремента DCX позволяют соответственно увеличить и уменьшить на 1 содержимое регистровых пар (B-, D- , H- пары) и указателя стека SP.

INX rp; rp + 1 → r

DCX rp; rp – 1 → r

Внимание! Эти команды не модифицируют регистр признаков.

Команды DAD, DAA

Команда двойного сложения DAD суммирует содержимое регистровой пары (H,L) и адресуемой регистровой пары rp (rp это B-,D- H- пары, SP)

DAD rp; (H,L) + rp → (H,L)

Внимание! Эта команда модифицирует только признак переноса.

Команда десятичной коррекции аккумулятора DAA осуществляет перевод 8- разрядного двоичного числа в аккумуляторе в две цифры двоично-десятичного кода с правильной установкой признака переноса CY. При этом производятся следующие действия: 1. Если младшая тетрада содержит число, больше 910, или установлен признак вспомогательного переноса AC=1, то содержимое аккумулятора увеличивается на 610. 2. Если после этого старшая тетрада аккумулятора содержит число, большее 9, или установлен признак вспомогательного переноса CY=1, то в старшую тетраду прибавляется 610

Внимание! При десятичной коррекции модифицируются все признаки регистра признаков.

Кроме рассмотренных типовой микропроцессор обладает четырьмя командами сравнения (см. четыре последние строки в табл. 9.1). Команды сравнения вычитают содержимое регистра или ячейки памяти из содержимого аккумулятора, но не изменяют содержимое ни того, ни другого. Индикаторы же подвержены воздействию команд сравнения.

На рис. 10.5 приведен пример использования команды СРАВНИТЬ регістр *L*.

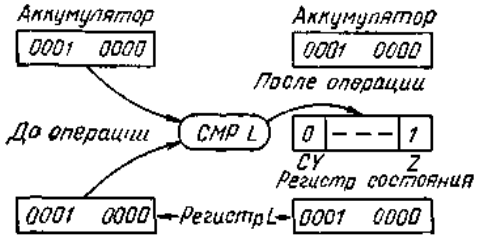
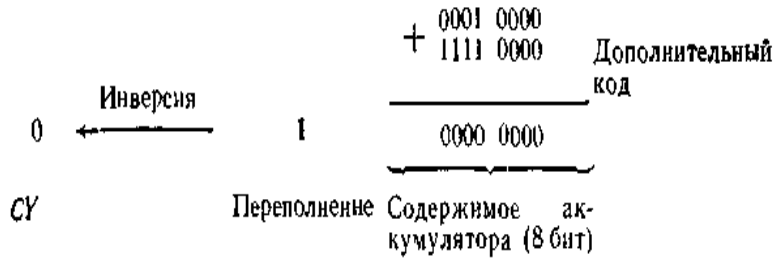


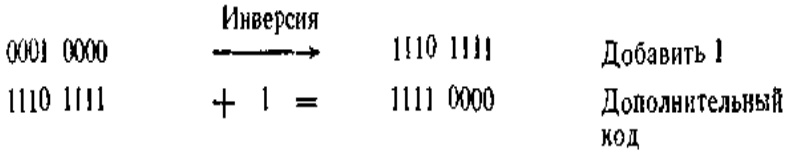
Рис. 10.5. Команда сравнения (A) с (L)

Равные числа 0001 0000_2 являются содержимым аккумулятора и регистра *L*и сравниваются микропроцессором. Заметим, что ни одно, ни другое из содержимых не изменяется после операции сравнения. Индикаторы подвержены влиянию результата сравнения.

Согласно приведенному в табл. 10.1 символическому представлению информации содержимое регистра *L* вычтено из содержимого акумулятора *A* по команде СМР *L:*



Второе число (содержимое регистра L) переведено в дополнительный код:



Первое число 0001 0000_2 и дополнительный код второго числа 1111 0000 складывается, что дает 1 0000 0000. Затем проверяется равенство 0 восьми младших разрядов. Индикатор нуля принимает значение 1. Переполнение 1 инвертируется АЛУ, и в этом примере индикатор переноса принимает значение 0. Сброшенный индикатор переноса *CY* означает, что содержимое аккумулятора больше или равно содержимому регистра L.Второй является команда сложить содержимое регистров *L*и *A* (мнемоника ADD L), на рис. 10.1, *б* показано ее выполнение. Содержимое аккумулятора (0000 1000_2) складывается с содержимым регистра *L*(0000 0001_2), получающаяся в результате выполнения команды ADD *L*сумма (0000 1001_2) помещается в аккумулятор.

Тема 11. Команди логічних операцій та зсувів.

Команды логических операций

В системах команд всех ЭВМ есть команды логических операций. Они выполняют поразрядные операции инвертирования, И, ИЛИ, исключительно ИЛИ (сложения по *mod 2*), а также циклический сдвиг аккумулятора. При этом

инвертирование заменяет каждый бит его дополнением; операция И дает в результате бит, равный единице, если соответствующие биты обоих операторов содержат единицу; операция ИЛИ - бит, равный единице, если, соответствующие биты в одном или обоих операнда содержат единицы; операция исключающего ИЛИ - бит, равный единице, если только соответствующий бит одного операнда содержит единицу.

Большинство применений логических операций связано с изменением, обнулением /сбросом/ и проверкой бит в байте. Эти действия выполняются с помощью логических операций над нужным байтом и вторым байтом - маской. Такая операция называется маскированием.

Логические команды сведены в табл. 11.1 и содержат команды И, ИЛИ, ИЛИ ИСКЛЮЧАЮЩЕЕ,НЕ (инверсия) и сдвига. Здесь именно аккумулятор составляет ядро большинства операций. Как и при арифметических командах, способ адресации и здесь влияет на способ и место нахождения других данных в системе.

Таблица11.1. Команды логических операций типовогоМП

Операция	Адресация	Мнемоника	КОП	Байты	Формат команды	Символика	Установка индикаторов
А И данные	Непосредственная	AN1	E6	2	коп данные	(A) ← (A) • (байт 2)	Z CY — Сброс
А И L	Регистровая	ANA L	A5	1	коп	(A) ← (A) • (L)	Z CY — Сброс
А И Н		ANA H	A4	1	коп	(A) ← (A) • (H)	Z CY — Сброс
А И LOC (HL)	Косвенная регистровая	ANA M	A6	1	коп	(A) ← (A) • (H)	Z CY — Сброс
А ИЛИ данные	Непосредственная	OR1	F6	2	коп данные	(A) ← (A) + (байт 2)	Z CY — Сброс
А ИЛИ L	Регистровая	ORA	B5	1	коп	(A) ← (A) + (L)	Z CY — Сброс
А ИЛИ Н		ORA H	B4	1	коп	(A) ← (A) + (H)	Z CY — Сброс
А ИЛИ LOC (HL)	Косвенная регистрация	ORAM	B6	1	коп	(A) ← (A) + ((H)(L))	Z CY — Сброс
А ИЛИ ИСКЛЮЧАЮ ЩЕЕ данные	Непосредственная	XRI	EE	2	коп данные	(A) ← (A) ○ (байт 2)	Z CY — Сброс
А ИЛИ ИСКЛЮЧАЮЩЕЕ А	Регистровая	XRA A	AF	1	коп	(A) ← (A) ○ (A) Сброс А	Z = 1 CY — Сброс
А ИЛИ ИСКЛЮЧАЮЩЕЕ L	»	XRAL	AD	1	коп	(A) ← (A) ○ (L)	Z CY — Сброс
А ИЛИ ИСКЛЮЧАЮ ЩЕЕ Н	»	XRA H	AC	1	коп	(A) ← (A) ○ (H)	Z CY — Сброс
А ИЛИ ИСКЛЮЧАЮЩЕЕ LOC (HL)	Косвенная регистровая	XRAM	AE	1	коп	(A) ← (A) + ((H)(L))	Z CY — Сброс
Инвертировать А	Неявная	CMA	2F	1	коп	(A) ← (A')	
Сдвиг А вправо	»	RAR	1F	1	коп	... →(CY) →(A) → (CY)→ ...	CY
Сдвиг А влево	»	RAL	17	1	коп	... ← (CY) ← (A) ← (CY) ← ...	CY

1. Логическое «И», логическое умножение или конъюнкция.

Конъюнкция - это сложное логическое выражение, которое считается истинным в том и только том случае, когда оба простых выражения являются истинными, во всех остальных случаях данное сложеное выражение ложно. Логическое «И» обычно обозначается следующими символами: &, *, или же отсутствием символа между двумя логическими высказываниями. Высказывание A&B (AB, AxB) истинно тогда и только тогда, когда истины оба высказывания A и B. Это обстоятельство удобно выразить в виде таблицы истинности для конъюнкции

Таблица истинности для конъюнкции

A	B	F
1	1	1
1	0	0
0	1	0
0	0	0

Рассмотрим выполнение типовым микропроцессором команды И непосредственно, рис.11.1,а.

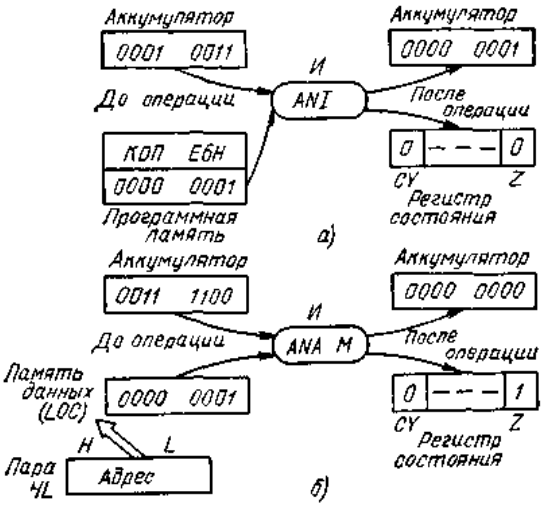


Рис. 11.1. Команда И с непосредственной адресацией (а) и команда ANA M *(б)*

Содержимое аккумулятора (0001 0011) подвержено операции И побитно. Согласно таблице истинности для операции И только самые младшие биты каждого числа равны 1, следовательно, результатом будет 0000 0001, он помещается в аккумулятор. Согласно последней колонке табл. 11.1 результатом всех операций И будет сброс индикатора переноса, мы это видим также на рис. 11.1, а. Результат операции *И* проверяется с целью определения — не нуль ли он, и если нет, то индикатор нуля сбрасывается в 0. Отметим использование точки (•) в табл. 11.1 в колонке символов для обозначения операции И.

На рис. 11.1,*б* приведен другой пример использования команды И, в этом случае — И косвенной адресации (мнемоника ANA *M).* Содержимое аккумулятора подвержено операции И (бит с битом) с содержимым ячейки памяти, указанной парой *HL.*После операции 0011 1100 и0000 0001 полученный результат 0000 0000 помещен в аккумулятор. Индикатор переноса сбрасывается (СБРОС) согласно табл. 11.1. Помещенный в аккумулятор результат проверяется, и, поскольку он равен 0, индикатор нуля устанавливается в 1.

Внимательно рассмотрим рис. 11.1,б. В этих двух примерах второй операнд — 0000 0001, он используется как *маска.* Маска 0000 0001 на рис. 11.1, а и *б* может быть использована для сброса в 0 семи старших бит или, с учетом наличия индикатора нуля, для тестирования значений 0 или 1 в позиции младшего бита аккумулятора (в этом случае на единственную 1 надевается маска 0000 0001). Но нужно быть осторожным. Заметим, что содержимое аккумулятора изменилось после операции И. Некоторые микропроцессоры снабжены специальными командами тестирования битов, которые выполняют

операции И с содержимым аккумулятора и с маской байта без изменения содержимого всего аккумулятора, изменяя состояние индикаторов.

2*.Логическое «ИЛИ», логическое сложение или дизъюнкция:*

Логическое «ИЛИ», называемое также дизъюнкцией и логическим сложением. - это сложное логическое выражение, которое истинно, если хотя бы одно из простых логических выражений истинно и ложно тогда и только тогда, когда оба простых логических выраженныя ложны. Обычно данная операция обозначается символом V (AVB) Операция дизъюнкции может быть представлена следующей таблицей истинности:

Таблица истинности для дизъюнкции

A	**B**	**F**
1	1	1
1	0	1
0	1	1
0	0	0

Четыре команды ИЛИ (см. табл. 11.1) выполняются с содержимым аккумулятора и содержимым какой-либо другой ячейки памяти и регистра. На рис. 11.2 приведен пример операции ИЛИ, когда содержимым аккумулятора будет 1100 1100, тогда как регистр *L*содержит 0000 1111. Результат—1100 1111. Числа подвержены операции ИЛИ побитно согласно таблице истинности ИЛИ Содержимое 0000 1111 регистра *L*может рассматриваться как маска, которая всегда будет обращать 4 младших бита в 1111. Отметим использование логического знака (+) для обозначения операции ИЛИ в колонке символов в табл. 11.1.

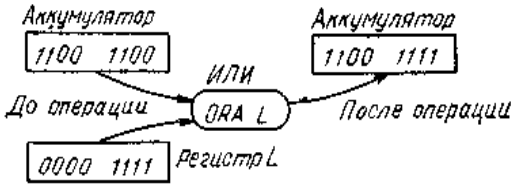


Рис. 11.2. Команда (A) ИЛИ (L)

3. *Логическая команда «исключающегоИЛИ»*

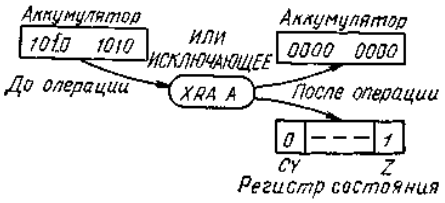
В табл. 10.1 приведены пять команд «исключающего ИЛИ» для типового микропроцессора. Эти команды производят поразрядное двоичное сложение операндов. Операция «исключающего ИЛИ», обозначаемую как ⊕. Так запись A⊕B означает либо A либо B, но не оба вместе. Таблица истинности для «исключающего ИЛИ» имеет вид:

A	**B**	**F**
0	0	0
1	0	1

0	1	1
1	1	0

Команда XRA применяется для инвертирования определённых битов слова с помощью слова маски. Например, для инвертирования седьмого и первого битов маска имеет вид 10000010. Тогда не зависимо от того какое число будет в этих разрядах после поразрядного сложения, согласно таблицы истинности, будет инвертирование в указанных разрядах.

На рис. 11.1 показан процесс выполнения команды «исключающего ИЛИ» *А с А.* Результатом «исключающего ИЛИ» 1010 1010 с самим собой (рис. 11.3) будет 0000 0000. Выполнение этой операции любого числа с самим собой всегда дает результат 0000 0000, индикатор нуля всегда устанавливается в 1, что означает нулевое содержимое индикатора, а в соответствии с табл. 11.1 индикатор переноса *CY* всегда будет сброшен в 0.



Рнс. 11.3. Команда XRA *A*

4. *Логическое отрицание «НЕ» или инверсия.*

Инверсия - это сложное логическое выражение. Если исходное логическое выражение истинно, то результат отрицания будет ложным, и наоборот, если исходное логическое выражение ложно, то результат отрицания будет истинным.

Таблица истинности для инверсии

А	**не А**
1	0
0	1

Операция инвертирования осуществляется с содержимым аккумулятора посредством мнемокоманд: CMA, CMP, CMP M.

Логические команды позволяют установить в единицу, сбросить в ноль, инвертировать и проверить интересующие нас биты. ***Установка бита*** в единицу производится формированием *маски*, которая определяет позицию требуемого бита. Маска для установки бита – это байт с нулями во всех битах, кроме искомого. Затем выполняется операция ИЛИ с содержимым аккумулятора и полученной маской. Аналогично, ***сброс*** бита в ноль осуществляется операцией И над маской, с единицами во всех битах, кроме требуемого, обозначенного нулем. После этих операций следует команда перехода по условию состояния флага Z.

Для примера запишем программы изменения содержимого пятого бита

регистра B:

```
;Установка в "1"      ;Сброс в "0"      ;Инверсия бита
MVI A,00100000B       MVI A,11011111B   MVI A,00100000B
ORA  B                ANA B             XRA  B
MOV B,A               MOV B,A           MOV B,A
```

Команды сдвига.

Восьмиразрядный микропроцессор имеет четыре команды циклического сдвига. Операндом их является содержимое аккумулятора, в котором формируется результат. Сдвиги выполняются влево и вправо на один разряд. В зависимости от того, что помещается в освобождающийся при сдвиге бит и как используется выдвигающийся бит и «флаг переноса» вводятся несколько команд сдвигов.

RLC – циклический сдвиг содержимого аккумулятора на одну позицию влево. Младший бит и флаг C принимают значение вытесненного бита, т.е. бывшего старшего бита;

RRC – циклический сдвиг содержимого аккумулятора на одну позицию вправо. Старший бит и флаг C принимают значение вытесненного бита, т.е. бывшего младшего бита;

RAL – циклический сдвиг содержимого аккумулятора на одну позицию влево вместе с флагом C. В младшем бите устанавливается содержимое флага C;

RAR – циклический сдвиг содержимого аккумулятора на одну позицию вправо вместе с флагом C. В старшем бите устанавливается содержимое флага C.

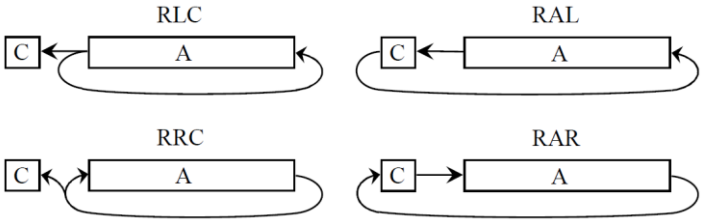


Рис.11.4.Выполнение команд циклического сдвига

Рассмотрим приведенный на рис. 11.5 пример использования команды циклического сдвига вправо с переносом: содержимое аккумулятора (0011 0001) сдвинуто на одну позицию вправо и его младший бит (1 в этом примере) передается в позицию бита индикатора переноса, тогда как имевшийся там бит занимает позицию старшего бита аккумулятора, в котором содержится 0001 1000 после завершения операции. Индикатор переноса установится в 1, индикатор нуля не изменится.

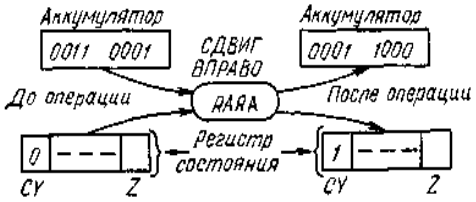


Рис.11.5. Команда RAR *A*

Используя одну или несколько команд циклического сдвига, можно тестировать весь заданный состав бит, а индикатор переноса может быть сброшен или установлен. Индикатор переноса может быть тестирован затем командой условного ветвления. Тест паритета является другим приложением использования команд сдвига. Паритет двоичного числа определяется числом содержащихся в нем единиц: четный паритет — общее число единиц четное; нечетный паритет — общее число единиц нечетное.

Команды сдвига оперируют только данными аккумулятора и не требуют других операндов, расположенных в памяти или регистрах. Таким образом, команды логических операций используются для манипуляции с переменными по законам алгебры логики. Они могут быть использованы для тестирования и сравнения бит.

Примеры на использование логических команд сдвига

Пример 1. Из содержимого ячейки *40H* выделить младшие 4 бита (младшую шестнадцатеричную цифру) и переслать их по адресу *41H*. Старшие 4 бита (старшую шестнадцатеричную цифру) содержимого ячейки *40H* переслать в младшие разряды ячейки *42H*.

Программа:

Адрес	Ассемблерный код	Комментарий
00	*LXI H, 40H*	; Загрузить
01		непосредственный
02		; адрес в регистровую пару
03		; *HL*
04	*MOV A, M*	; Загрузить в *A*
		данные по адресу *HL*
05	*MOV B, A*	; Запомнить данные в
		регистре *B*
06	*ANI* 00001111B	; Маскировка
		старшей цифры
		;
07	*INX H*	; Увеличить на 1
		содержимое *HL* . .
08	*MOV M, A*	; Запомнить
		младшую цифру
09	*MOV A, B*	; Переслать
		исходное число
		из *B* в *A*
0A	*RRS*	; Четырехкратный
0B	*RRS*	; циклический
0C	*RRS*	; сдвиг
0D	*RRS*	; вправо
0E	*ANI 00001111B*	; Маскировка
		младшей цифры
0F		;
10	*INX H*	; Увеличить на 1
		содержимое *HL*
11	*MOV M, A*	; Запомнить старшую

```
                              цифру
12          END               ; Конец.
```

Пример 2. В ячейке 0040 находится слово, биты 2 и 3 которого необходимо селективно установить, бит 6 сбросить, а бит 5 - изменить, а затем проверить. В ячейках 0030-0032 хранятся маски, соответствующие кодам *OC, BF* и 20.

Ячейки 0030, 0031, 0032, 0040 соответственно имеют следующие метки*: MASKS, MASKS+1, MASKS+2* и *CTRL* .

Программа:

```
Адрес  Ассемблерный код      Комментарий
 00     LXI   H, MASKS       ; Загрузить
                              непосредственный
 01                           ;адрес маски в регистровую
 02                           ; пару HL
 03     LDA   (CTRL)          ; Загрузить в A
                                содержимое
 04                           ; ячейки CTRL
 05                           ;
 06     ORA   M               ; Установить бит 2 и
                              3
 07     INX   H               ; Инкремент до MASKS+1
 08     ANA   M               ; Сбросить бит 6
 09     INX   H               ; Инкремент до MASKS+2
 0A     XRA   M               ; Изменить бит 5
 0B     STA   CTRL            ; Запомнить в CTRL
 0C                           ;
 0D                           ;
 0E     ANA   M               ; Проверить бит 5
 0F       *                   ; Конец.
```

Таким образом, с помощью команд логических операций и сдвигов могут быть выполнены логические операции И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ (сложение по модулю 2) с использованием двух операндов. Один из операндов всегда размещается в аккумуляторе. Результат ***выполнения команды*** фиксируется в аккумуляторе.

Команда ANA

Команда ANA осуществляет логическую операцию И над содержимым аккумулятора и содержимым одного из регистров РОН, аккумулятора и ячейки М

ANA r; A & r→ A

ANA M; A & M→ A

Команда ORA

Команда ORA реализует логическую ИЛИ над содержимым аккумулятора и содержимым одного из регистров РОН, аккумулятора и ячейки М

ORA r; A Ú r→A

ORA M A Ú M→A

Команда XRA

Команда XRA осуществляет логическую операцию ИСКЛЮЧАЮЩЕЕ ИЛИ над содержимым аккумулятора и содержимым одного из регистров РОН,

аккумулятора и ячейки M

XRA r; A Å r→ A

XRA M; A Å M→ A

Команды ANI, ORI, XRI

Эти команды отличаются от команд ANA, ORA, XRA тем, что второй операнд следует непосредственно за кодом операции.

ANI data8; A & data8→ A

ORI data8; A Ú data8→ A

XRI data8; A Å data8→ A

Команды CMP, CPI

Команда CMP используется для сравнения двух чисел, одно из которых находится в аккумуляторе, а другое в одном из регистров РОН, аккумуляторе или ячейке M. При сравнении одно из чисел вычитается из другого числа. В соответствии с результатом формируются признаки регистра признаков. Содержимое аккумулятора при этом не изменяется. CMP r; A – r

CMP M; A – M

CPI data8 A – data8

Разница между командами CMP и SUB в том, что при выполнении команды CMP результат операции не фиксируется в аккумуляторе.

Внимание! Команды логических операций и сравнения модифицируют регистр признаков.

Команда CMA

Команда CMA используется для инвертирования содержимого аккумулятора.

CMA; A → A

Внимание! Команда CMA не модифицирует регистр признаков.

Команды STC, CMC

STC; 1 → CY

CMC; CY → CY

Команда STC устанавливает признак переноса CY. Команда CMC инвертирует признак переноса.

Команды сдвига RLC, RRC, RAL, RAR

Если надо произвести операции сдвига над данными, то их необходимо предварительно поместить в аккумулятор. Операндом однобайтных команд сдвига является содержимое аккумулятора, в котором формируется результат. Сдвиги выполняются влево (RLC, RAL) и вправо (RRC, RAR) только на один разряд. Выполнение команд сдвига поясняется на рисунке ниже.

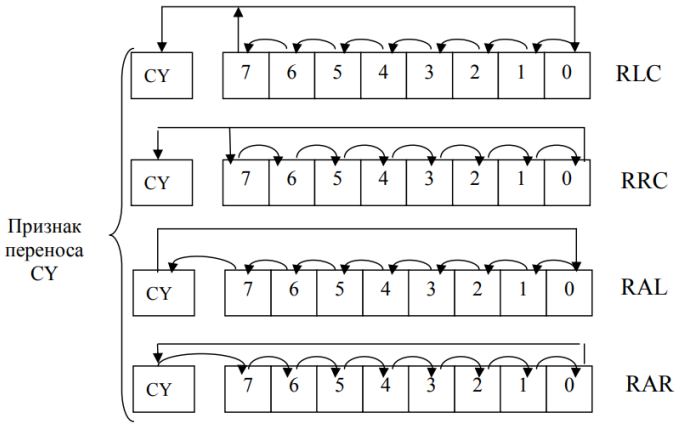


Рис.11.6 Команды сдвига

Состав команд прочих операций

Эти команды составляют последнюю категорию, которыми наделен типовой микропроцессор. Они сведены в табл. 11.2 и содержат команды помещения в стек, извлечения из стека, отсутствия операции и команду остановки. При их выполнении индикаторы не изменяются.

Команды помещения в стек и извлечения из него уже упоминались. Они используются всегда парно, так как то, что в стек помещается, должно быть из него извлечено. Они широко распространены при использовании подпрограмм. Команда поместить в стек содержимое аккумулятора *A* и индикаторов могла бы быть, например, первой командой подпрограммы. Она сохранила бы содержимое аккумулятора и индикаторов независимо от подпрограммы. Точно так перед операцией возврата команда извлечь из стека *A* и индикаторы восстановила бы начальное содержимое аккумулятора и индикаторов.

Таблица 11.2. *Прочие команды типового микропроцессора*

Операция	Адресация	Мнемоника	КОП	Байт	Формат команды	Симнолика
Поместить в стекАи индикаторы	Косвенная регистровая	PUSH PSW	F5	1	коп	((SP) - 1) ← (A) ((SP) - 2) ← (индикаторы) (SP) ← (SP) - 2
Поместить в стек HL	То же	PUSH H	E5	1	коп	((SP) - 1) ← (H) ((SP) - 2) ← (L) (SP) ← (SP) - 2
Извлечь из стека А и индикатора	»	POP PSW	Fl	1	коп	(индикаторы) ← ((SP)) (A) ← ((SP) + 1) (SP) ← (SP) + 2
Извлечь из стекла HL	»	POP H	El	1	коп	(L) ← ((SP)) (H) ← ((SP) + 1) (SP) ← (SP) + 2
Нет операций	Неявная	NOP	00	1	коп	(L) ← ((SP)) (H) ← ((SP) + 1)
Останов	Неотделим	HLT	76	1	коп	(SP) ← (SP) + 2 (PC) ← (PC) +1

Рассмотрим первую команду поместить в стек *A* и индикаторы (PUSH *PSW).*Часть *PSW*соответствует слову состояния программы которое в данном случае является содержимым аккумулятора и регистра состояния

(индикаторов). Команда PUSH *PSW*является однобайтовой, содержимое аккумулятора помещается первым, а регистра состояния — вторым.

Команда НЕТ ОПЕРАЦИЙ соответствует отсутствию всякого выполнения операций в течение 1 или 2 мкс. Это однобайтовая команда, единственным эффектом которой является инкремент счетчика команд. Никакой другой регистр не затрагивается. Эта команда используется как дополнение (когда одна или две команды отменены в ходе наладки) и связывает две части программы так, чтобы МП мог обратиться от одной к другой. Она может также служить для ввода интервала времени в цикл временной задержки.

Команда ОСТАНОВ используется в конце программы для остановки микропроцессора. В этом случае только СБРОС или команда вызова прерывания может позволить новый запуск типового микропроцессора.

Тема 12. Організація переходів у програмах.
Команды операций ветвления

Команды ветвления составляют четвертую группу команд, которой снабжен типовой МП. Они приведены в табл. 12.1. Заметим, что описание относится к командам перехода. Действительно, термины *ветвление* и *переход* приводятся в этой главе как синонимы. Однако некоторые разработчики усматривают разницу между ними. Эти команды называют иногда *командами передачи управления.* Там, где они будут встречаться, мы все-таки их будем квалифицировать как команды перехода, следуя договору с фирмой Intel.

Обычно микро-ЭВМ выполняет команды последовательно. Шестнадцатиразрядный счетчик команд типового микропроцессора хранит всегда адрес следующей извлекаемой из памяти команды до ее выполнения. Содержимое его обычно повышается в каждом счете. Команды ветвления или перехода являются средством изменения значения содержимого счетчика команд и, следовательно, изменения нормальной последовательности выполнения программы. Эти команды разделены на две группы; *безусловного перехода* и *условного перехода.* Первая команда в табл. 12.1 является командой безусловного перехода. Команда ПЕРЕЙТИ непосредственной адресации является трехбайтовой, используемой для изменения специфического адреса

в счетчике команд МП. На рис. 12.1 приведен пример использования такой команды безусловного перехода. Здесь адрес 2000H загружен в счетчик команд, информация о нем следует непосредственно за КОП, поэтому адресация называется непосредственной. Заметим на рис. 12.1, что младшая часть адреса находится во 2-м байт памяти,

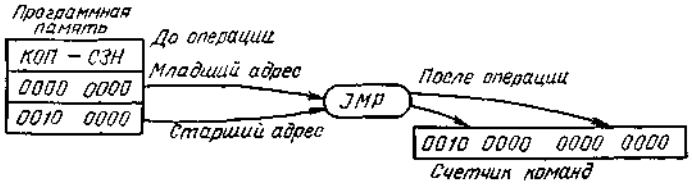


Рис. 12.1. Команда безусловного перехода

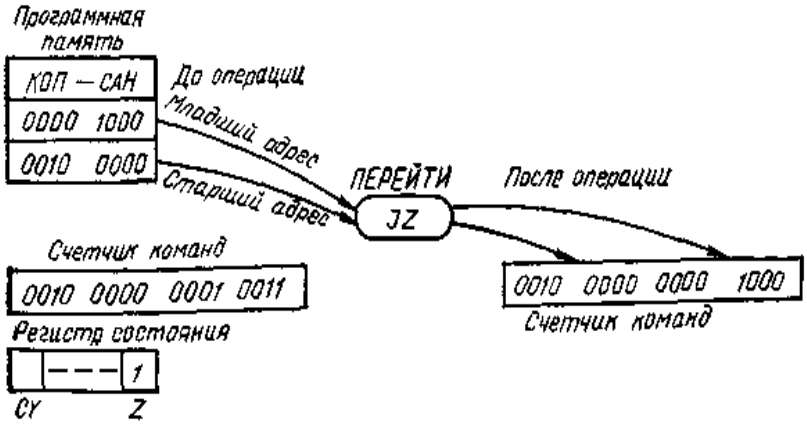
а старшая является содержимым 3-го байт памяти. Одна команда этого типа будет использована для запуска счетчика команд в момент начала выполнения

новой программы. Мы можем рассматривать команду ветвления или безусловного перехода как способ загрузки новой информации об адресе в счетчик команд.

Таблица 12.1. Команды ветвления или перехода

Операция	Адресация	Мнемони-ка	коп	Байты	Формат команды	Символика
Перейти в LOC	Непосредственная	JMP	C3	3	коп мл.адрес ст. адрес	(PC) ← (адрес)
Перейти в LOC, если 0	»	JZ	CA	3	КОП мл.адрес ст. адрес	Если Z = 1, то (PC) ← (адрес)
Перейти в LOC, если не 0	»	JNZ	C2	3	КОП мл.адрес ст. адрес	Если Z = 0, то (PC) ← (адрес)
Перейти в LOC, если перенос	»	JC	DA	3	КОП мл.адрес ст. адрес	Если CY = 1, то (PC) ← (адрес)
Перейти в LOC, если нет переноса	»	JNC	D2	3	коп мл.адрес ст. адрес	Если CY = 0, то (PC) ← (адрес)

Четыре последние команды ветвления в табл. 12.1 являются командами условного перехода. Эти команды повлекут за собой непосредственно загрузку адреса, только если будут выполнены специальные условия. В противном случае счетчик команд будет инкрементирован нормально. На рис. 12.2 показан пример команды перейти, если 0. В этом случае счетчик команд 2013H до операции будет нормально инкрементирован, если только индикатор нуля не установлен в 1. Микропроцессор проверяет это и находит 1, значит, результатом последней логической или арифметической операции был 0. Условия перехода выполнены, и МП загружает новый адрес 2008H в счетчик команд. Этот новый адрес поступает из памяти программы; команда опроса использует непосредственную адресацию. Следующей выполняемой командой будет команда размещения данных в памяти по адресу 2008H (но не по адресу 2013H), как мы могли бы предположить при последовательном выполнении программы. Наш МП перешел к новой ячейке памяти программы. Отметим, что пример на рис. 12.2 показывает переход назад, что случается наиболее часто.



Рнс. 12.2. Команда перехода, если нулевой результат

Команды перехода или ветвления существуют практически во всех программах МП. Они очень эффективны как средство принятия решений и удобны для формирования циклов программы.

Ветвление программ

Рассмотрим простую задачу сравнения двух чисел. Нужно найти наибольшее и поместить его в определенную ячейку памяти (см. структурную схему решения этой задачн на рис. 12.3). В двух первых прямоугольных кадрах представлена операция загрузки двух чисел в регистры А (аккумулятор) и (данные). Следующий прямоугольный кадр соответствует операции сравнения, где содержимое регистра вычитается из содержимого регистра А. Команды сравнения не разрушают содержимого регистров, но влияют на индикатор.

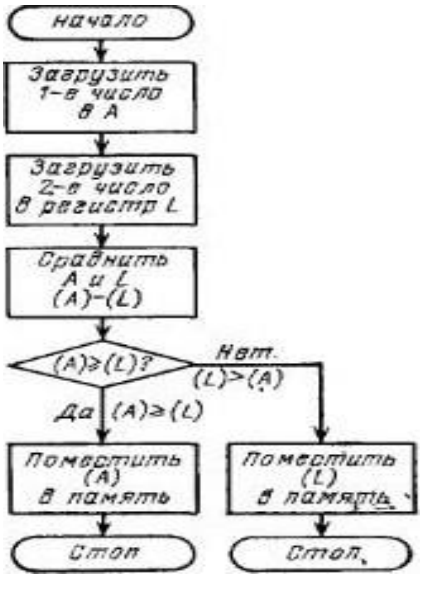


Рис.12.3 Структурная схема программы сравнения двух чисел и помещения в память

Следующий кадр называется знаком принятия решения. Он завершает в программе этап решения и содержит поставленный вопрос [(A)≥(L)?]. Если ответ на этот вопрос Да, программа продолжается последовательно и содержимое регистра А помещается н память, затем процессор останавливается. Если Нег, программа ответвляется вправо и содержимое регистра помещается в память,

затем МП останавливается, Использование знаков принятия решения приводит в программах к тому, что называется внутренним ветвлением программы.
В табл. 12.2 представлена на ассемблере задача, которую мы только что описали и в которой наибольшее число помещено в память 2040H. Заметим, что здесь аккумулятор A загружен числом 15H, а регистр L числом 6H. Сравнивая (третья команда в этом примере), находим, что (A)>(L), и, следовательно, индикатор переноса (CY) сброшен в 0. Команда перехода (мнемоника JC в примере) проверяет индикатор переноса находит (CY)=0. Микропроцессор обращается тогда к следующей последовательной команде ПОМЕСТИТЬ (A) в память по адресу 2040H и наибольшее число помещено командой STA в память по адресу 2040H. Затем МП останавливается (команда HLT). Три последние команды не были использованы в этом примере.

Таблица 12.2 Ветвящаяся программа на ассемблере

Метка	Мнемоника	Операнд	Комментарий
	MVI	*A*, 0FH	Загрузить первое число (15_{10}) в аккумулятор
	MVI	*L*, 06H	Загрузить второе число (6_{10}) в регистр *L*
	CMP	*L*	Сравнить (A) и (L). Индикатор *CY*=1, если (A)<(L)
	JC	STORE *L*	Перейти к STORE *L*, если *CY*=1 (если A<*L*); если нет, продолжать последовательно
	STA	2040H	Поместить (A) в ячейку памяти 2040H
	HLT		Остановить МП после помещения (A)
STORE *L*	MOV	*A*, *L*	Передать (L) в аккумулятор
	STA	2040H	Поместить (A) в ячейку памяти 2040H
	HLT		Остановить МП после помещения

Если обратиться к рис. 12.3, мы установим, что программа прошла последовательно.

Рассмотрим теперь выполнение такой же программы, но в случае, когда содержимое аккумулятора меньше содержимого регистра L. Программа остается та же (табл.12.2). Содержимое аккумулятора — 0AH (10_{10}), содержимое регистра L — 0EH (14_{10}) здесь больше, Команда сравнения устанавливает индикатор CY в 1, так как (A)<(L). Команда ПЕРЕЙТИ, если индикатор переноса установлен в 1 (JC здесь), проверяет индикатор CY и определяет, установлен ли он, При этом изменяется содержимое счетчика команд, т. е. загружается адрес следующей выполняемой команды (STORE L, в нашем примере). Заметим, что МП перешел через две команды для того, чтобы обратиться туда, где находится символический адрес STORE L. Микропроцессор выполняет тогда три последние команды, которые помещают содержимое регистра L, т. е. наибольшее число 0EH в ячейку памяти по адресу 2040H. Обратившись к схеме на рис. 12.3, мы установим, что программа выполнена последовательно до знака принятия решения, затем ответвилась вправо, заканчиваясь оператором Стоп.

Рассмотрим снова три последние команды в табл. 12.1. Целью этих команд является размещение содержимого регистра L в память по адресу 2040H. Наш макропроцессор не позволяет поместить содержимое регистра L прямо в память. Нужно последовательно передать сначала содержимое L в A, затем поместить содержимое A в память. Таким образом, часто оказывается, что МП с ограниченным составом команд (как в случае типового МП) использовать сложнее, чем устройства, более мощные по своим возможностям.

Таким образом, рассмотренный на структурной схеме способ ветвления осуществляется командами перехода (или ветвления), согласно которым принимаются решения, основанные на состоянии специальных индикаторов. Мы встретились здесь с использованием символического адреса (метки) при команде перехода. Ветвление очень широко используется при программировании.

Циклы

Микропроцессоры особенно эффективны в случае выполнения повторяющихся задач. Структурная схема на рис. 6.44, например, представляет программу, которая будет производить счет от 0 до 254 (0—FEH) и выводить результат счета на печать. Исходя из вершины, она является обычной для установки регистров. В этом примере содержимым аккумулятора A является 00H. Затем содержимое аккумулятора выводится на избранную периферию: это действие является повторяющимся процессом. Затем содержимое аккумулятора инкрементируется или изменяется. Знак сравнения и принятия решения выполняет процедуру проверки, чтобы определить, содержит ли аккумулятор значение FFH. Если ответ на поставленный в знаке принятия решения вопрос (будет ли A=FFH?) отрицателен (*Нет*), программа ветвится влево от знака принятия решения и возвращается в блок введения. Это составляет цикл. Программа будет продолжаться, повторяясь 254_{10} раз.

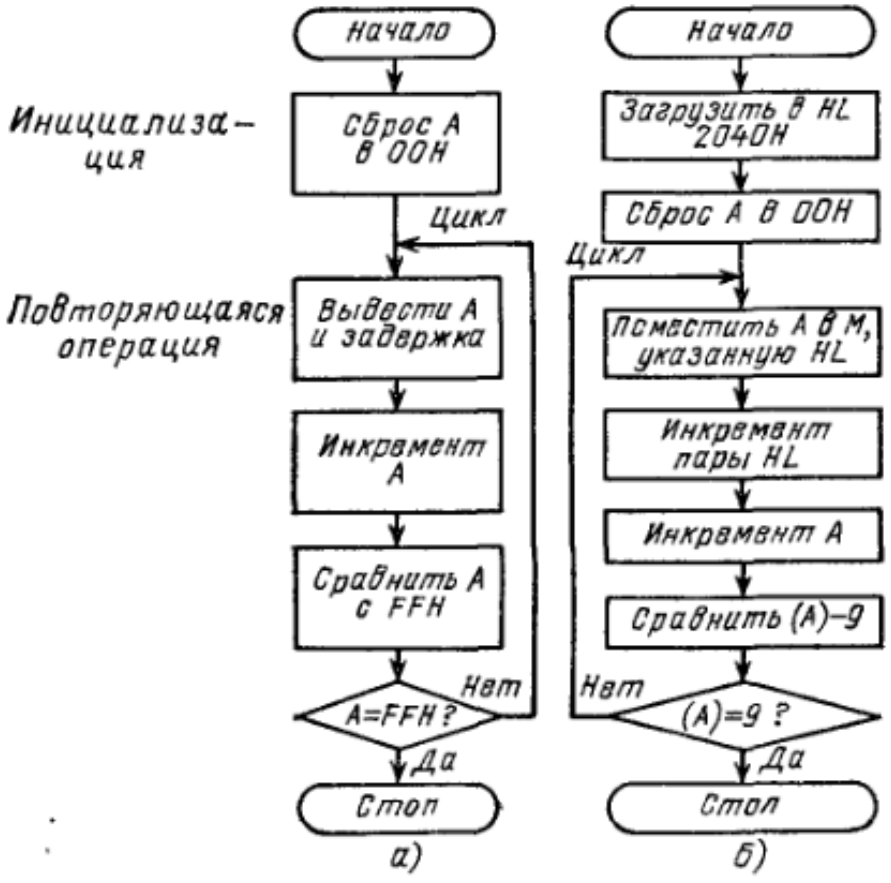


Рис. 6.44. Структурная схема последовательного счета с использованием циклов (*а*) и ее развитие (*б*)

Структурная схема на рис. 6.44, *а* представляет собой программу, которая может содержать только 20 или 30 команд. Если бы 255_{10} пропусков через программу были запрограммированы последовательно, их список составил бы тысячи команд. Циклы в программе являются очень эффективным методом ее сокращения.

На рис. 6.44, *б* приведена другая циклическая программа размещения ряда чисел (от 0 до 8) последовательно в память с адресами от 2040H до 2048H. Две первые прямоугольные рамки соответствуют начальной загрузке пары регистров *HL* и аккумулятора *A* значениями 2040H и 00H соответственно. Третья рамка соответствует процессу размещения данных в памяти, который будет повторен 9 раз в ходе выполнения этой программы. Микропроцессор, повторяя свои действия, разместит содержимое аккумулятора в памяти по адресу, указанному парой *HL*. При первом прохождении цикла содержимое (00H) аккумулятора будет помещено в память по адресу 2040H.

Четвертый и пятый кадры представляют операции, которые изменяют адрес в паре *HL* и счет в регистре *A*. Например, в ходе первого прохождения пара *HL* инкрементируется до 2041H, а аккумулятор — до 01H.

Прямоугольник сравнения и знак принятия решения составляют операцию тестирования. Команда сравнения вычитает 09H из содержимого *A* для восстановления или сброса индикатора нуля. Если (A)<9, индикатор нуля будет сброшен, если (A)=9, индикатор нуля будет установлен в 1. Знак принятия решения опрашивает: Равно ли (A) девяти? Если ответ *Да*, программа выходит из цикла и заканчивается, если, однако, ответ *Нет*, ветвится снова к команде размещения по соседству с вершиной структурной схемы. Команды сравнения и условного перехода используются для проверки изменяющегося счета для определения момента выхода из цикла. Программа, показанная на рис. 6.44, *б*, повторит цикл 9 раз прежде, чем выйти на знак *Стоп*.

В табл. 6.12 приведена версия программы на ассемблере, соответствующая структурной схеме на рис. 6.44, *б*.

Эта программа последовательно поместит числа от 0 до 9 в память по адресам от 2040H до 2048H. Две первые команды эквивалентны двум первым кадрам схемы и предназначены для установки в паре *HL* и в аккумуляторе начальных значений 2040H и 00H соответственно. Команда MOV *M*, *A* помещает содержимое аккумулятора в ячейку

Таблица 6.12. *Версия на ассемблере программы, соответствующей рис. 6.44, б*

Метка	Мнемоника	Операнд	Комментарий
	LXI	*H*, 2040H	Загрузить 2040H в пару *HL* (указатель адреса)
	XRA	*A*	Сброс аккумулятора в 00H, т. е. (A)⊕(A)=00H в аккумуляторе
LOOP	MOV	*M*, *A*	Поместить содержимое аккумулятора ячейку памяти, указанную парой *HL*
	INX	*H*	Инкрементировать пару *HL*
	INR	*A*	Инкрементировать аккумулятор
	CPI	09H	Сравнить (A)=09H? Если (A)=09H, индикатор нуля установлен в 1
	JNZ	LOOP	Перейти к LOOP, если *Z*==0, т. е. если (A)=09H; если нет, продолжать последовательно
	HLT		Остановить МП

памяти, указанную парой *HL*. Две следующие команды (INX *H* и INR *A*) инкрементируют пару *HL* и *A*. Команда CPI сравнивает содержимое *A* с константой 09H. Если *A* содержит значение между 0 и 8, индикатор нуля сброшен в 0, но если *A* содержит 9, индикатор нуля устанавливается в 1.

Команда JNZ проверяет индикатор нуля. Если индикатор нуля сброшен в 0, значит результат вычитания (A) — 09H не равен 0, следовательно, осуществляется переход по символическому адресу LOOP. Если индикатор нуля установлен в 1, результат вычитания (A) — 09H нулевой; при этих условиях программа выходит из цикла и выполняет следующую последовательную команду (HLT), по которой завершается выполнение программы.

Тема 13. Робота з підпрограмами і команди решти операцій.

Во многих программах часто приходится сталкиваться с необходимостью в нескольких ее местах решать одну и ту же задачу. В этом случае обычно пишут подпрограмму (п/п), решающую эту задачу, и обращаются к ней по мере необходимости, не повторяя одни и те же команды в разных точках программы. При использовании п/п необходимо, нарушить обычную последовательность выполнения команд, перейти к п/п, а также обеспечить запоминание адреса возврата в основную программу. Для запоминания адреса возврата в основную программу и для других целей в области оперативного запоминающего устройства (ОЗУ) отводится так называемая стековая область, или стек. Стек - это совокупность регистров, которые принимают и выдают информацию по принципу "последним записан - первым считан" (*last input first Output-LIFO*), т.е. доступ к стеку возможен только с одного конца. Для приема и хранения адреса ячейки стека, к которой было последнее обращение, предназначен шестнадцатиразрядный указатель стека. Содержимое указателя стека уменьшается на единицу при поступлении данных в стек и соответственно увеличивается при выборке из него.

Состав команд операций вызова подпрограмм и возврата в основную программу

Эти команды составляют пятую категорию состава команд типового МП. Их только две, и они приведены в табл. 13.1, Команды вызова (CALL) и возврата (RET) всегда используются парами. При их выполнении индикаторы не изменяются.

Трехбайтовая команда CALL используется основной программой для перехода МП (или ветвления) к подпро-грамме. В примере на рис. 13.1 подпрограмма является короткой последовательностью команд, целью которой является создание интервалов времени в течение 1 с. Когда МП передает первую команду CALL по адресу 1000H, он находит адрес перехода в двух следующих байтах программы. Адрес следующей команды за CALL отправляется в стек (не показанный на рис. 13.1), и МП переходит тогда в начало подпрограммы по адресу 1000H. Команды, которые составляют эту программу счета времени, выполняются, пока МП не передаст команду возврата (RET).

Таблица 13.1. Команды вызова подпрограмм и возврата типового МП

Операция	Адресация	Мнемоника	код	Байты	формат команды	Символика
Вызов подпрограммы	Непосредственная, косвенная регистровая	CALL	CD	3	КОП мл.адрес ст. адрес	((SP) - 1) ← (PCH) ((SP) - 2) ← (PCL) (SP) ← (SP - 2) (PC) ← (адрес)
Возврат из подпрограммы	Косвенная регистровая	RET	C9	1	КОП	(PCL) ← ((SP)) (PCH) ← ((SP) + 1) (SP) ← (SP) + 2

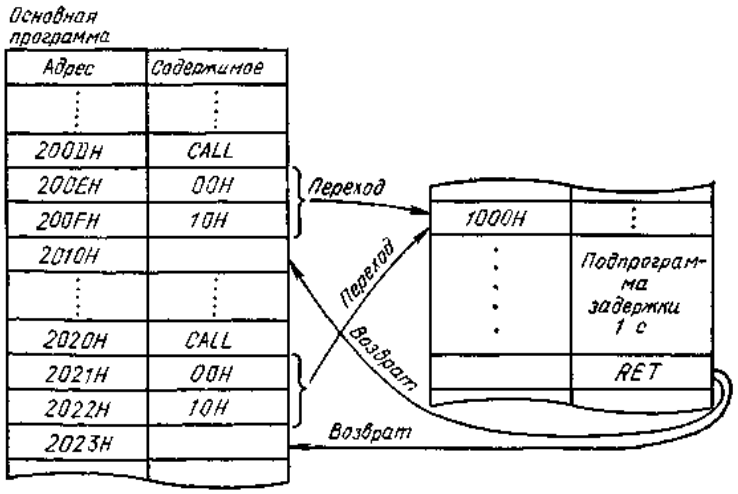


Рис. 13.1. Взаимодействие основной программы и подпрограммы при командах CALL (ВЫЗОВ) и RET (ВОЗВРАТ)

Сохраняющийся в стеке адрес (2010H) отыскивается счетчиком команд, и МП продолжает выполнение основной программы, принимая ее там, где он ее покинул. Это нормальное выполнение продолжается до тех пор, пока МП не встретит другую команду вызова по адресу 2020H. Микропроцессор сохраняет адрес следующей команды (2023H) в стеке и переходит на подпрограмму, начинающуюся адресом 1000H. После завершения выполнения этой подпрограммы команда возврата извлекает из стека адрес следующей команды основной программы и загружает его в счетчик команд. Данная подпрограмма может быть использована много раз в ходе выполнения одной и той же основной программы. Подпрограмма может быть расположена в ОЗУ или ПЗУ.

Команда вызова сочетает функции операций загрузки в стек и перехода. Использование ее показано на рис. 13.2. Сначала она загружает в стек содержимое счетчика команд. Затем счетчик команд должен быть загружен новым адресом для выполнения перехода в подпрограмму.

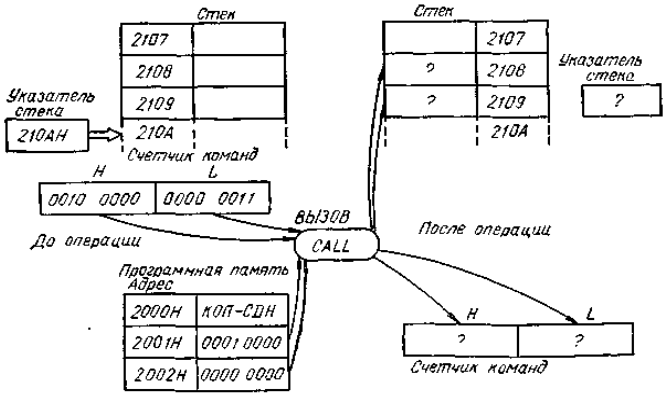


Рис. 13.2. Команда вызова подпрограммы

1. Указатель стека декрементирован от 210AH до 2109H
2. Старший байт счетчика команд загружается в стек по адресу 2109H.
3. Указатель стека декрементирован от 2109H до 2108H.
4. Младший байт счетчика команд загружается в ячейку памяти по адресу 2108H.

5. Младший байт адреса (второй байт памяти) загружается в младший байт счетчика команд.

6. Старший байт адреса (третий байт памяти) загружается в старший байт счетчика команд.

Здесь МП ответвляется по адресу, на который указывает счетчик команд (в приведенном примере 1000Н), являющемуся адресом начала подпрограммы.

Рассмотрим теперь пример применения команды возврата в основную программу RET (рис. 13.3). По командевозврата содержимое стека должно быть передано в счетчик команд. Проследим последовательность выполнения команды по номерам, взятым на рис. 13.3 в кружки.

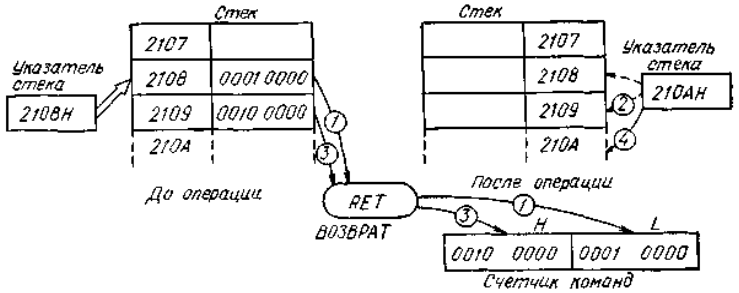


Рис. 13.3. Команда возврата из подпрограммы

1. Вершина стека (адрес 2108Н) извлекается, ее содержимое передается в младший байт счетчика команд.
2. Указатель стека инкрементируется от 2108Н до 2109Н.
3. Вершина стека (теперь ее адрес 2109Н) извлекается, ее содержимое передается в старший байт счетчика команд.
4. Указатель стека инкрементируется от 2109Н до 210АН.

Теперь счетчик команд содержит 16-разрядный адрес (2010Н) следующей извлекаемой из памяти команды.

Рассмотрим программу которая преднозначена для образования циклов неопределенно долго и принимает новые наборы данных в каждом цикле. Она покидает цикл для процедуры аварийной сигнализации только в случае возникновения соответствующей ситуации. В этом случае оператор или другая программа выдает процедуру отработки аварийного состояния. Это показано на структурной схеме кружком.

На рис. 13.4,а приведен пример функциональной структурной схемы. В задачу МП входит ввести два отобранных числа, образовать их сумму и сохранить ее. Этой процедуре соответствуют три кадра в начале схемы. Затем сумма должна быть умножена на масштабный коэффициент и результат размещается в памяти. Это связано с тем, что сумма (в третьем кадре) должна быть восстановлена в аккумуляторе. Затем она проверяется знаком принятия решения. В том случае, если сумма равна или больше 10Н (16_{10}), программа выводится на сигнализацию тревоги, если меньше 10Н циклится и снова вводит выборку чисел. Такой тип программ мог бы быть использован в промышленных устройствах, когда сумма двух входов должна постоянно контролироваться; быть подвержена масштабированию, а затем размещена в памяти;

контролироваться по крайней мере один раз в секунду для определения, не приняла ли она опасного значения, и если да, то программа должна выдать аварийный сигнал АС.

На рис. 13.4,б представлена схема программы. Каждый кадр ее эквивалентен одной операции МП. Выделенная рамкой ячейка, представляет собой группу команд, называемую подпрограммой умножить. В табл. 13.2 представлена программа на ассемблере к рис. 13.4, б. Этот список привлекает только команды, которые составляют часть состава команд нашего типового МП (список, относящийся к подпрограммам умножения, на рисунке не приведен).

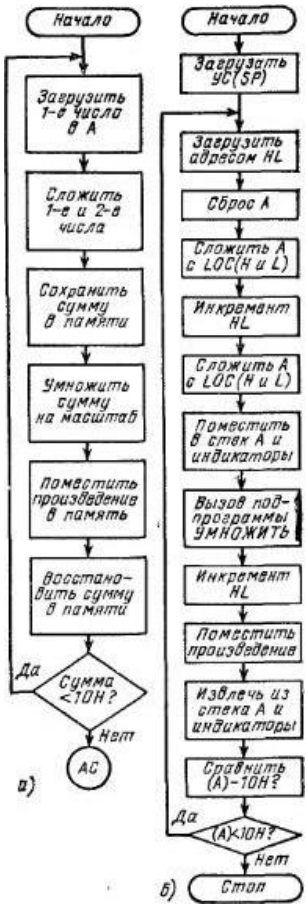


Рис. 13.4. Структурная схема программы контроля оборудования (а) и ее развитие (б)

На рис. 13.4,б представлена схема программы. Каждый кадр ее эквивалентен одной операции МП. Выделенная рамкой ячейка, представляет собой группу команд, называемую подпрограммой умножить.

В табл. 13.2 представлена программа на ассемблере к рис. 13.4, б. Этот список привлекает только команды, которые составляют часть состава команд нашего типового МП (список, относящийся к подпрограммам умножения, на рисунке не приведен).

Три первые команды (табл. 13.2) восстанавливают указатель стека, пару HL и аккумулятор А соответственно до 20С0Н, 2040Н и 00Н. Четвертая складывает содержимое (А) (т. е. 00Н) с содержимым ячейки памяти, на которую указывает пара HL (адрес 2040Н). Обратимся к рис. 13.5, который представляет собой

основную программу, данные, подпрограмму память стека, использованные в этом примере. Заметим, что содержимым памяти по адресу 2040H, сложенным с (A) является 05H. После выполнения четвертой команды содержимым (A) является 05H (00H + 05H=05H).

Таблица 13.2. Праграмма на ассемблере к рис.13.4,б

Метка	Мнемоника	Операнд	Комментарий
	LXI	*P*, 20C0H	Поместить 20C0H в указатель стека
START	LXI	*H*, 2040H	Поместить 2040H в пару *HL* (указатель адреса основной программы)
	XPA	*A*	Сброс аккумулятора в 00H
	ADD	*M*	Сложить (A) с содержнмым ячейки памяти 2040H (сложнть первое слагаемое с содержимым ячейкн памяти 2040H)
	INX	*H*	Инкрементировать пару *HL* до 2041H
	ADD	*M*	Сложить *A* с содержнмым ячейки памяти 2041H (сложить второе число с содержимым ячейкн памяти 2041H)
	PUSN	*PSW*	Поместнть в стек (A) и индикаторы
	CALL	MULTIPLY	Вызов подпрограммы MULTIPLY в ячейку памятн 2050H
	INX	*H*	Инкрементнровать пару *HL* до 2042H
	MOV	*M*, *A*	Поместить произведение в ячейку памяти 2042H
	POP	*PSW*	Извлечь нз стека и восстановить *A* и нндикаторы
	CPI	10H	Сравннть (A) и 10H, т. е. (A) — 10H. Если (A)<10H, *CY*=1; если нет, *CY*=0
	JC	START	Перейти к START (ячейка памяти 2003H), еслн *CY*=1; еслн нет, продолжать последовательно
	HLT		Остановить МП

Пятая команда основной программы инкрементирует пару HL. Шестая складывает содержимое памяти по адресу 2041H с (A). Эта ячейка памяти содержит 09H (см, рис.13.5), A содержит 05H, сумма в A после второго ADDM– 0EH является суммой двух выбранных чисел.

Следующей командой основной программы является PUSH PSW, ко сохраняет содержимое аккумулятора и индикатора. Это должно быть выпол потому что следующая команда вызова разрушит содержимое этих регистров, вып

подпрограмму умножения. Операция вызова помещает адрес команд последовательности основной программы (в нашем примере 200EH) в стек, переходит к адресу первой команды подпрограммы (адрес 2050H).

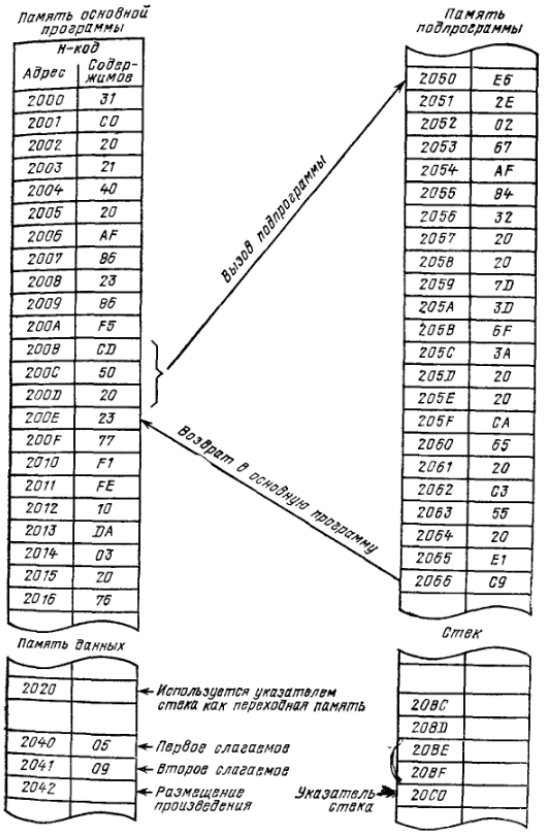


Рис.13.5 Воображаемая память программы, приведенной в табл. 13.2.

Этот вызов показан на рис. 13.4, б. Теперь рассмотрим команду вызов команду, которая просто умножает содержимое А на два. Подпрограмма в н примере осуществляет умножение 0EHx2=1CH, и произведение, помещенн аккумулятор к моменту возврата, будет lCH.

После возврата в основную программу адресом следующей последовате команды становится 200EH. Команда INX H инкрементирует пару HL, и произвед выполненное подпрограммой, помещается теперь в памяти по адресу 2042H ком MOV M, A.

Вспомним, что команды помещения в стек и извлечения из него парные. Т что мы поместили в стек содержимое аккумулятора и индикаторов командой PSW, мы их восстановим теперь командой POP PSW. Содержимое аккумул является не произведением, а суммой 0EH в нашем примере). Эта сумма тестирована последующей командой.

Важно отметить, что (0EH) отправлена в подпрограмму для обработки. Напр именно произведение было передано подпрограммой и, следовательно, размеще памяти. На жаргоне информатики эта операция составляет прохождение парамет данном случае использован именно регистр А, но в других случаях это могло бы выполнено посредством памяти.

Две следующие команды основной программы предназначены для тестиро суммы в аккумуляторе. Команда сравнить непосредственно выполняет опер вычитания 0EH—10H=FEH с использованием дополнительного кода числа -

Содержимое А меньше чем 10Н индикатор переноса CY установлен в 1. Ко ПЕРЕЙТИ, если перенос, проверяет индикатор, он установлен в 1, н МП перехо символическому адресу START, который является не чем иным, как началом команды LX1 H (адресация 2003H, см, рис. 13.5).

Для операции двоичного умножения могут быть использованы различные способы на базе команд типового МП. Вытекающие непосредственно из правил умножения методы приводят к повторяющемуся сложению. Рассмотрим типичное решение на примере выполнения операции $5x3=15_{10}$.

Множимое Множитель Произведение

5 x 3 = 15_{10}

При повторяющемся сложении эта операция выполнится иначе:

Множитель = 3 Произведение

5+5+5= 15_{10}

Множимое 5

В этом случае множимое повторяется слагаемым число раз, равное множителю, а результатом этой операции будет произведение. В нашем примере подпрограмма эффективно выполняет операцию сложения (вместо 0Ex2=1CH). Множимое складывается с самим собой (0E+0E=lCH) , что дает искомое произведение.

На рис. 13.6 приведена подробная структурная схема подпрограммы умножения, и каждый ее шаг соответствует одной команде программы на ассемблере, приведенной в табл. 13.2. Первый шаг на рис.13.6 обеспечивает хранение текущего содержимого пары HL операцией помещения в стек. Это характерно для многих подпрограмм — помещать в содержимое регистров МП, потому что они могут быть использованы во время выполнения подпрограммы, что изменит их содержимое. Совершенно очевидно, что содержимое пары HL будет восстановлено в конце программы командой извлечения из стека.

Три следующих шага восстанавливают регистры L, H и A. Регистр L - будет содержать множитель (здесь 02H) и он будет декрементирован до 00H в ходе программы умножения. Регистр H будет содержать множимое (здесь 0EH). т. е. сумму, посланную в подпрограмму основной программой. Аккумулятор здесь сбрасывается в 00H.

Пятый шаг соответствует сложению множимого (здесь 0EH) с A). Частичное произведение 0EH затем сохраняется во временной памяти по адресу 2020H, тогда как аккумулятор используется для декрементирования содержнмого L. Один раз множитель декрементируется от 02H до 01H и помещается в L. Затем частичное произведение восстанавливается в A.

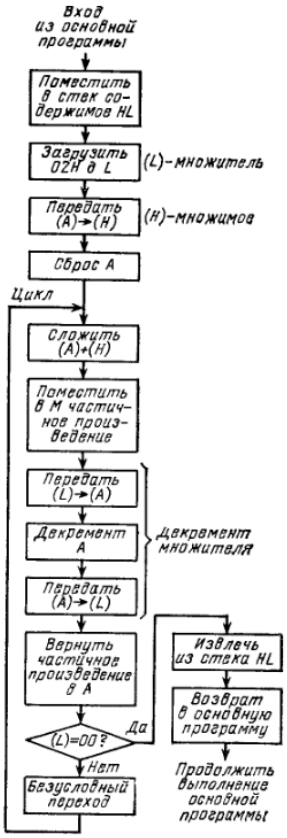


Рис. 13.6. Структурная схема подпрограммы умножения в задаче, приведенной на рис. 13.4.

На шаге принятия решения возникает вопрос: (L)=00? Если ответ отрицательный, программа переходит снова в цикл, если положительный — ветвится вправо. Содержимое пары HL восстанавливается командой извлечения из стека, и, наконец, команда возврата возвращает нас в основную программу. Согласно рис. 13.5 возврат осуществляется по адресу 200EH основной программы. В случае, когда множителем является 02H, программа дважды пройдет в последовательности сложить–поместить–передать–декрементировать —передать—загрузить перед возвратом в основную программу. Отметим, что окончательное произведение 1CH поступит в основную программу, оставаясь в аккумуляторе.

В табл. 13.2 приведена программа на ассемблере. Каждая команда соответствует одному шагу структурной схемы на рис. 13.6. Рекомендуем читателю попытаться проследить поток данных сначала по ней, а затем по программе, приведенной в табл. 13.2.

Используя команды вызова и возврата, следует быть очень внимательным и использовать их парами. Нужно следить за тем, чтобы требуемые регистры были правильно инициализированы, многократно проверять состояние указателя стека. Следует также убедиться, что парно используются команды загрузки и извлечения из стека, а также глубоко осознать способы обмена параметрами между подпрограммами и основными программами.

Таблица 13.2. Программа на ассемблере, соответствующая рис.13.6.

Метка	Мнемоника	Операнд	Комментарий
	PUSH	*H*	Поместить в стек содержимое регистров *H* и *L* для сохранения содержимого пары *HL*
	MVI	*L*, 02H	Поместить 02H (масштабный коэффициент) в *L*; 02H — множитель
	MOV	*H*, *A*	Передать (A)→(H); содержимое *H* — множимое
	XRA	*A*	Сброс *A*
LOOP	ADD	*H*	Сложить (H)+(A); сумму поместить в *A*
	STA	2020H	Поместить (A) в ячейку памяти 2020H (временная память)
	MOV	*A*, *L*	Передать (*L*) в *A*
	DCR	*A*	Декрементировать содержимое *A*
	MOV	*L*, *A*	Передать (A) в *L*
	LDA	2020H	Поместить содержимое ячейки памяти 2020H (временной памяти) в *A* (восстановление *A* из ячейки памяти 2020H)
	JZ	DONE	Перейти к DONE (ячейка памяти 2065H), если *Z*=1, т. е. декрементировано до 00H; если нет — продолжать последовательно
	JMP	LOOP	Перейти (всегда) к LOOP (адрес 2055H)
DONE	POP	*H*	Извлечь из стека содержимое *HL*
	RET		Возврат в основную программу

Тема 14 Прикладне програмування

Обработка структуры данных

Под структурами данных понимают наборы некоторым образом организованных данных. Программисту следует знать наиболее распространенные структуры данных, способы хранения их в памяти МП-систем и эффективного использования в прикладных программах. Наиболее простой структурой данных является одномерный массив, представляющий собой набор элементов данных одинаковой длины, размещённых в области смежных ячеек памяти с начальным / базовым / адресом, например, BASE/. Число элементов в массиве называется его длиной. Положение любого элемента в массиве характеризуется его порядковым номером, называемым индексом. Адрес элемента равен сумме базового адреса и индекса, умноженного на длину элемента в байтах. Целесообразно применять массивы, длина элементов которых равна степени двух. Тогда при вычислении адреса элемента операция умножения

заменяется сдвигами. Формирование и обработка массивов осуществляется циклическими программами. Примером обработки массива является программа поиска максимального числа в массиве однобайтовых целых чисел, рис.14.1.

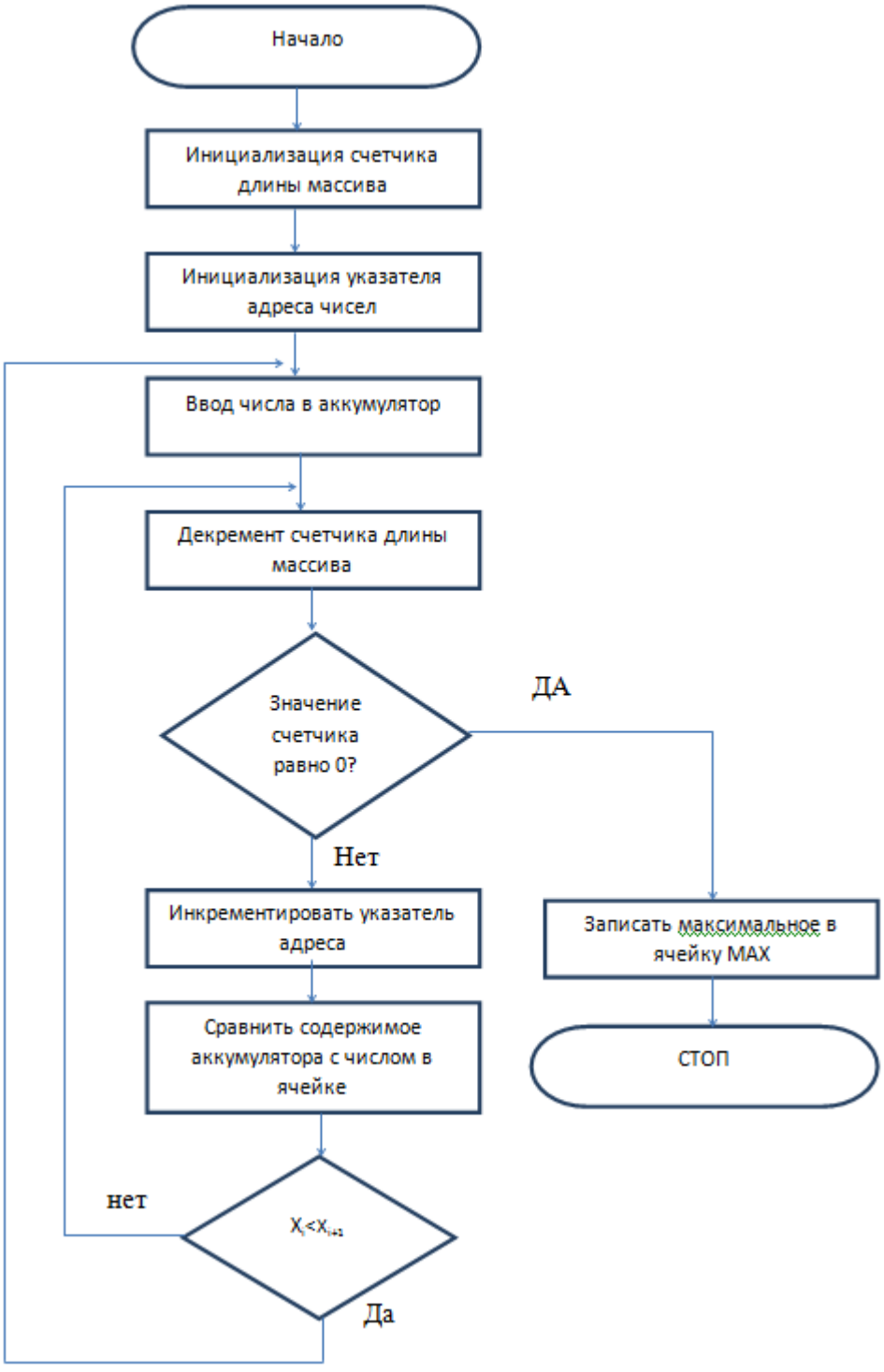


Рис.14.1 . Структурная схема поиска максимального числа.

Длина массива хранится в ячейке с условным адресом LENGT, адрес первого элемента - BASE, в качестве счетчика используется регистр B. Максимальное число направляется в ячейку памяти с адресом MAX.

Метка	Мнемоника	Операнд	Комментарий
	LDA	LENGT;	Инициализация счетчика длины массива
	MOV	B,A;	
	LXI	H.BASE;	Инициализация указателя адреса чисел
NEWMX:	MOV	A,M;	
NEXT	DCR	B;	Загрузка массива
			Проверка окончания цикла
	JZ	DONE;	Переход на конец программы, если флаг Z = 1
	INX	H;	Переход к следующему элементу
	CMP	M;	Сравнение его с максимумом
	JC	NEWMX;	Новый максимум
	JMP	NEXT;	Следующий элемент меньше
	STA	MAX;	Запись максимума ячейку
	HLT		Останов

Сначала первый элемент массива принимается в качестве максимального, а затем каждый последующий элемент сравнивается с ним. Если текущий элемент больше ранее найденного максимума , он замещает его в аккумуляторе. Таким образом, просматривается в цикле весь массив данных.

Сортировка

В некоторых прикладных задачах возникает необходимость сортировки, т.е. упорядочение элементов массива данных в возрастающем или убывающем порядке. Известно несколько методов решения данной задачи. Наиболее часто применяется т.н. "пузырьковая" сортировка - меньшие элементы перемещаются вверх, а большие - вниз. Сортировка заключается в последовательном просмотре массива и сравнения значений соседних элементов. Если значение элемента в ячейке X больше значения элемента в ячейке x-1 , производится обмен этих элементов (x)←→(x+1). Если (x)<(x+1), то обмена не происходит. После этого аналогично сравниваются элементы в ячейках x+1 и x+2. Процесс сравнения и обмена продолжается до достижения конца массива. Факт наличия хотя бы одного обмена фиксируется записью в некотором регистре называемом "индикатором обмена", не нулевого кода. После окончания просмотра массива анализируется значение индикатора обмена и, если в нем зафиксирован обмен, весь массив просматривается ещё раз. Сортировка заканчивается, когда при просмотре массива не было произведено ни одного обмена.

Структурная схема программы сортировки массива 8-битовых целых без знака приведена на рис. 14.2. Программа BS0RT имеет два параметра: начальный адрес массива записывается в регистрах HL, образуется длина массива, для чего начальный адрес вычитается из конечного. После этого начальный и адрес и длина массива запоминаются в стеке для использования на последующих

просмотрах. Затем производится сравнение значения двух соседних элементов массива и при необходимости осуществляется их обмен. В качестве индикатора обмена используется регистр С: в начале просмотра регистр С сбрасывается, а когда при просмотре массива производится обмен, в него загружается код FFH. После сравнения значений пары элементов выполняется декремент содержимого регистров D,E и происходит зацикливание до окончания одного просмотра. Наконец проверяется индикатор обмена - если регистр содержит, не ноль, инициируется новый просмотр массива.

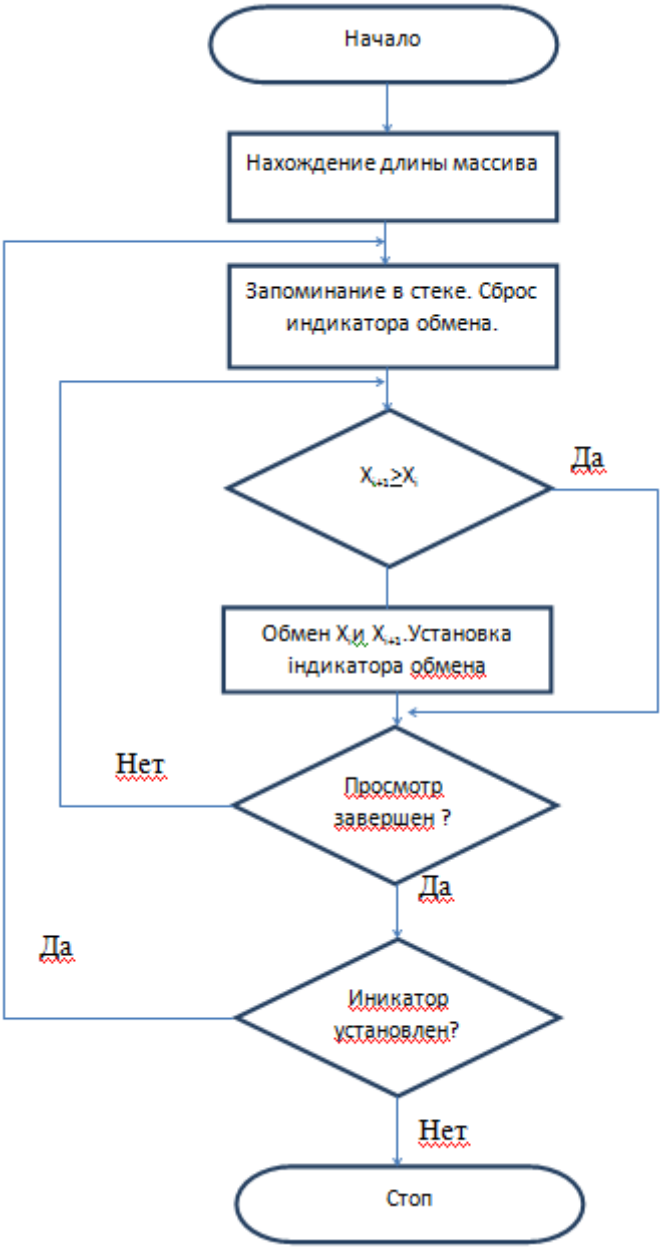


Рис. 14.2. Структурная схема программы сортировки

BSORT:	MOV	A,E	Образование в регистрах D,E длины массива
	SUB	L	
	MOV	E,A	
	MOV	A,D	
	SBB	H	
PASS:	PUSH	H	

	PUSH	D	
	MVI	C,00H	Сброс индикатора обмена
COMP:	MOV	A,M	Считывание значения
	INX	H	Указатель на X1+1
	CMP	M	Сравнение X1<X1+1
	JC	NEXT	Обмен не нужен
	JZ	NEXT	X1+1≥1
	MOV	B,M	Обмен значений элементов X1 и X1+1
	MOV	M,A	
	DCX	H	
	MOV	M,B	
	INX	H	Указатель на X1+1
	MVI	C,FFH	Установка индикатора обмена
NEXT:	DCX	D	Декремент счётчика элементов
	MOV	A,D	Анализ окончания просмотра
	ORA	E	
	JNZ	COMP	Просмотр закончен
	MOV	A,C	Да, проверка индикатора обмена
	ORA	A	
	POP	D	Восстановление начального адреса и длины
	POP	H	
	JNZ	PASS	Сортировка закончена?
	HLT		Да, стоп

Временные задержки

Организация процессов обработки информации в МП-системе осуществляется в виде последовательности простейших операций, проводимых во времени по сигналам специального синхронизирующего / тактового / генератора прямоугольных импульсов, характеризуемого частотой f или периодом синхронизации T= 1/f называемом тактом. Время, необходимое для считывания команды из памяти и ее исполнения, называется циклом команд и определяется числом тактов, необходимых для выполнения требуемой последовательности элементарных действий. Для определения времени выполнения команды нужно знать, какое число тактов содержится в цикле команд и чему равен период сигнала тактирования.

Временная задержка может быть сформирована программой, в которой некоторое множество команд не выполняет никаких операций, но занимает машинное время.

Для формирования временных задержек применяется метод, при котором в рабочий регистр загружается некоторое число и затем оно последовательно уменьшается на единицу до тех пор, пока оно не станет равным

0. Чем большую задержку необходимо получить, тем большее число нужно занести в этот регистр, рис 14.3.

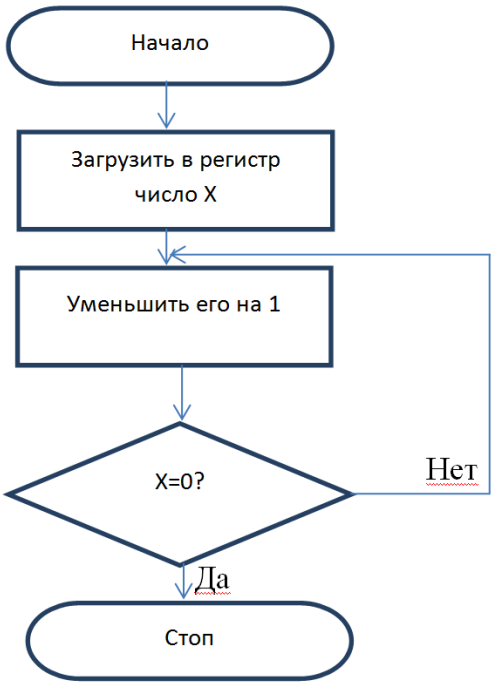


Рис. 14.3 Структурная схема задержки времени

Пример построения такой подпрограммы приведен ниже.

Метка	Мнемоника	Операнд	Комментарий
DELAY:	MVI	B, 0XN ;	Загрузить в регистр в число 0XN
FORW:	DCR	B;	Уменьшить на единицу
	JNZ	FOR W;	Повторить процесс, если результат не 0
	RET		Возврат в основную программу

Подпрограмма не работоспособна до тех пор, пока не будет задано значение 0XH. Зная команды и время на их выполнение можно определить число 0XH, обеспечивающее требуемую задержку. При расчете нужно учесть, что данная подпрограмма будет вызываться командой CALL DELAY.

Согласно Приложению и учитывая, что для микропроцессора К-580 обычно T = 0,5 мкс (f = 2 МГц) определяется время выполнения команд

CALLDELAY - 17 тактов 0,5 = 8,5 мкс
MVI B.X - 7тактов 0,5 = 3,5 мкс
DCR B - 5тактов 0,5 = 2,5 мкс
JNZ FORW - 10 тактов 0,5 = 5 мкс
RET - 10 тактов 0,5 = 5 мкс

Командные CALLDELAY, MVIB, 0X и RET используются в программе 1 раз, команды DCR B и JNZ FORW используются многократно в зависимости от значения 0XH. Чтобы получить задержку времени равную 100 мкс команды

DCR B и JNZ FORW должны повторится столько раз, чтобы этот процесс выполнялся за 100-17 = 83 мкс. Время выполнения этих команд равно 2,5+5,0=7,5 мкс. Задержку 83 мкс нельзя получить повторением этой пары, т.к. 83 не делится на 7,5 без остатка. Для корректировки времени используется команда выполняемая за 4 такта и время цикла равно 2 мкс. Окончательно подпрограмма имеет вид:

Метка	Мнемоника	Операнд	Комментарий
DELAY:	MVI	B, 10H;	Загрузить в регистр B число 10
FORW:	DCR	B;	Уменьшить содержимоеB на 1
	JNZ	FORW;	Повторить процесс, пока результат не 0
	NOOP		Доводка времени задержки до заданного значения
	NOOP		
	NOOP		
	NOOP		
	RET		Возврат в основную программу

При организации больших временных задержек используют в качестве счетчиков регистровые пары, строят программы со вложенными циклами.

Логические функции

В системах управления технологическими процессами и объектами необходимые логические связи между входными и выходными параметрами могут решаться или аппаратными средствами с помощью комбинационных логических схем или программно /гибкое или настраиваемое управление/. Считается, что применение микропроцессорных систем, по сравнении с системами с жесткой логикой обеспечивает большую гибкость, более низкую стоимость и меньше время разработки. Однако использование таких систем требует оценки их быстродействия.

Программируемая логическая система реализует заданную логическую функцию путем последовательного сравнения комбинации входных сигналов с комбинациями, заданными таблицей истинности логической функции, так называемыми масками. Результат сравнения используется для перехода в состояние, соответствующее выполнению заданного условия /операнды совладают/, либо для продолжения операций сравнения со следующими наборами /масками/. В качестве примера рассмотрим решение логической функции F= f(A,B,C) представленной следующей таблицей истинности.

Положим, что входные сигнал A.B.C представляются состояниями второго (b_2), первого(b_1) и нулевого(b_0) битов слова, поступающего с порта 01_{16}. Выходной сигнал F реализуется третьим битом управляющего слова, записанного в ячейке памяти и UPR и выдаваемого в порт вывода 05_{16}.

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Программа реализации функции F представлена ниже:

Метка	Код	Операнд	Комментарий
	IN	01H	Ввод входных сигналов
	ANI	07H	Сброс, битов b_7, b_6, b_5, b_4, b_3 в ноль
	CPI	08H	Сравнение с маской согласно 4-ой строки таблицы
	JZ	FF1	Если равенство, переход к формированию F= 1
	CPI	05H	Если нет ,продолжение сравнения
	JZ	FF1	См. выше
	CPI	06H	Сравнение с 7-ой строкой
	JZ	FF1	См. выше
	LDA	UPR	Формирование нуля в третьем бите управляющегослова
	ANI	F7H	
	JMP	WW	Переход к выводу управляющего слова
FF1:	LDA	UPR	Формирование управляющего слова
	ORI		
WW:	OUT		Вывод управляющего слова
	HLT		

Следует учитывать, что при необходимости изменения вида логической функции, коррекция в систему вносится только путём изменения значения масок. Время реализации заданной логической функции в системе с программным управлением может оказаться больше, чем в системе с жестким управлением.