2.7 Машинные операции

Компьютерная программа, записанная на машинном языке, состоит из **машинных инструкций**, каждая из которых представлена в машинном коде в виде т. н. опкода — двоичного кода отдельной операции из системы команд машины. Для удобства программирования вместо числовых опкодов, которые только и понимает процессор, обычно используют их условные буквенные мнемоники. Набор таких мнемоник, вместе с некоторыми дополнительными возможностями (например, некоторыми макрокомандами, *директивами*), называется языком ассемблера.

Каждая модель процессора имеет свой собственный набор команд, хотя во многих моделях эти наборы команд сильно перекрываются. Говорят, что процессор *A совместим* с процессором *B*, если процессор *A* полностью «понимает» машинный код процессора *B*. Если процессоры *A* и *B* имеют некоторое подмножество инструкций, по которым они взаимно совместимы, то говорят, что они одной «архитектуры» (имеют одинаковую архитектуру набора команд).

Набор команд — соглашение о предоставляемых архитектурой средствах программирования, а именно:

- определённых типах данных,
- инструкций,
- системы регистров,
- методов адресации,
- моделей памяти,
- способов обработки прерываний и исключений,
- методов ввода и вывода.

Система команд представляется спецификацией соответствия (микро)команд наборам кодов (микро)операций, выполняемых при вызове команды, определяемых (микро)архитектурой системы. (При этом на системах с различной (микро)архитектурой может быть реализована одна и та же система команд. Например, Intel Pentium и AMD Athlon имеют почти идентичные версии системы команд x86, но имеют радикально различный внутренний дизайн.)

Базовыми командами являются, как правило, следующие:

1. Команды передачи данных (перепись), копирующие информацию из одного места в другое.

2. Арифметические операции, которым фактически обязана своим названием вычислительная техника. Конечно, доля вычислительных действий в современном компьютере заметно уменьшилась, но они по-прежнему играют в программах важную роль. Отметим, что к основным арифметическим действиям обычно относятся сложение и вычитание (последнее в конечном счете чаще всего тем или иным способом также сводится к сложению). Что касается умножения и деления, то они во многих ЭВМ выполняются по специальным программам.

3. Логические операции, позволяющие компьютеру анализировать обрабатываемую информацию. Простейшими примерами могут служить сравнение, а также известные логические операции И, ИЛИ, НЕ (инверсия). Кроме того к ним часто добавляются анализ отдельных битов кода, их сброс и установка.

4. Сдвиги двоичного кода влево и вправо. Для доказательства важности этой группы команд достаточно вспомнить правило умножения столбиком: каждое последующее произведение записывается в такой схеме со сдвигом на одну цифру влево. В некоторых частных случаях умножение и деление вообще может быть заменено сдвигом (вспомните, что дописав или убрав ноль справа, т.е. фактически осуществляя сдвиг десятичного числа, можно увеличить или уменьшить его в 10 раз).

5. Команды ввода и вывода информации для обмена с внешними устройствами. В некоторых ЭВМ внешние устройства являются специальными служебными адресами памяти, поэтому ввод и вывод осуществляется с помощью команд переписи.

6. Команды управления, реализующие нелинейные алгоритмы. Сюда прежде всего следует отнести условный и безусловный переход, а также команды обращения к подпрограмме (переход с возвратом). Некоторые ЭВМ имеют специальные команды для организации циклов, но это не обязательно: цикл может быть сведен к той или иной комбинации условного и безусловного переходов. Часто к этой же группе команд относят немногочисленные операции по управлению процессором -типа «останов» или НОП («нет операции»). Иногда их выделяют в особую группу.

С ростом сложности устройства процессора увеличивается и число команд, анализирующих состояние управляющих битов и воздействующих на них. Здесь для примера можно назвать биты режима работы процессора и биты управления механизмами прерываний от внешних устройств.

В последнее время все большую роль в наборе команд играют команды для преобразования из одного формата данных в другой (например, из 8-битного в 16-битный и т.п.), которые заметно упрощают обработку данных разного типа, но в принципе могут быть заменены последовательностью из нескольких более простых команд.

Оптимальными в различных ситуациях являются разные способы построения системы команд:

- Если объединить наиболее часто используемую последовательность микроопераций под одной микрокомандой, то надо будет обеспечивать меньше микрокоманд. Такое построение системы команд носит название **CISC** (Complex Instruction Set Computer), в распоряжении имеется небольшое число составных команд.
- С другой стороны, это объединение уменьшает гибкость системы команд. Вариант с наибольшей гибкостью — наличие множества близких к элементарным операциям команд. Это **RISC** (Reduced Instruction Set Computer), в распоряжении имеются усечённые, простые команды.

- Ещё большую гибкость системы команд можно получить, используя **MISC**-подход, построенный на уменьшении количества команд до минимального и упрощении вычислительного устройства обработки этих команд.

Подводя итог, еще раз подчеркнем, что основной набор команд довольно слабо изменился в ходе бурной эволюции ЭВМ. В то же время способы указания адреса расположения информации в памяти претерпели значительное изменение и заслуживают особого рассмотрения.

В принципе, машинный код можно рассматривать как примитивный язык программирования или как самый низкий уровень представления скомпилированных или ассемблированных компьютерных программ. Хотя вполне возможно создавать программы прямо в машинном коде, сейчас это делается редко в силу громоздкости кода и трудоёмкости ручного управления ресурсами процессора, за исключением ситуаций, когда требуется экстремальная оптимизация. Поэтому подавляющее большинство программ пишется на языках более высокого уровня и транслируется в машинный код компиляторами. Машинный код иногда называют **нативным кодом** (также **собственным** или **родным кодом** — от англ. *native code*), когда говорят о платформенно-зависимых частях языка или библиотек.

Программы на интерпретируемых языках (таких как Бейсик или Python) не транслируются в машинный код; вместо этого они либо исполняются непосредственно интерпретатором языка, либо транслируются в псевдокод (байт-код). Однако интерпретаторы этих языков (которые сами можно рассматривать как процессоры), как правило, представлены в машинном коде.

В некоторых компьютерных архитектурах поддержка машинного кода реализуется ещё более низкоуровневым слоем программ, называемых микропрограммами. Это позволяет обеспечить единый интерфейс машинного языка у всей линейки или семейства компьютеров, которые могут иметь значительные структурные отличия между собой, и облегчает перенос программ в машинном коде между разными моделями компьютеров. Примером такого подхода является семейство компьютеров IBM System/360 и их преемников: несмотря на разные шины шириной от 8 до 64 бит и выше, тем не менее, у них общая архитектура на уровне машинного языка.

Использование слоя микрокода для реализации эмулятора позволяет компьютеру представлять архитектуру совершенно другого компьютера. В линейке System/360 это использовалось для переноса программ с более ранних машин IBM на новое семейство — например, эмулятор IBM 1401/1440/1460 на IBM S/360 model 40.

2.8 Машинные команды

В общем виде машинная команда имеет структуру, изображенную на рис. 2.8.1.

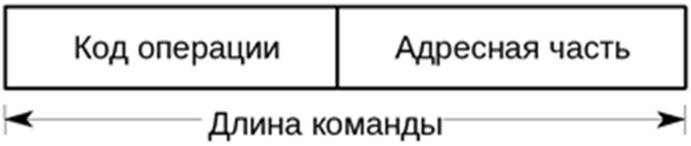


Рис. 2.8.1 Общая структура команд

Таким образом, команда состоит из операционной и адресной частей. Эти части, в свою очередь, могут состоять из нескольких полей (особенно адресная).

Операционная часть – содержит код, который задает вид операции (сложение, умножение, передача и т.д.).

Адресная часть – содержит информацию об адресах операндов и результата операции, а в некоторых случаях и следующей команды.

Структура команды – определяется составом, назначением и расположением полей в команде.

Формат команды – это ее структура с разметкой номеров разрядов, определяющих границы отдельных полей команды.

Задача выбора оптимальных структур и форматов команд при проектировании новых ЭВМ является одной из важнейших, поскольку от правильности ее решения зависит быстродействие и производительность ЭВМ.

Проблема состоит в том, что, с одной стороны, в команде желательно разместить максимум информации о выполняемой операции. С другой стороны, для упрощения аппаратуры и повышения быстродействия ЭВМ длина формата команды должна быть согласована с длиной обрабатываемых машинных слов, составляющей обычно 16-32 бита (для того чтобы можно было использовать для хранения и обработки операндов и команд одни и те же аппаратные средства). Формат команды должен быть, по возможности, короче, укладываться в машинное слово или полуслово, а для ЭВМ с коротким словом (8-16 бит) быть малократным машинному слову. Решение проблемы выбора оптимального формата команды значительно усложняется в микроЭВМ, работающих с коротким словом.

В абсолютном большинстве случаев ОП универсальных ЭВМ является адресной. Это значит, что каждой хранимой в ОП единице информации (байту, слову, двойному слову) ставится в соответствие специальное число – адрес, определяющий место ее хранения в памяти. В современных ЭВМ различных типов *минимальной*адресуемой в памяти единицей информации в большинстве случаев является*один байт*, т.е. 8 бит с 9-м контрольным разрядом. Иногда бывает и полубайт, т.е. 4 разряда и даже один бит. Более крупные единицы информации – слово, двойное слово и т.д. образуются из целого числа байт. В зависимости от способа хранения информации в ОП их адресом считается адрес старшего или младшего байта.

В общем случае разрядность машинного слова может определяться разрядностью АЛУ процессора, разрядностью шины данных, шириной выборки ОП и другими факторами. Наиболее часто разрядность машинного слова соответствует разрядности операндов, которые наиболее эффективно

обрабатываются процессором. Например, процессор I80386 имеет 32-разрядные АЛУ и 32- разрядную шину данных. Однако разработчики устройств на базе этого процессора за машинное слово выбрали 16-разрядный двоичный код при ширине выборки ОП 1 байт. Следует иметь в виду, что ширина выборки ОП – это техническая характеристика БИСов памяти, а байт, слово, двойное слово и т.д. – логические единицы информации, которые формируются контроллером памяти и операционной системой и обычно кратны ширине выборки. При рассмотрении процессов передачи и обработки информации внутри ЭВМ в большинстве случаев оперируют именно этими логическими единицами.

2.9 Типы адресации

Процесс эволюции ЭВМ и расширение сферы их целевого использования, совершенствование аппаратного и программного обеспечения ЭВМ привели к созданию машинных команд очень сложной структуры. Однако, если отбросить детали построения команд конкретных процессоров, возможные структуры машинных команд сводятся к пяти основным типам (рис. 2.9.1).

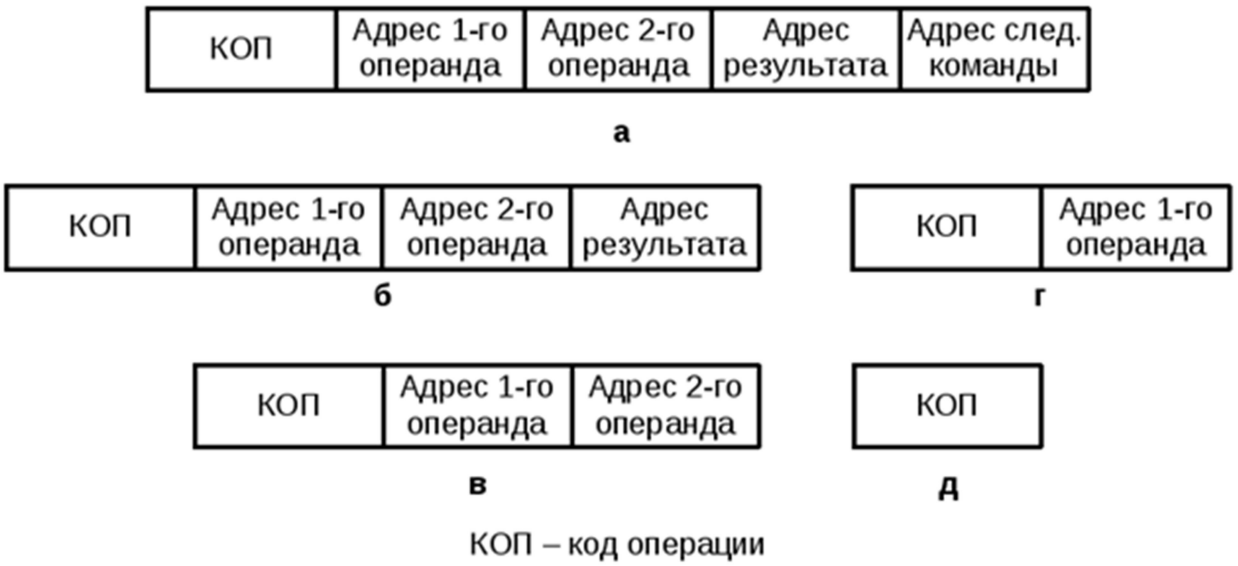


рис. 2.9.1 Структуры команд а – четырех адресная, б – трехадресная, в – двухадресная, г – одноадресная, д - безадресная

1. *Четырехадресная структура*

Такая команда содержит наиболее полную информацию о выполняемой операции, так как она содержит поле кода операции и четыре адреса для указания ячеек памяти двух операндов, участвующих в операции, ячейки, в которую помещается результат операции, и ячейки, содержащей следующую команду. Такой порядок выборки команд называется *принудительным*. Он использовался в первых моделях ЭВМ, имеющих небольшое число команд и очень незначительный объем ОП.

Рассмотрим длину такой команды применительно к ЭВМ, имеющей порядка 200 команд и объем памяти порядка 16 Мбайт. В этом случае длина КОП будет:

$N_{КОП}= \log_2 200$, т.е. 8 разрядов ($2^8= 256$).

Для обеспечения доступа ко всем ячейкам памяти потребуется

$$N_{адр} = \log_2 16777216 = \log_2 16 \cdot 1024 \cdot 1024 = \log_2 2^4 \cdot 2^{10} \cdot 2^{10} = \log_2 2^{24} = 24 .$$

Таким образом, длина четырехадресной команды составит

$$N_{ком} = 8 + 24 * 4 = 104 \ разряда .$$

Полученная длина команды оказывается недопустимо большой, поэтому в современных ЭВМ такая структура команд не используется.

2. *Трехадресная структура*

Можно построить ЭВМ так, что после выполнения команды по адресу K (команда занимает L ячеек памяти) выполняется команда по адресу K+L. Такой порядок выборки команд называется *естественным*. Он нарушается только специальными командами передачи управления. При естественном порядке выборки адрес следующей команды формируется в устройстве, называемом*счетчик адреса команд*. В этом случае команда становится трехадресной.

3. *Двухадресная структура*

В большинстве случаев непосредственно перед выполнением операции операнды помещаются во внутренние регистры процессора. Можно построить аппаратную часть процессора таким образом, что результат операции будет всегда помещаться в фиксированный регистр процессора, например на место первого операнда. В этом случае команда будет двухадресной, поскольку адрес результата подразумевается (рис. 2.9.1, в).

4. *Одноадресная структура*

В одноадресной команде подразумеваемые адреса имеют уже и результат операции, и один из операндов. Для этого аппаратная часть процессора должна быть построена так, чтобы один из операндов, например первый, и результат операции размещались в одном и том же фиксированном регистре. Выделенный для этой цели внутренний регистр процессора получил название *аккумулятор*. Адрес же другого операнда указывается в команде.

5. *Безадресная структура*

Использование безадресных команд возможно только при естественном порядке выборки команд и подразумеваемых адресах обоих операндов и результата операции, например при работе со стековой памятью. В этом случае один операнд находится в вершине стека, а второй операнд – в аккумуляторе. Туда же помещается и результат.

Обычно в ЭВМ используется несколько форматов команд разной длины. Кроме того, трехадресные команды в современных универсальных ЭВМ практически не используются, так как являются слишком громоздкими (хотя и самыми удобными с точки зрения программиста). Обычно используются одно- и двухадресные команды и их модификации. Следует

отметить, что трехадресные команды используются при работе с памятями небольшого объема. Обычно это массивы внутренней памяти процессора. При выполнении операций типа "регистр-регистр" и достаточно больших объемах регистровой памяти (десятки регистров), как, например, в процессорах классической RISC-архитектуры, подобные команды очень эффективны.

Рассмотренные структуры команд достаточно схематичны. В реальных ЭВМ адресные поля команд большей частью содержат не сами адреса, а только информацию, позволяющую определить действительные (*исполнительные*) адреса операндов в соответствии с используемыми в командах способами адресации.

2.10 Способы адресации

Определимся с терминами, которые будут использоваться далее.

Адресный код (АК) – это информация об адресе операнда, содержащаяся в команде.

Исполнительный адрес (АИ) – это номер ячейки ОП, к которой производится фактическое обращение.

В большинстве команд современных ЭВМ адресный код не совпадает с исполнительным адресом. Проблема выбора способов адресации и формирования исполнительного адреса, как и системы команд, относится также к важнейшим при разработке новых процессоров и ЭВМ.

Как уже отмечалось, процесс эволюции ЭВМ привел к появлению команд очень сложной структуры. Это в полной мере относится и к системам адресации, используемым в современных универсальных ЭВМ, поддерживающих многозадачные режимы и виртуальную память. Между тем эти системы адресации являются сложными комбинациями относительно небольшого количества способов адресации, разработанных еще для ЭВМ первых поколений. Ниже рассмотрены основные из этих способов адресации.

Подразумеваемый операнд

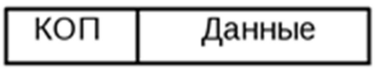
В команде не содержится явных указаний о самом операнде или его адресе. Операнд подразумевается и фактически задается кодом операции команды. Этот метод используется редко, но иногда бывает очень удобен, например, в командах подсчета, когда к содержимому счетчика необходимо добавить фиксированное приращение. Это касается и других операций, связанных с добавлением (вычитанием) константы.

Подразумеваемый адрес

В команде не содержится явных указаний об адресе участвующего в операции операнда или адресе результата операции. Так, например, при выполнении данной операции результат всегда засылается по адресу второго операнда или в другой, заранее определенный, регистр. В простейших микропроцессорах результат всегда помещается в аккумулятор.

Непосредственная адресация

В команде содержится не адрес операнда, а непосредственно сам операнд (рис. 2.10.1). Это способ уменьшения объема программы и занимаемой памяти, так как не требует операций обращения к памяти и самой ячейки памяти.

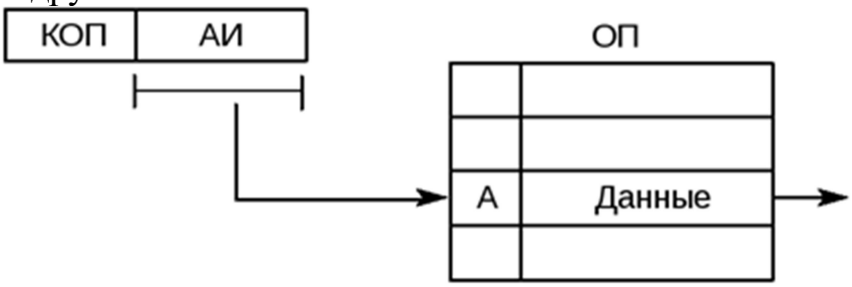


2.10.1 Структура команды при непосредственной адресации

Применение данного способа ускоряет вычисления. Обычно он используется для хранения различных констант.

Прямая адресация

Исполнительный адрес совпадает с адресной частью команды, т.е. адресный код совпадает с исполнительным адресом (рис. 2.10.2). Этот способ был основным в первых ЭВМ. В настоящее время используется в комбинации с другими способами.



КОП – код операции;
АИ – исполнительный адрес;
ОП – оперативная память

2.10.2 Прямая адресация

Достоинства: быстрота исполнения и простота реализации. Недостатком является длинная адресная часть команды.

Относительная адресация, или базирование

Исполнительный адрес (АИ) определяется как сумма адресного кода команды (АК) и некоторого числа АБ, называемого *базовым адресом*. АИ = АБ + АК (рис. 2.10.3).

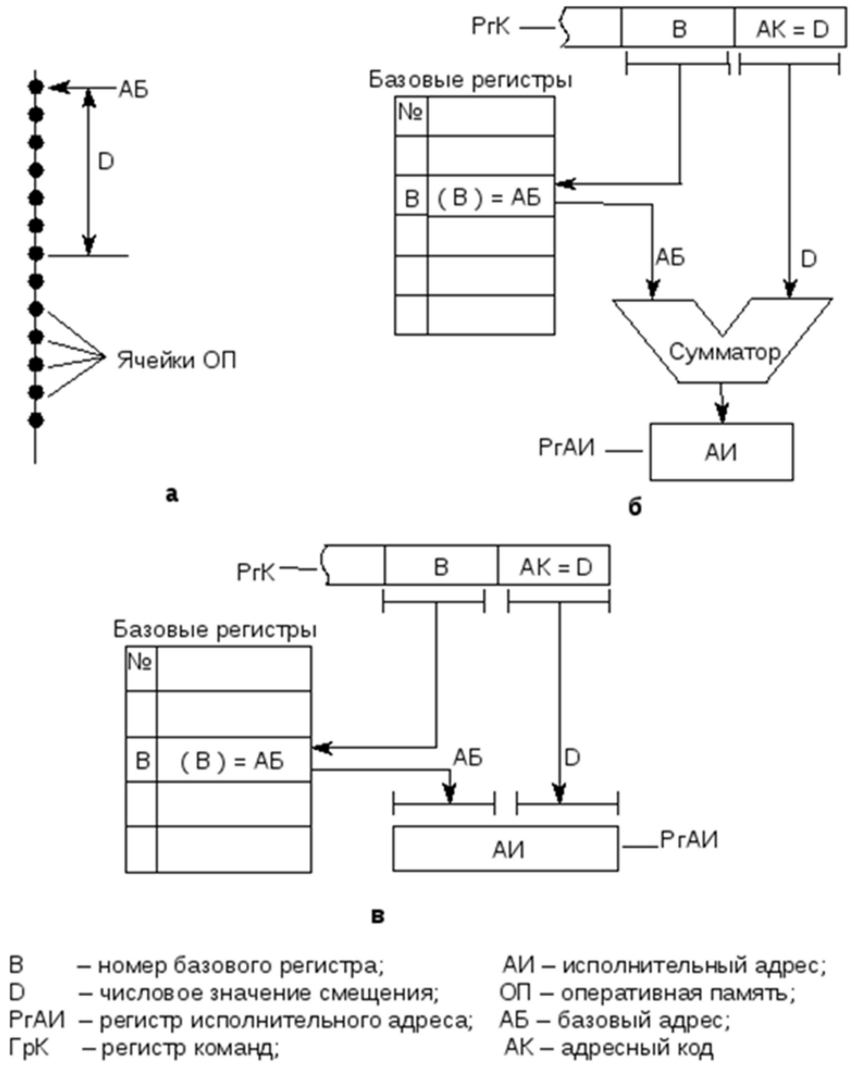


рис. 2.10.3 Базирование (относительная адресация): а – образование адреса элемента одномерного массива; б – формирование исполнительного адреса суммированием; в – формирование исполнительного адреса совмещением

Для хранения АБ используются *базовые регистры*. Это или специальные внутренние регистры процессора и сверхоперативной памяти, или специально выделенные ячейки ОП с короткими начальными адресами. В команде выделяется поле "B" для указания номера базового регистра. Число разрядов в базовом адресе АБ выбирается таким, чтобы можно было адресовать любую ячейку ОП. Адресный код АК самой команды имеет мало разрядов и используется для представления лишь сравнительно короткого "смещения" (D). Это смещение определяет положение операнда относительно начала массива, задаваемого базовым адресом АБ (рис. 2.10.3, а).

Возможны два варианта формирования АИ при базировании.

- *Метод суммирования*(рис. 2.10.3, б).

Этот метод прост и позволяет задавать в качестве АБ любой адрес ОП. Недостатком является то, что на операцию суммирования уходит время.

- *Метод совмещения*(рис.2.10.3, в).

В этом случае АБ содержит старшие, а адресный код АК младшие разряды исполнительного адреса АИ, которые в регистре адреса ОП объединяются (операция *конкатенации*). При таком методе формирования АИ базовый адрес АБ может задавать не любую ячейку, а только ту, адрес которой содержит нули в младших разрядах, соответствующих смещению. Формирование адреса осуществляется быстрее, так как не требуется операции суммирования.

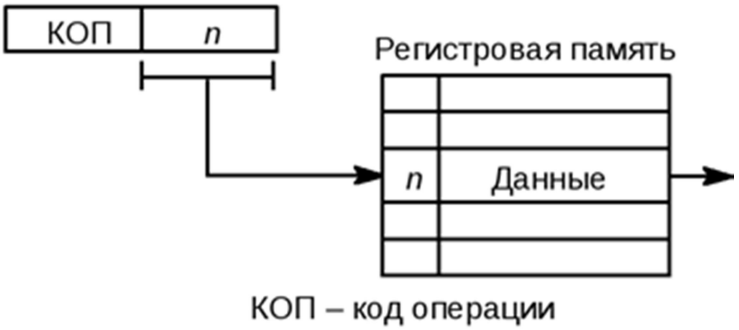
Относительная адресация обеспечивает возможность передвижения программ в памяти без изменений внутри самой программы за счет изменения базовых адресов (так называемая *перемещаемость* программ), что является основой для построения механизма виртуальной памяти. Кроме того, она облегчает компоновку программ, части которых написаны разными программистами.

Регистровая адресация

Это частный случай так называемой *укороченной* адресации, суть которой сводится к тому, что используется только небольшая группа фиксированных ячеек памяти с начальными (короткими) адресами (0000001, 0000010, 0000011 и т.д.). Такая адресация используется только совместно с другими типами адресации.

При использовании укороченной адресации длина команды существенно сокращается, так как используются только младшие разряды адресов.

В случае ***регистровой адресации*** (рис. 2.10.4) в качестве фиксированных ячеек с короткими адресами используются регистры внутренней памяти процессора, которых обычно немного. Поэтому разрядность АК также невелика.



2.10.4 Регистровая адресация

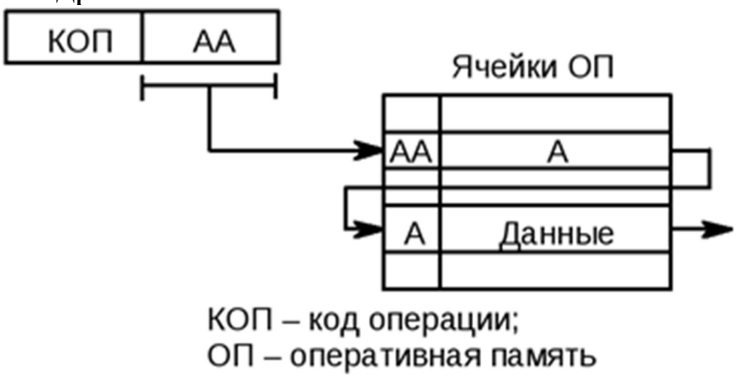
Это, фактически, прямая адресация к сверхбыстрой памяти процессора. Достоинства данного способа адресации – укорочение команд, увеличение скорости выполнения операций. Недостаток – малое число адресов.

Косвенная адресация

Адресный код (АК) команды указывает адрес ячейки ОП, в которой находится исполнительный адрес (АИ) операнда или команды, т.е. это адрес адреса – АА. Схема косвенной адресации представлена на рис. 2.10.5.

На косвенную адресацию указывает код операции (КОП) команды. В некоторых ЭВМ в команде отводится специальный разряд (*указатель*

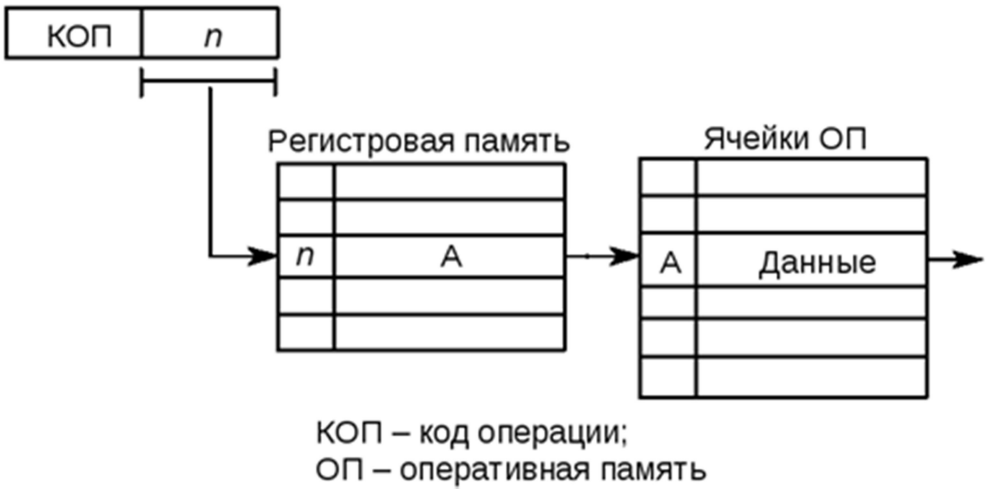
адресации– УА), и цифра 0 или 1 в нем указывает, является адресная часть команды прямым адресом или косвенным.



2.10.5 Косвенная адресация

В ряде случаев используется многоступенчатая косвенная адресация. В этом случае ячейки ОП также содержат разряд УА. Перебор ячеек ОП происходит до тех пор, пока не будет найдена ячейка, в которой УА определит прямую адресацию.

Косвенная адресация широко используется в микропроцессорах и малых ЭВМ, имеющих короткое машинное слово, для преодоления ограничений короткого формата команды. Такой способ адресации удобен также, когда требуется модификация исполнительных адресов, например, при циклической обработке элементов массива. Для этого в микропроцессорах и малых ЭВМ совместно используют *регистровую* и *косвенную* адресации (рис. 2.10.6).



2.10.6 Регистровая и косвенная адресация

Такая адресация позволяет микропроцессору адресоваться к ОП достаточно большого объема при небольшой длине адресного поля команды (адресного поля достаточно только для указания номеров нескольких внутренних регистров микропроцессора). Естественно, что перед выполнением этой команды в соответствующий внутренний регистр микропроцессора должен быть загружен полный адрес интересующей ячейки ОП.

Автоинкрементная и автодекрементная адресации

Выше отмечалось, что при косвенной регистровой адресации в соответствующий регистр перед использованием необходимо загрузить

исполняемый адрес, по которому произойдет обращение к ячейке ОП. На это уходит время. Однако при обработке массива данных этих потерь можно избежать, добавив механизм автоматического приращения или уменьшения содержимого регистра при каждом обращении к нему. В этом случае загрузка начального адреса массива происходит один раз, а затем при каждом обращении формируется адрес следующего элемента массива.

Автоинкрементная адресация– сначала (при каждом обращении) содержимое регистра используется как адрес операнда, а затем получает приращение, равное числу байт в элементе массива, т.е. формируется адрес следующего элемента.

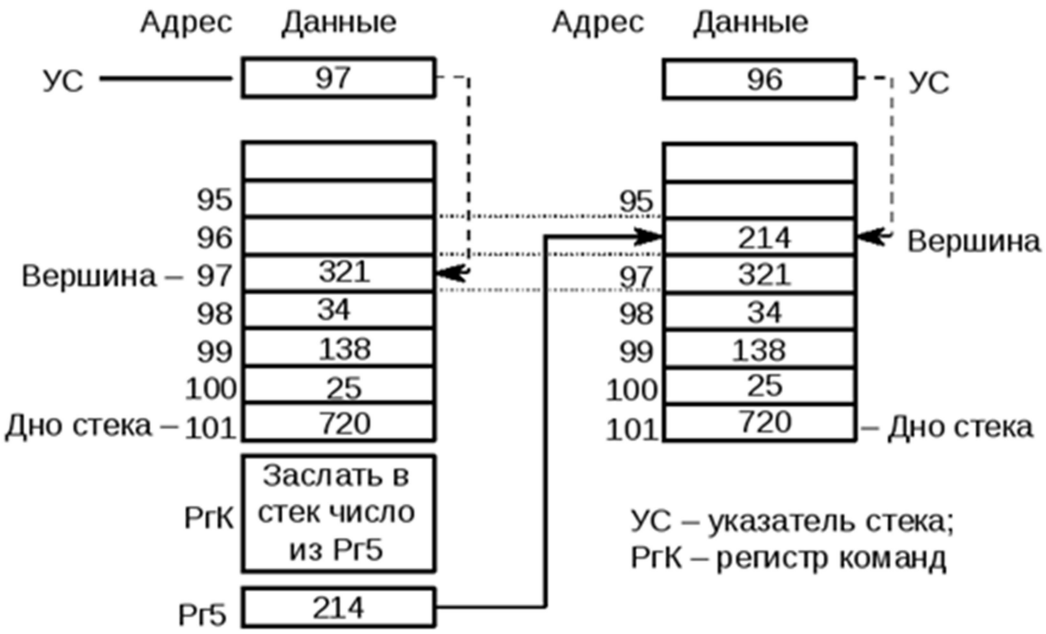
Автодекрементная адресация – сначала содержимое соответствующего регистра уменьшается на число, равное числу байт в элементе массива, а затем используется как адрес операнда.

Следует иметь в виду, что автоинкрементная и автодекрементная адресации являются упрощенным вариантом *индексации* – весьма важного механизма преобразования исполнительных адресов команд и организации вычислительных циклов. Эти типы адресации называют также *автоиндексацией*.

Стековая адресация

При рассмотрении устройств памяти отмечалось, что основной принцип работы стекового ЗУ соответствует правилу: "последний пришел – первый ушел" (имеется в виду стек LIFO). Это правило реализуется автоматически. Поэтому при операциях со стеком возможно безадресное задание операнда – команда не содержит адреса ячейки стека, а содержит только адрес (или он подразумевается) регистра или ячейки ОП, откуда слово загружается в стек или куда выгружается из стека. Стек может быть реализован как аппаратным путем, так и программно. В первом случае стек представляет собой одномерный массив регистров, связанных между собой разрядными цепями передачи данных. Обычно он снабжен счетчиком стека, по содержимому которого можно контролировать переполнение стека. Во втором случае стек организуется на последовательно расположенных ячейках ОП. Для его реализации требуется еще один регистр – указатель стека, в котором хранится адрес вершины, т.е. последней занятой ячейки ОП из массива ячеек, отведенных под стек.

Стек является эффективным элементом архитектуры современных ЭВМ, позволяющим во многих случаях существенно повысить скорость обработки информации. В универсальных ЭВМ общего применения (таких как персональный компьютер) программисту в большинстве случаев доступен только программный стек. На рис. 2.10.7 приведена схема записи числа в "перевернутый" программный стек, который используется наиболее широко.



2.10.7 Операция записи числа в «перевернутый» программный стек

При выполнении команды загрузки слова в стек (содержимое РгК) из регистра (в данном случае из Рг5) или ячейки ОП сначала содержимое указателя стека (УС) уменьшается на 1 (стек "перевернутый"), а затем слово помещается в ячейку стека, указываемую УС. При выгрузке слова из стека в регистр или ОП слово сначала извлекается из вершины стека, а затем УС увеличивается на 1 (на рисунке не показано).

При надлежащем расположении операндов в стеке можно произвести вычисления полностью безадресными командами. Команды этого типа обычно инициируют извлечение из стека одного или двух операндов (слов), выполнение над ними указанных в коде команды операций и запись результата в какой-либо фиксированный регистр, например, аккумулятор. Вычисления с использованием стековой памяти удобно описывать и программировать с помощью инверсной (бесскобочной) записи арифметических выражений.

Безадресные команды на основе стековой адресации предельно сокращают формат команд, экономят память и повышают производительность ЭВМ.

В современных ЭВМ (микропроцессорах) стек и стековая адресация широко используется для:

- сохранения содержимого регистров при переходе к подпрограмме и выходе из нее;
- сохранения информации, содержащейся во внутренних регистрах процессора при прерываниях программы;
- организации хранения элементов массивов при их циклической обработке.

Как уже отмечалось, современные ЭВМ во многих случаях используют сложные, комбинированные системы адресации, которые не могут быть в чистом виде отнесены к какому-либо одному из рассмотренных выше способов адресации. Это позволяет программисту более гибко и эффективно

использовать все ресурсы ЭВМ. Однако разнообразие форматов команд и их длины ведет к усложнению УУ процессора и замедлению процесса выполнения команды.