## §4. УЧЕБНАЯ МОДЕЛЬ МИКРОКОМПЬЮТЕРА

В предыдущем разделе были кратко изложены наиболее важные черты реально существующего простого процессора ЭВМ. Конечно, на самом деле все устроено заметно сложнее, поэтому мы ограничились достаточно поверхностным знакомством. Более современные процессоры устроены ещесложнее.

Существует альтернативный подход - изучение сложного объекта на упрощенной модели, сохраняющей все его наиболее существенные черты. Именно такой подход использован в данном разделе. Рассмотрение фундаментального материала, освобожденного от второстепенных технических деталей, помимо простоты имеет и еще одно преимущество - полученные знания не так быстро «стареют».

Прием, состоящий в объяснении принципов работы ЭВМ на базе учебной модели, возник довольно давно. Имеется целый ряд учебных моделей: «Кроха», «Нейман», «Малютка» и другие (описание которых можно найти в рекомендуемой в конце главы дополнительной литературе).

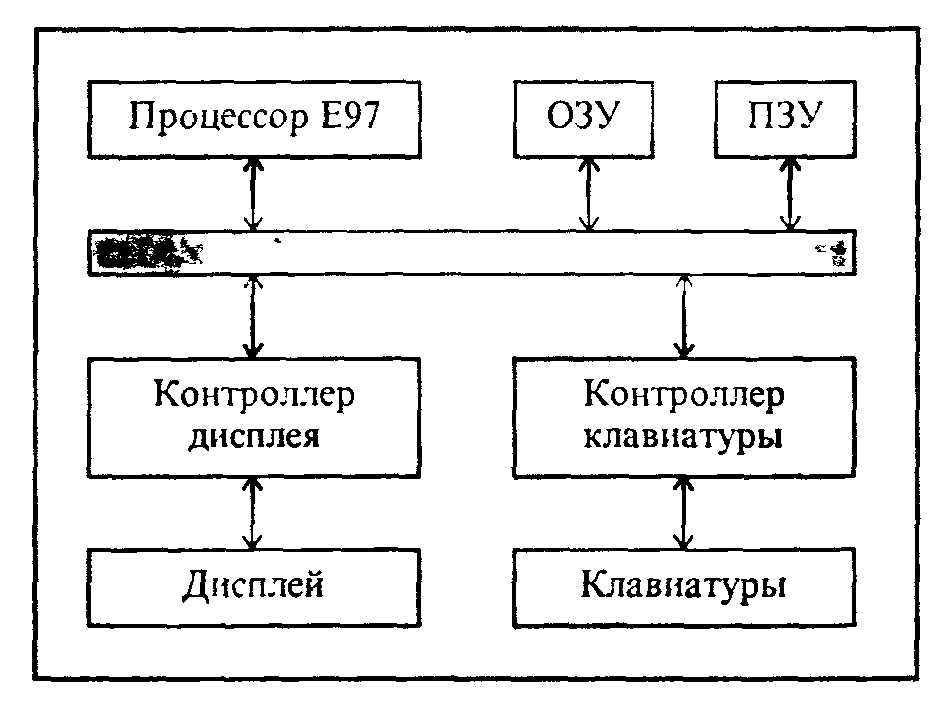
В данном параграфе рассматривается учебная модель микропроцессорной ЭВМ четвертого поколения, предложенная преподавателем Пермского педагогического университета Е.А.Ереминым. Модель носит название «Е97» В нее включены наиболее характерные черты устройства современных компьютеров. Модель не повторяет полностью ни один реальный компьютер, хотя дает достаточно полное представление об его устройстве. В «Е97» нет излишних технических деталей, он достаточно прост и содержит только то, что необходимо для изучения.

«Е97» может работать как с числовыми, так и с нечисловыми данными, отражает общепринятую байтовою организацию памяти, способен обрабатывать массивы информации (поскольку имеет современные методы адресации), реалистически отражает работу с периферийными устройствами современного компьютера. Существенно, что наряду с теоретической моделью компьютера создана программа-эмулятор, позволяющая имитировать «Е97» на различных компьютерах, в том числе IBM-совместимых. Ряд непринципиальных ограничений и нереализованных возможностей, указанных ниже, относятся именно к этой программе-эмулятору.

### 4.1. СТРУКТУРА УЧЕБНОГО МИКРОКОМПЬЮТЕРА

В состав учебного микрокомпьютера входят следующие устройства (рис. 4.16):

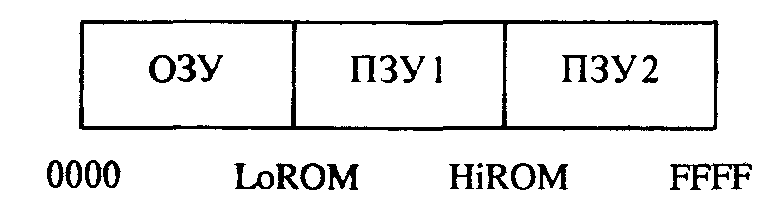
центральный процессор, память двух видов (ОЗУ и ПЗУ), а также два наиболее важных внешних устройства (клавиатура для ввода информации и дисплей для вывода; оба устройства, как принято в современных ЭВМ, подключены через обеспечивающие согласование контроллеры).



***Рис.*** *4.16.* Схема устройства учебного компьютера «Е97»

Главным блоком компьютера служит 16-разрядный процессор «Е97», способный работать как с двухбайтовыми словами, так и с отдельными байтами. Таким образом, он может оперировать с данными разной длины. Познакомившись с тем, как «Е97» обрабатывает разные типы информации, читатель легко сможет в будущем обобщить логику «один байт или много» на случай большей разрядности процессора.

В процессоре имеются внутренние регистры памяти, при помощи которых реализован метод косвенной адресации к ОЗУ. Очевидно, что полное 16-разрядяое адресное пространство «Е97» позволяет напрямую адресовать до 64 кбайт памяти; для учебной ЭВМ это более чем достаточно. Поэтому реально существующей памяти будут соответствовать лишь некоторые диапазоны адресов (рис. 4.17). Для первой программной реализации модели «Е97» приняты следующие значения констант: LoROM *=* 4000, HiROM =4180.



*Рис. 4.17.* Структура памяти микрокомпьютера «Е97» (LoROM - нижняя граница ПЗУ 1. HiROM-верхняя граница ПЗУ 1, совпадающая с началом ПЗУ 2)

В «Е97», как и в реальном компьютере, существует память двух видов -оперативная (ОЗУ) и постоянная (ПЗУ) В первой хранится текущая информация (т.е. программа и данные) по решаемон задаче, причем она может как считываться, так и записываться. Во второй, предназначенной только для считывания, содержатся разработанные при проектировании ЭВМ подпрограммы наиболее важных и часто используемых действий, среди которых важное место занимают алгоритмы обмена с внешними устройствами.

Видеопамять в «Е97» размещается в контроллере дисплея; для простоты модели будем считать, что видеопамять не входит в адресное пространство процессора. Доступ к видео-ОЗУ осуществляется путем обращения к внешнему устройству.

Как уже отмечалось, минимальной адресуемой ячейкой памяти в современных ЭВМ является байт. Все байты в памяти «Е97» пронумерованы и их 16-разрядные номера находятся в пределах от 0000 до FFFF. Реальные адреса должны попадать в выделенные на рис. 4.17 области. При обращении к любому другому адресу происходит аварийное прекращение выполнения программы.

Байты в памяти могут объединяться в слова, которые для данного компьютера состоят из двух соседних байтов (у современных процессоров обычно из четырех, но для понимания основных принципов это не очень существенно). По традиции .примем, что слово адресуется наименьшим из номеров байтов, причем этот адрес соответствует младшему байту слова. Для удобства изображения содержимого ОЗУ на экране дисплея будем считать, что адрес слова *всегда является четным*

Для наглядности проиллюстрируем все эти важные положения примером (рис. 4.18). Слово с адресом 40 имеет значение FFOO, а слово с адресом 42 - 1234; адреса 41 и 43 для слов запрещены. Обратите внимание на то, что байт 40 имеет нулевое значение, а байт с адресом 41 - значение FF. Иначе говоря, байты слова хранятся в памяти «задом наперед» - *сначала младший, а затем старший.* Именно так хранится слово в большинстве компьютеров.

Рассмотрим теперь роль ПЗУ. Его наличие дает целый ряд преимуществ, важнейшими из которых являются следующие.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 40 | 41 | 42 | 43 |  | 41 | FF | 00 | 40 |
| 00 | FF | 34 | 12 |  | 43 | 12 | 34 | 42 |
| Представление в байтах | | | |  | Представление в словах | | | |

*Рис. 4.18.* К вопросу о представлении информации в «Е97»

Во-первых, в современных ЭВМ не всякое элементарное действие реализуется в виде команды процессора - многие из них выполняются по программе. В «Е97», например, по программе происходит нахождение остатка от деления, вычисление модуля, перевод целого числа из двоичной системы в десятичную. Наличие таких подпрограмм освобождает модель от необходимости вводить в систему команд учебного процессора разные нестандартные и несуществующие в реальных процессорах команды типа «вывод числа на экран в десятичной форме» и ей подобные.

Во-вторых, наличие ПЗУ позволяет не конкретизировать алгоритм работы с внешними устройствами на уровне системы команд процессора, а перенести их в ПЗУ (именно так делается и в настоящих компьютерах). ПЗУ такого рода носит специальное название **BIOS** (от английского Basic Input Output System - **базовая система ввода-вывода**) и обслуживает конкретные типы внешних устройств, входящих в состав машины BIOS имеет стандартные точки входа, к которым и обращается все программное обеспечение при необходимости произвести обмен информации с внешним устройством.

Завершая обсуждение ПЗУ, отметим еще одну деталь. В данной учебной ЭВМ оно условно делится на две части, названные на рис. 4.17 ПЗУ1 и ПЗУ2. ПЗУ1 доступно для изучения и представляет собой настоящие программы в кодах процессора «Е97». Другая часть долговременной памяти - ПЗУ2 - напротив, «скрыта» от просмотра. Определены лишь входные точки и их назначение; в основном это операции ввода. Деление на ПЗУ1 и ПЗУ2 является чисто техническим приемом и не имеет никаких аналогий в реальных ЭВМ (наличие «закрытого» ПЗУ2 просто позволяет легче и быстрее осуществить программный имитатор учебного микрокомпьютера).

Перейдем к рассмотрению контроллеров внешних устройств В учебном микрокомпьютере они фигурируют в виде портов ввода-вывода Каждому из устройств соответствует два таких порта: **регистр данных и регистр состояния.** Через первый происходит обмен информацией, а второй позволяет этот обмен синхронизировать. Если в знаковый (старший) бит младшего байта регистра состояния записана единица, то это означает, что устройство к обмена готово.

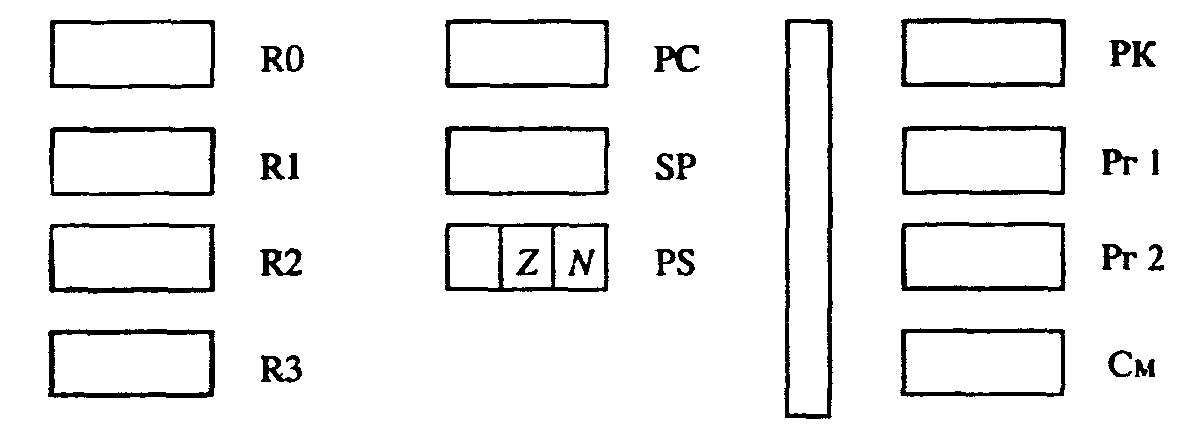
Отметим, что контроллеры современных периферийных устройств имеют обычно более двух портов. Например, в IBM-совместимых компьютерах для работы со стандартным печатающим устройством используется три порта - к уже названным добавляется регистр управления. Ничего принципиально нового наличие дополнительных портов не вносит.

Такова внутренняя структура учебного микрокомпьютера. Рассмотрим теперь устройство самого процессора, каким он видится программисту. «Е97» состоит из семи 16-разрядных регистров. 4 регистра общего назначения **RO - R3,** счетчика адреса команд **PC** (Program Counter), указателя стека **SP** (Stack Pointer) и регистра состояния процессора **PS** (Processor Status), в котором мы будем использовать только два младших бита N и Z. «Поведение» этих управляющих битов согласно общепринятым закономерностям следующее:

N = 0 - результат ≥ 0, N = 1 - результат < 0,

Z = 0 - результат ≠ 0, Z = 1 - результат = 0.

Здесь «результат» означает результат последней из выполненных арифметических операций. Все эти объекты изображены на рис. 4.19.



***Рис.*** *4.19,* Внутреннее устройство учебного процессора «Е97»

Кроме уже названных, в «Е97» имеется ряд *внутренних* регистров, которые процессор использует при исполнении операций. Это **регистр команд РК**, предназначенный для хранения кода исполняемой в данный момент команды; **регистры операндов Рг1 и Рг2**, куда считываются исходные данные; **сумматор См**, в котором производится требуемое в команде действие и получается результат. Мы не можем непосредственно изменять содержимое этих «служебных» регистров, но в данном учебном компьютере они доступны для наблюдения. Вертикальная заштрихованная полоса на рисунке разделяет программно-доступные (слева) и программно-недоступные (справа) регистры.

### 4.2. СИСТЕМА КОМАНД

Перейдем к самому важному - системе команд, которые умеет выполнять учебный процессор. Как мы уже знаем, машинная команда состоит из операционной и адресной частей: первая указывает, что надо сделать с данными, а вторая - где их взять и куда поместить результат. В этом разделе мы будем говорить, в основном, об операционной части, лишь коротко упоминая об адресной; в последней нас пока, главным образом, будет интересовать число операндов (адресов). Вопросам, связанным с подробностями адресации данных, будет посвящен следующий пункт.

Итак, рассмотрим структуру команды «Е97» (рис. 4.20). В зависимости от конкретной операции, ее формат может иметь некоторые особенности, но в наиболее полной форме он состоит из четырех частей по 4 бита каждая (см. рис. 4.20, б): **модификатор МОД,** **код операции КОП** и два **операнда ОП1 и ОП2.** Назначение КОП и операндов было описано в предыдущем параграфе. Что же касается МОД, то он указывает варианты реализации команды, например, адресовать байт или слово, по каким управляющим битам переходить и др.

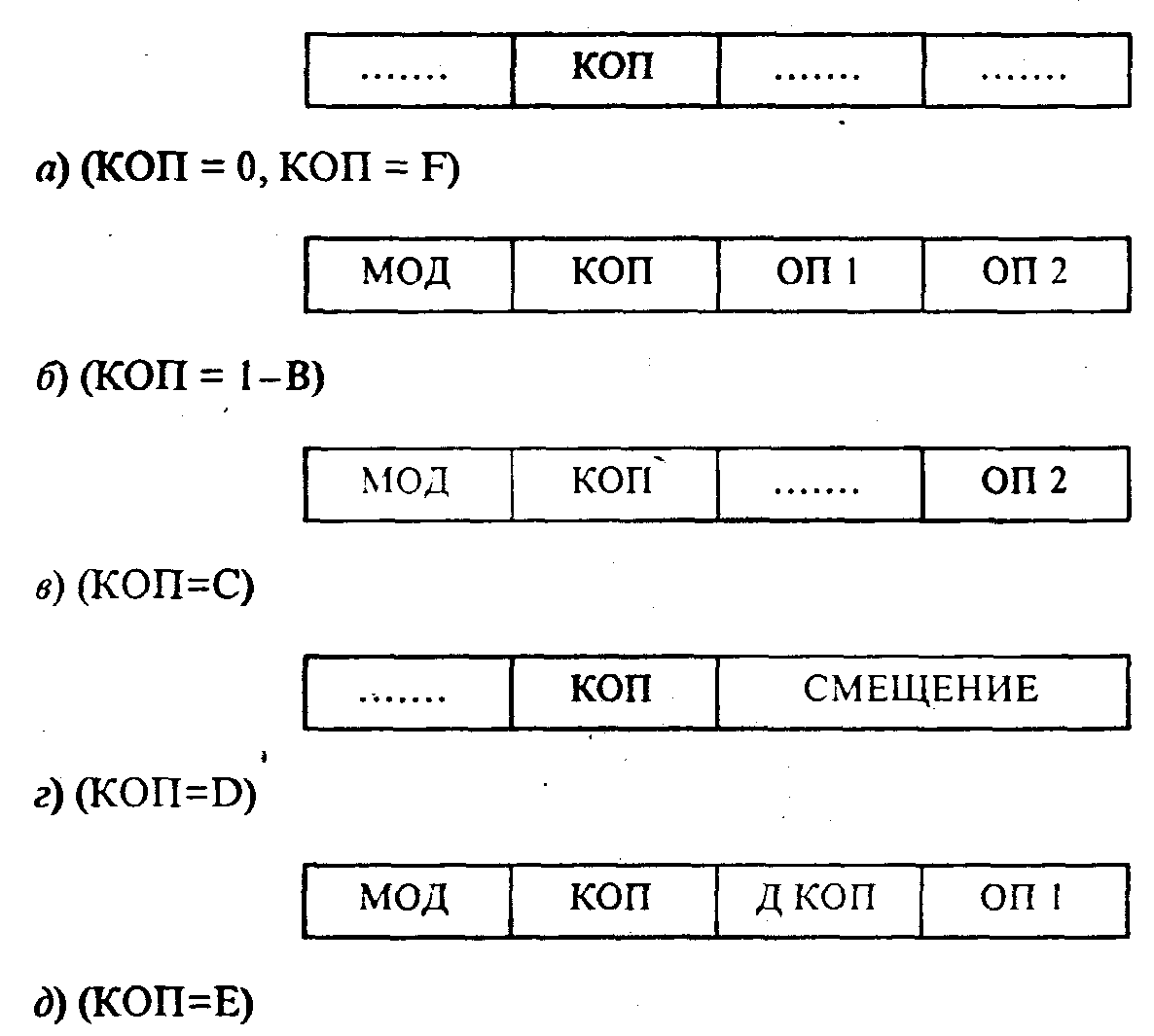
Наиболее простой формат команд из всех возможных, имеют две **-нет операции** (ее код 0) и **останов** (код F). Как видно из рис. 4.20, а, в этих командах задействован только КОП. остальные 12 бит значения не имеют. Основная масса команд, коды которых заключены в интервале от 1 до В. являются двухадресными и соответствуют уже упоминавшемуся ранее рис. 4.16, б. К ним относятся:

1 - перепись, 5 - умножение, 9 - исключающее «ИЛИ»,

2 - сложение, 6 - деление, А - ввод из порта,

3 - вычитание, 7 - логическое «И», В - вывод из порта.

4 - сравнение, 8 - «ИЛИ»,



*Рис. 4.20.* Форматы команд учебного процессора «Е97»

Операция переписи выполняется достаточно тривиально: информация считывается из ОП1 и копируется в ОП2. Совершенно аналогично работают ввод и вывод из порта, с той лишь разницей, что в качестве одного из операндов указывается номер порта. Все остальные двухадресные команды с кодами 2-9 представляют собой определенные действия над двумя данными, выполняемые по универсальной схеме

ОП2 *операция* ОП 1 => ОП2

Например, по команде деления процессор извлекает ОП2, делит его на ОП1 и результат помещает вместо первоначального значения ОП2.

Некоторую особенность имеет команда сравнения. При ее исполнении производится вычитание ОП2 - ОП1, но результат никуда не записывается. («Тогда для чего же вычитать?» - спросите вы? - «Исключительно ради установки управляющих битов, которые в дальнейшем могут быть проанализированы командами условных переходов».)

Арифметические действия осуществляются над целыми числами и результатыих -целочисленные в формате «16-битные целые числа со знаком». При делении получается целая часть частного. Что же касается операций над вещественными числами, то они могут быть реализованы программным путем (соответствующие программы могут быть, например, помещены в одно из ПЗУ).

Перейдем теперь к рассмотрению команд переходов. Как мы уже знаем, они бывают *абсолютные,* когда значение адреса для перехода задается явно, и *относительные.* когда адрес следующей команды вычисляется путем значения текущего программного счетчика и указанного в теле команды смещения. В соответствии с этим в «Е97» есть два типа переходов с кодами операций С и D; их форматы представлены на рис. 4.20, *в, г.*

Начнем с абсолютного перехода, код которого равен С. Если процессор встретит в программе команду из двух слов

1C0D

0056

то следующей будет выполняться команда с адресом 56. Иными словами, адрес перехода берется из самой команды. В команде 1C0D старшая шестнадцатеричная цифра - модификатор, соответствующий безусловному переходу; С - код операции; 0 - неиспользуемая цифра; D - операнд, указывающий что адрес перехода хранится в следующем слове, составляющем с 1C0D единое целое. Подробности такого способа адресации - в следующем пункте.

По-другому обстоит дело сотносительным переходом, код которого D. В качестве примера возьмем команду

1D06

Для определенности будем считать, что эта команда находится в памяти по адресу 42. В строгом соответствии с основным алгоритмом работы процессора, после выборки рассматриваемой команды счетчик адреса команд PC автоматически увеличивается до 44. Затем, выполняя расшифрованную команду перехода, процессор прибавит к текущему содержимому PC смещение 06 и тем самым осуществит переход на адрес 44 + 6 = 4А. Обратите внимание, что итоговый адрес в случае относительного перехода зависит от расположения команды перехода в ОЗУ.

При отрицательном смещении возможно получение адреса меньшего, чем исходный. При обсуждении команд перехода мы незаметно включили в работу модификатор команд. Для переходов его роль проста и наглядна: МОД показывает, по какому условию осуществляется переход. Таблица всех используемых в «Е97» значений модификаторов выглядит так:

0 - возврат из подпрограммы;

1 - безусловный переход;

2 - N = 0 (≥0);

3 - N = 1 (<0);

4 - Z *=* 0 (≠0);

5 - Z *=* 1 (=0);

6 - N=1 orZ=l (≤0);

'7 - N=0andZ=0(>0);

9 - вызов подпрограммы.

Становится очевидным, что рассмотренные в обоих предыдущих примерах команды с МОД = 1 являются наиболее простым вариантом перехода - безусловным.

Для работы с условными переходами следует твердо запомнить следующее правило:

***«Если анализируемое условие справедливо, т. е. состояние управляющих признаков совпадает с требуемым, то переход происходит. В противном случае никаких действий не производится, а значит переход просто игнорируется и процессор, как обычно, выбирает следующую команду».***

Кстати, так обстоит дело и в языках высокого уровня.

И еще об одном виде перехода следует поговорить особо - о **переходе с возвратом** или о **переходе к подпрограмме**, как его часто называют. На практике подпрограммы (процедуры) и функции играют очень важную роль. Подпрограммы полезны, когда *в разных местах* программы требуется выполнить *одни и те лее действия.* В этом случае имеет смысл оформить повторяющиеся действия в виде подпрограммы, а затем просто вызывать ее в нужных местах. В каком-то смысле это похоже на публикацию текстов песен, когда припев пишется один раз, а в дальнейшем просто ставится ссылка на него в виде слова «ПРИПЕВ».

Наиболее важное отличие перехода к подпрограмме от безусловного перехода состоит в том, что требуется иметь возможность вернуться из подпрограммы в то же самое место, откуда она была вызвана. Применительно к процессору возможность возврата означает запоминание где-нибудь значения программного счетчика PC; для возврата достаточно будет просто восстановить в PC сохраненное значение.

Для обеспечения возможности вложения друг в друга подпрограмм необходимо уметь сохранять не одно значение PC, а несколько. Для реализации такого механизма памяти используется стек, который идеально подходит для любой вложенности конструкций и при этом требует наличия всего одного выделенного регистра-указателя стека SP.

Рассмотрим конкретный пример. Пусть в некотором месте программы находится команда из двух слов

9C0D

0030

Исполнять эту команду процессор будет так. Прежде всего, он уменьшит SP на 2 и запомнит по полученному адресу текущее содержимое PC (вспомните, что счетчик к этому времени уже будет показывать на следующую команду). Затем последует переход по адресу 30, считанному ранее из второго слова команды. Таким способом мы попадем в подпрограмму, надежно спрятав в стеке адрес основной программы, куда нужно вернуться. Обсудим теперь, как произойдет возврат.

В конце любой подпрограммы должна стоять команда 0С00 или 0D00. Встретив ее. процессор извлечет из стека занесенное туда ранее значение и поместит его в PC. При этом он увеличит SP на два, освободив ненужную более ячейку памяти в стеке. Таким образом, прерванное **на** время выполнение основной программы продолжится с нужного места.

*Примечание.* Команды с кодами 0С и 0D тождественныи их младшие 8 бит неиспользуются.

Для того, чтобы лучше разобраться в описанном механизме, попробуйте самостоятельно проанализировать ситуацию, когда одна подпрограмма вызывает другую.

Займемся последней группой команд процессора, для которой КОП = Е. Прежде всего, почему такому КОП соответствует группа, а не одна команда? Все дело в том, что эти команды одноадресные, и освобождающиеся от одного из операндов 4 бита можно использовать для задания номера операции в этой группе. Назовем полученные биты дополнением к КОП - ДКОП. В итоге получим формат, приведенный на рис.4.20, д). Из рисунка видно, что код операций для всех одноадресных операций состоит из двух шестнадцатеричных цифр, причем первая из них всегда Е. Познакомимся с этими командами более подробно.

Команда с кодом Е1 выполняет над единственным операндом ОП1 логическую операцию НЕ, т.е. заменяет нулевые биты единицами и наоборот (инверсия).

Команды с кодами Е2 - Е9 обеспечивают работу со стеком. Так, при коде операции Е2 ОП1 заносится в стек, а при ЕЗ - считывается оттуда. Например, вот как можно поменять местами содержимое регистров R1 и R2 с использованием стека:

0000 0Е21 записать R1 в стек

0002 0Е22 записать R2 в стек

0004 0Е31 считать значение из стекав R1

0006 0Е32 считать значение из стека в R2

0008 0F00 останов

Команды Е4 и Е5 позволяют изменять значение SP на величину ОП1, что часто бывает полезно при работе со стеком, например, при освобождении в нем сразу нескольких «этажей». По кодам Е6 и Е7 можно задать новое значение SP и прочитать его текущее значение в ОП1. Наконец, наиболее экзотические из этой группы команды Е8 и Е9 сохраняют в стеке и восстанавливают для последующего анализа регистр состояния процессора PS. Эти команды замечательны тем, что обрабатывают вполне определенный операнд, поэтому содержимое ОП1 в команде значения не имеет; *договоримся заполнять его нулем.*

Осталось рассмотреть последнюю группу команд - сдвиги. Их коды ЕА - ЕС. Все они осуществляют сдвиг кода в ОП1 на один разряд влево или вправо в зависимости от значения ДКОП. Полезно помнить, что сдвиг влево эквивалентен умножению, а вправо - делению на два.

Команда ЕС, называемая арифметическим сдвигом, отличается от обычного сдвига ЕВ тем, что старший знаковый разряд при арифметическом сдвиге сохраняет свое значение, например:

Исходное значение ОП 1: 1111 0000 1111 0000

Результат команды ЕВ: 0111 1000 0111 1000

Результат команды ЕС: 1111 1000 0111 1000.

Арифметический сдвиг бывает полезен для деления отрицательных чисел, так как в этом случае автоматически сохраняется признак знака минус - единица в старшем разряде.

В данной модели при сдвиге, приводящем в выходу старшей или младшей цифры за пределы разрядной сетки, соответствующая информация теряется. В реальных процессорах для ее сохранения существует специальный управляющийбит, называемый «битом переноса».

### 4.3. АДРЕСАЦИЯ ДАННЫХ

Теперь, когда мы знаем практически все об операционной части команды, можно заняться адресной. Посмотрим, какими способами могут представляться операнды ОП1 иОП2.

Начнем с того, что под кодирование каждого операнда всегда отводится четыре двоичных разряда. Из них старшие два будут всегда задавать тип адресации данных. а младшие - его конкретизировать. В большинстве случаев два младших бита будут просто представлять собой номер регистра, с участием которого осуществляется адресация.

**Старшая** «половинка» модификатора операнда, соответствующая типу адресации, может содержать четыре неповторяющиеся двоичные комбинации:

**00 - *регистровый*** метод адресации: операнд является содержимым указанного регистра;

**01** - метод ***косвенной*** адресации: операндом является содержимое ячейки ОЗУ, адрес которой задан в указанном регистре;

**10** - резерв; в будущей версии здесь будет реализован индексный метод адресации;

**11** - адресация ***по счетчику адреса команд*** PC: операнд извлекается с использованием информации, входящей в команду (более подробные объяснения приведены ниже).

Рассмотрим перечисленные способы адресации подробнее.

Наиболее простым является регистровый метод, который мы фактически уже использовали в примерах предыдущего раздела. Учитывая, что этому методу соответствуют нулевые значения старших битов, полный код операнда совпадает с номером регистра: двоичная комбинация 0000 соответствует R0, 0001 - R1 и т.д.

В качестве данных для операции используется информация, содержащаяся в указанном регистре. Например, если Rl = 3, а R2 = 5, то в результате выполнения команды

**0212**:R2+R1=>R2

получится R2 = 8.

При косвенной адресации код операнда выглядит несколько сложнее: косвенное обращение по регистру R0 имеет вид 0100 (т.е. 4), по Rl - 0101 (5) и т.д. Содержимое указанного регистра при этом служит не операндом, а его адресом в ОЗУ. Рассмотрим команду

**0263**: R3 + (R2) => R3

где скобки у R2 символизируют косвенную адресацию. Пусть содержимое R2 в данный момент равно 30, а R3 = 6. Примем также, что в ячейке памяти с адресом 30 хранится число 10. Тогда процессор «Е97», выполняя команду, к имеющемуся в R3 значению 6 прибавит число из ячейки 30, на которую указывает R2, и результат операции - 16 - занесет в R3.

И, наконец, рассмотрим способы адресации по программному счетчикуPC**.** Поскольку в этом случае регистр, по которому производится адресация, уже однозначно определен, освобождаются два младших бита операнда, которые можно использовать для других целей. В связи с этим удается получить четыре различных способа адресации по счетчику:

**11 00** - резерв;

**11 01** - операндом служит константа, входящая в команду;

**11** **10** - операнд извлекается из ячейки ОЗУ, адрес которой входит в команду;

**1111**-резерв.

Итак, в «Е97» существует два метода адресации по PC, соответствующих шест-надцатеричным кодам **D** и **Е** в качестве операнда. Изучим их на примерах. Команда, состоящая из двух слов

02D1

0020

выполняет операцию Rl + 20 => Rl следующим образом: к текущему значению Rl прибавляется извлеченное из команды число 20 и результат помещается в Rl. Если первоначальное значение Rl было, скажем, F0, то в результате операции в Rl запишется 110.

Рассмотрим еще одну команду с адресацией по PC:

021Е

0020

выполняющую операцию (20) + Rl => (20) так: к числу, хранящемуся в ячейке памяти 20, прибавляется значение Rl. В случае, когда Rl = F0, a (20) = 40, ответ будет: (20)= 130.

На этом рассказ о методах адресации можно было бы и закончить, если бы не наличие в командах модификатора, хранящегося в первой цифре шестнадцатерич-ного представления команды. Его значение также может существенно влиятьнаизвлечение данных и запись результата.

Во всех приведенных в данном пункте примерах МОД = 0 и, следовательно, не оказывает влияния на расшифровку и выполнение операции. Если же он имеет ненулевое значение, необходимо дополнительно учитывать и этот фактор.

Модификатор состоит из четырех битов, причем два старших отвечают за «переключением бант/слово, а два младших - за особый способ представления данных, называемый «**короткой константой**». Многие команды оперируют с небольшими константами, значения которых не превышает по модулю 15. Такие констангы вполне «уместятся» в четырех отведенных под ОП1 битах. Для реализации такой возможности в «Е97» используются два младших бита модификатора -0-й и 1-й. Признаком, по которому процессор распознает «короткую константу», служит наличие единицы в первом разряде модификатора. В этом случае содержимое ОП1 рассматривается как двоичное число, а 0-й бит модификатора служит его знак ом,например:

25A3:R3\* 10=>R3

или

3653:R3/(-5)=>R3

Из приведенных примеров видно, что операции с «короткой константой» занимают всего одно слово. Данный способ оказывается, таким образом, экономичнее описанного чуть раньше метода адресации по PC (вспомните, что там константа хранится во втором слове команды).

Наличие операций с «короткой константой» позволяет упростить систему команд процессора. Например, в программах часто требуются команды, увеличивающие или уменьшающие данные на единицу. Любой процессор имеет для этой важной ситуации специализированные команды, обозначаемые чаще всего INC и DEC. В «Е97» эти команды получаются как частный случай операций с «короткой константой».

Во всех предыдущих рассуждениях мы молчаливо полагали, что процессор работает только с 16-разрядными словами. Но это не всегда так - иногда требуется выполнить те или иные действия над байтами, например, при обработке текста. Старшие биты модификатора как раз и предусмотрены для оказания режима байт/слово: 3-й бит соответствует ОП1, а 2-й - ОП2. Если соответствующий бит модификатора установлен в единицу, то процессор оперирует с байтом, иначе - со словом. Как всегда, наличие двух битов порождает четыре варианта:

**00** ОП1-слово, ОП2-слово;

**01** ОП1-слово, ОП2-байт;

**10** ОП 1 - байт, ОП2 - слово;

**11** ОП1-байт, ОП2-байт.

Наиболее простыми и очевидными являются комбинации слово - слово и байт -байт, когда оба операнда и результат имеют одинаковый тип. Смешанные комбинации слово - байт и байт - слово более экзотичны и нуждаются в особом рассмотрении. Не вдаваясь в тонкости, можно считать, что в двухадресных командах модификатор всегда равен либо **0 для операции со словами,** либо **С для байтов.**

### 4.4. РАБОТА С ВНЕШНИМИ УСТРОЙСТВАМИ

Как уже говорилось ранее, модель «Е97» отражает наиболее простые методы обмена с внешними устройствами из тех, что свойственны современным компьютерам.

Как и в реальных микропроцессорах, «общение» с периферийными устройствами осуществляется через порты ввода-вывода. Для учебной ЭВМ принята наиболее простая модель, когда каждому из имеющихся внешних устройств соответству ет два порта - **порт данных и порт состояния.** Можно считать, что каждый порт представляет собой 16-разрядный регистр, находящийся в контроллере. Через порт данных происходит обмен информацией. Порты состояния позволяют управлять процессом обмена данными, так как предоставляют процессору сведения о текущем состоянии внешнего устройства. Реальная синхронизация ввода-вывода является достаточно сложной проблемой и существенно зависит от типа периферийного устройства. Поэтому в «Е97» регистр состояния устроен максимально просто: в нем имеется единственный используемый бит - **бит готовности**. В обсуждаемом учебном компьютере бит готовности помещается в седьмом разряде регистра состояния.

Все порты являются 16-разрядными, но реальная информация всегда располагается в их младшем байте. Содержимое битов с 8-го по 15-й в выходные порты формально заносится, но никакого влияния на устройство вывода не оказывает.

Каждому порту соответствует свой номер (адрес). В «Е97»из шестнадцати возможных используются следующие порты:

**0** - порт состояния клавиатуры (только чтение);

**1** - порт данных клавиатуры (только чтение);

**2** - порт состояния дисплея (только чтение);

**3** - порт данных дисплея (только запись).

Стандартный алгоритм обмена с использованием портов ввода-вывода выглядит следующим образом. Считывается порт состояния и проверяется значение *знакового бита* его *младшего* банта. Эта операция повторяется до тех пор, пока бит готовности не будет установлен внешним устройством в единицу, что является сигналом процессору к началу обмена. Только после него процессор может записать информацию в порт данных, если речь идет об устройстве вывода, или считать их из порта, если это порт ввода.

Вот как может выглядеть правильная программа вывода одиночного символа из R0 на дисплей (именно так она реализована в ПЗУ «Е97»):

0А21 порт2=>R1

Е401 сравнить с 0 R1b

2DFA если *≥* 0, PC - 6 (к повторению проверки)

0В03 R0 => порт 3

Первые три команды считывают и проверяют бит готовности в порту 2, а последняя команда обеспечивает собственно вывод требуемого символа на дисплей.

Обратите внимание на команду с «короткой константой» 0,которая ***обязательно*** должна проверять знаковый разряд***младшего байта,*** а не всего слова в целом.

Мы подробно рассмотрели программирование процедуры вывода символа; ввод осуществляется совершенно аналогично.

### 4.5. ПРИМЕРЫ ПРОГРАММ

*Пример 1.* Вычислить полную поверхность параллелепипеда со сторонами А, В и С. Считать, что исходные значения находятся в ячейках ОЗУ. Результат также поместить в ячейку памяти.

*Решение.* Полная поверхность параллелепипеда вычисляется по формуле

S=2\*(A\*B+A\*C+B\*C)

Для упрощения программы выражение удобно представить в виде S = 2 \* [А \* (В + С) + В \* С]

*Программа 134*

0000 01ЕО (22) => R0 В => R0

0002 0022

0004 0101 R0 => Rl В => Rl

0006 02Е0 R0 + (24) => R0 В + С

0008 0024

000А 05Е0 R0 \* (20) => R0 A \* (В + С)

000С 0020

000Е 05Е1 Rl \* (24) => Rl В \* С

0010 0024

0012 0210 R0 + Rl => R0 A \* (В + С) + В \* С

0014 0200 R0 + R0 => R0 2 \* [A \* (В + С) + В \* С]

0016 010Е R0 => (26) результат => S

0018 0026

001A 0F00 останов

……….

0020 0002 А

0022 0003 В

0024 0004 С

0026 0034 S

Примечание. Не забывайте, что ответ 34 в памятиЭВМ представленв шестнадцатеричной системе. В десятичной, как и положено, получится 52.

*Пример 2.* Организовать ввод латинских букв таким образом: программа принимает латинскую букву и обрабатывает ее так, чтобы она всегда была заглавной.

*Решение.* Главная «хитрость» решения состоит в том, чтобы понять, чем отличаются заглавные буквы от строчных. Для этого из таблицы ASCII, приведенной в главе 1, выберем наугад одну из букв и выпишем в двоичном виде коды заглавного и строчного символов. Например, для буквы R получим

R 0101 0010

r 0111 0010

Повторив аналогичные действия еще для нескольких букв, сделаем вывод, что ***коды заглавных ч строчных букв отличаются единственным битом - пятым,*** если считать младший бит за нулевой. Запомним этот факт,поскольку он справедлив для любого современного компьютера.

Итак, для нашей задачи требуется сохранить все биты, кроме пятого, значит потребуется выполнить логическую операцию **И** с константой

1101 1111 =DF,

после чего любой код «потеряет» ненужный нам бит, сохраняя все остальные.

Еще одной особенностью решения является активное использование подпрограммиз ПЗУ. Поскольку при вызове подпрограмм обязательно используется стековая память, то должен быть корректно определен указатель стека SP; программа начинает работу с установки указателя стека.

*Программа 135*

0000 0E6D 26 => SP установка указателя стека

0002 0026

0004 9C0D вызов п/п 40FE ввод символа (без эхо-печати!)

0006 40FE

0008 0101 R0 => Rl сохранить введенный символ

000A 07D0 DF and R0 => R0 сделать букву заглавной

000C 00DF

000E 9C0D вызов п/п 4088 вывести результат

0010 4088

0012 0F00 останов

*Пример 3.* Найти максимум из трех чисел, находящихся в регистрах Rl, R2 и R3. Переписать наибольшее из них в R0.

*Решение.* Сначала максимальное из чисел в Rl и R2 занесем в R0. После этого, если R3 окажется больше R0, «исправим положение», переписав в качестве ответа R3.

*Программа 136*

0000 0412 сравнить R2 с Rl

0002 3D04 если < 0, то РС=РС+4 к записи Rl (к 0008)

0004 0120 R2 => R0 запомнить R2

0006 1D02 РС=РС+2 обход второй ветви (к 000A)

0008 0110 Rl => R0 запомнить Rl

000A 0403 сравнить R3 с R0

000C 3D02 если < 0, то РC=РС+2 к выходу (к 0010)

000E 0130 R3 => R0 запомнить R3

0010 0F00 останов

*Пример 4.* Найти сумму первых 100 натуральных чисел.

Говорят, такую задачу некогда решил в уме юный Гаусс, когда учился в школе. Мы будем решать задачу «в лоб», т.е. честно суммировть с помощью компьютера. *Решение.* Поместим в R3 обрабатываемое в данный момент число N (меняется от 1 до 100), а в R0 - результирующую сумму S. Зададим им начальные значения и будем циклически добавлять к S текущее значение N. Признаком окончания цикла будет ситуация, когда N >100.

*Программа 137*

0000 2113 1 => R3 1 => N

0002 2100 0 => R0 0 => S

0С04 0230 R0 + R3 => R0 S = S + N

0006 2213 R3 + 1 => R3 N=N+1

0008 04D3 сравнить R3 с 100 сравнить N и 100

000A 0064

000C 6DF6 если ≤0, то PC=PC+F6 если N≤100,

к повторению (000E + FFF6 = 0004)

000E 0F00 останов

*Пример 5.* Вывести на экран весь латинский алфавит от А до Z. *Решение.* Разместим в RO выводимый латинский символ, первоначальное значение которого будет «А» (код 65 = 41h). Вывод будем осуществлять обращением к соответствующей подпрограмме ПЗУ. Для перехода к следующему символу алфавита достаточно прибавить 1 к коду текущего символа (очень похоже на переход к следующему числу в предыдущем примере). Остается только проверить, не выходит ли вновь полученный символ за латинский алфавит, т.е. не превышает ли его код 5Ah («Z»), и, если ответ будет «да» (превышает), то закончить процедуру.

*Программа 138*

0000 0E6D 26 => SP установка указателя стека

0002 0026

0004 01D0 41 => R0 код первого символа

0006 0041 («А»)

0008 9C0D вызов п/п 4088 вывод символа

000А 4088

000С 2210 R0 + 1 => R0 следующий символ

000Е 04D0 сравнитьR0 с 5А его код ≤ «Z»?

0010 005А («Z»)

0012 6DF4 если ≤0, то PC=PC+F4 к повторению (0008)

0014 0F00 останов

*Пример 6.* В памяти, начиная с адреса 001А, хранится некоторый текст, длина которого равна 15 (Fh) байтам Определить номер первого, совпадающего с образцом, символа в тексте. При отсутствии требуемого символа результат равен 0 (это практически полный аналог функции POS в Паскале).

*Решение* Поместим в R1 счетчик символов, в R2 - адрес текущего символа. Затем будем сравнивать каждый символ текста с образцом в R0 и в случае совпадения прервем выполнение цикла. При несовпадении будем продолжать цикл до теx пор, пока счетчик не превысит Fh, т.е. не станет равным 10h Если цикл завершится по выполнении этого условия, то символ-образец найти не удалось и в качестве ответа в R1 следует занести 0.

*Программа 139*

0000 2111 1 => R1 номер символа

0002 01D2 1А => R2 адрес начала текста

0004 001А

0006 С460 сравнить R0b с (R2)b сравнить символ с образцом

0008 5D0C если = 0, то РC=РС+2 выход при совпадении (к 0016)

000А 2211 R1 + 1 => R1 увеличить номер символа

000С 2212 R2 + 1 => R2 вычислить следующий адрес

000Е 04D1 сравнить R1 с 10 текст не закончился?

0010 0010

0012 4DF2 если ≠ 0, то PC=PC+F4 нет - к повторению (к 0006)

0014 2101 0 => R1 при отсутствии символа - 0

0016 0F00 останов

0018 0000

001А 4854 «ТН» текст

001С 5349 «IS» «THIS IS MY TEXT»

001E 4920 «I»

0020 2053 «S »

0022 594D «MY»

0024 5420 «Т»

0026 5845 «EX»

0028 0054 «Т»

**4.6. НЕКОТОРЫЕ СПРАВОЧНЫЕ ДАННЫЕ ПО Е-97**

***А. Система команд процессора***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| МОД | коп | ОП1 | ОП 2 | Пояснения |
| х | 0 | X | X | Нет операции |
| х | 1 | X | X | оп l => oп 2 |
| х | 2 | X | X | oп 2 + oп 1 => oп 2 |
| х | 3 | X | X | oп 2 – oп l =>oп 2 |
| х | 4 | X | X | oп 2 + oп 1 (сравнить) |
| х | 5 | X | X | oп 2 \* oп l *=>* oп 2 |
| х | 6 | X | X | oп 2/oп l => oп 2 |
| X | 7 | х | х | оп 2 AND oп 1 => oп 2 |
| X | 8 | х | х | oп 2 OR oп l => оп 2 |
| X | 9 | х | х | oп 2 XOR oп l => oп 2 |
| X | А | х | х | порт! => oп 2 |
| X | В | х | х | oп 1 => порт 2 |
| X | С | х | х | Переход по адресу |
| X | D | X | х | Переход по смещению |
| X | Е | \* | х | (Одноадресные операции) |
| X | F | X | х | Останов |

|  |  |  |  |
| --- | --- | --- | --- |
| МОД | коп | ОП1 | Пояснения |
| х | Е1 | х | NOT oп l |
| х | Е2 | х | оп 1 => стек |
| х | ЕЗ | х | стек => oп l |
| х | Е4 | х | SP + oп l => SP |
| х | Е5 | х | SP – oп 1 => SP |
| х | Е6 | х | Oп 1 => SP |
| х | Е7 | х | SP => oп l |
| х | Е8 | 0 | PS => стек |
| х | Е9 | 0 | стек => РS |
| х | ЕА | X | Сдвиг влево oп l |
| х | ЕВ | X | Сдвиг вправо oп l |
| х | ЕС | X | Арифметический сдвиг вправо oп l |

***Б. Двоичные коды модификатора МОД для переходов***

0000 - возврат из подпрограммы

0001 - безусловный переход

0010 - N=0 (>=0)

0011 - N=l (<0)

0100 - Z=0 (<>0)

0101 - Z=l (=0)

0110 - N=1 or Z=l (<=0)

0111 - N=0 and Z=0 (>0)

1001 - вызов подпрограммы.

***В. Важные кочанды с «короткой константой»***

XX 10 0001 0000 ХХХХ - очистить oп l

ХХ 10 0010 0001 ХХХХ - +1 в оп 1

XX 11 0010 0001 ХХХХ - \*\*\*\*

или

ХХ 10 0011 0001 ХХХХ - -1 из oп l

XX 10 0100 0000 ХХХХ - сравнить 0 с oп 1

XX 11 0101 0001 ХХХХ - \*(-1) оп1

ХХ 10 0111 0001 ХХХХ -oп l MOD 2 = => oп l

***Г. Кодирование операндов***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0000 | R0 |  | 1000 | резерв |
| 1 | 01 | R1 |  | 01 | резерв |
| 2 | 10 | R2 |  | 10 | резерв |
| 3 | 11 | R3 |  | 11 | резерв |
| 4 | 0100 | (R0) |  | 1100 | резерв |
| 5 | 01 | (R1) |  | 01 | константа |
| 6 | 10 | (R2) |  | 10 | адрес ОЗУ |
| 7 | 11 | (R3) |  | 11 | резерв |