Лекция7. Система команд и способы адресации операндов. Конвейерный принцип выполнения команд

Система команд и способы адресации операндов

Система команд (instruction set) — это совокупность команд, или инструкций, выполнение которых на аппаратном уровне поддерживает процессор. Понятие «система команд» не следует путать с более широким понятием «архитектура системы команд». *Архитектура системы команд* — это не только система команд, но и средства для их выполнения, такие как форматы данных, системы регистров, способы адресации, модели памяти.

Отметим, что различные процессоры могут использовать одну систему команд, но при этом иметь разное внутреннее устройство. В качестве примера можно привести МП общего назначения фирм Intel и AMD. Программы, разработанные для одного из них, могут быть выполнены и на другом, несмотря на то, что оба процессора созданы независимыми группами людей, имеют разные архитектуры и наборы технических решений.

В настоящее время существует множество систем команд, отличающихся по сложности и функциональным возможностям. Некоторые системы команд известны уже не один десяток лет и постоянно расширяются. Проследим историю наиболее популярной из них — системы команд x86 на примере процессоров фирмы Intel:

- 8086 — 16-разрядные регистры и подсистема памяти, 80 команд (1978 г.);
- 80186 — добавлены 7 команд (1982 г.);
- 80286 — добавлены 17 команд (1982 г.);
- 80386 — регистры и подсистема памяти расширены до 32 разрядов, введены 33 новые команды (1985 г.);
- 80486 — добавлены 6 команд, а также команды для работы с блоком вычислений с плавающей точкой (1989 г.);
- Pentium — добавлены 6 команд, всего их более 150 (1993 г.);

• x86-64 — регистры и подсистема памяти расширены до 64 разрядов, введены 14 новых команд (2003 г.).

Команды, входящие в систему команд, можно классифицировать по нескольким признакам (рис. 2.2.1): по функциональному назначению; числу адресов, к которым обращается команда; способу адресации.

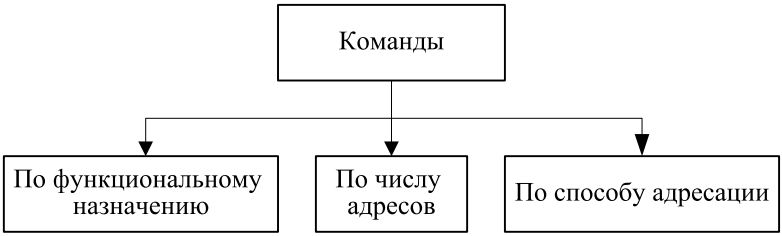

Рис. 2.2.1. Классификация команд процессора

По *функциональному назначению* команды разделяются на следующие основные группы операций:

- пересылки;
- арифметические;
- логические;
- сдвига;
- сравнения;
- управления программой;
- управления процессором.

Обычно вместо машинных кодов используются близкие к ним машинно-ориентированные языки — ассемблеры, которые позволяют применять мнемоническое представление команд. При дальнейшем рассмотрении будем использовать мнемонические ассемблерные команды условного процессора, не привязываясь к конкретной системе команд. Тем не менее, нижеприведенные обозначения широко распространены и встречаются в составе ассемблеров многих современных процессоров.

Операции пересылки обеспечивают передачу данных между двумя регистрами или регистром и ячейкой памяти. Некоторые процессоры предполагают использование отдельных команд для общения

с портами ввода/вывода, пересылки между ячейками памяти и групповой пересылки, когда содержимое нескольких регистров пересылается в память или загружается из нее. Формат команд этой группы: MOV <приемник>, <источник> — пересылка между регистрами; LD — загрузка данных из памяти в регистр; ST — сохранение данных из регистра в память.

Арифметические операции включают такие команды, как сложение, вычитание, умножение и деление. Различают команды, которые выполняют операции непосредственно с операндами, и команды, которые учитывают сформированный предыдущими командами бит переноса. Команды, учитывающие бит переноса, используются при обработке слов с разрядностью, превышающей разрядность процессора. Так, в простейших системах операции умножения и деления не входят в систему команд процессора и реализуются программно. При умножении разрядность произведения в два раза больше разрядности операндов, поэтому для хранения результата используются два регистра (младшая и старшая части произведения). При делении делимое тоже имеет удвоенную разрядность и размещается в двух регистрах. В этом случае в качестве результата в двух регистрах сохраняются частное и остаток. Формат арифметических команд: ADD <операнд 1/приемник результата>, <операнд 2>, SUB, MUL, DIV — сложение, вычитание, умножение и деление соответственно.

Логические операции включают операции И (AND), ИЛИ (OR), НЕ (NOT), Исключающее ИЛИ (XOR). Логические операции выполняются поразрядно над содержимым двух регистров, регистра и ячейки памяти или с использованием операнда, который размещен в самой команде. Для этого применяют двухадресные команды AND (OR, XOR) <операнд 1/приемник результата>, <операнд 2>. Одноадресная команда NOT инвертирует каждый разряд операнда.

Микропроцессоры могут включать в свою систему команд логические, арифметические и циклические сдвиги адресуемых операндов на один или несколько разрядов (рис. 2.2.2). Сдвигаемый операнд может находиться в регистре или ячейке памяти, а число разрядов, на которое сдвигается операнд может либо задаваться с помощью операнда, содержащегося в команде при непосредственной адресации, либо определяться содержимым заданного регистра C. Мнемоника таких команд выглядит следующим образом:

• SHL (SHR) <операнд/приемник результата> — логический сдвиг влево (вправо);

• SAL (SAR) — арифметический сдвиг влево (вправо);

• ROL (ROR) — циклический сдвиг влево (вправо);

• RCL (RCR) — сдвиг влево (вправо) на заданное число разрядов.

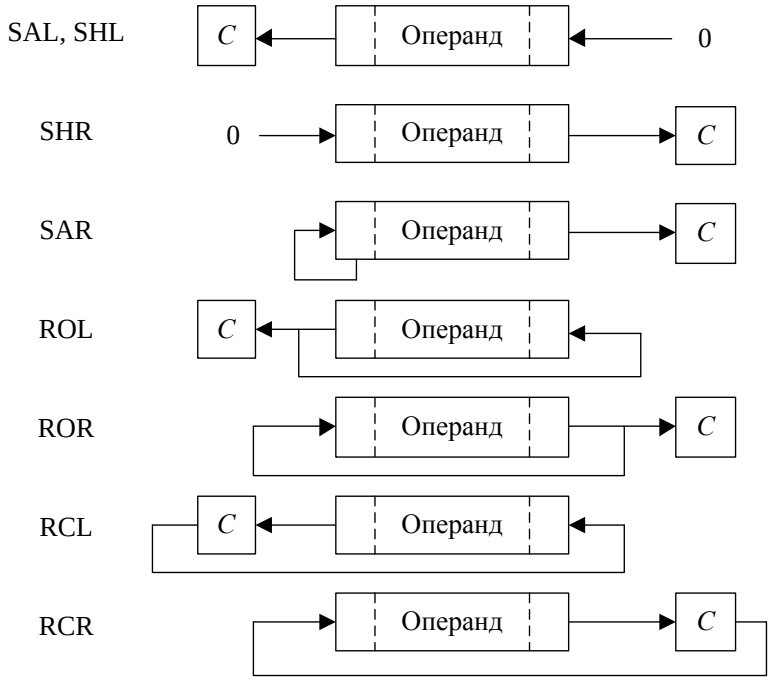

Рис. 2.2.2. Реализация команд сдвига

Операции сравнения в общем виде можно представить следующим образом: CMP <операнд 1>, <операнд 2>. Для выполнения данной команды производится вычитание двух указанных операндов, при этом результат вычитания не сохраняется, операнды не изменяются, а результатом операции является изменение содержимого регистра состояний процессора.

Операции управления программой делятся на команды безусловной передачи управления (CALL — вызов подпрограммы, JMP — переход по указанному адресу), команды возврата (RET — возврат из подпрограммы, IRET — возврат из прерывания) и команды условного перехода (JCC), которые выполняются в зависимости от состояния флагов (нулевого результата, переноса и других) процессора.

Операции управления процессором включают команды:

- HALT — остановка выполнения текущей программы;
- RST (reset) — сброс всех регистров, в том числе регистров состояния процессора;
- NOP (no operation) — нет операции, цикл ожидания.

Отметим, что при проектировании процессора и выборе набора команд для него чаще всего применяют проблемно-ориентированный подход. Для универсальных процессоров используют большой набор команд. В набор команд специализированных процессоров, например обработки сигналов, включают команду умножения с накоплением, которая выполняется за один такт и позволяет повысить производительность процессора в задачах фильтрации и обработки сигналов. Недорогие микроконтроллеры содержат минимальный набор необходимых инструкций (что упрощает их разработку), дающий возможность, однако, реализовать сложные функции через последовательность простых. Конечно, при этом приходится жертвовать быстродействием, но для ряда задач важно минимальное энергопотребление в ущерб производительности.

В соответствии с концепцией проблемно-ориентированного подхода широкое распространение сейчас получили «проблемно-ориентированные процессоры» — компоненты систем на кристалле, спроектированные для выполнения специфических операций (например, векторные сопроцессоры, процессорные ядра цифровой обработки сигналов), позволяющие достичь компромисса между универсальностью МП общего назначения и производительностью специализированной заказной микросхемы.

Команда микропроцессора состоит из двух полей (рис. 2.2.3): поля кода операции (КОП) и поля адреса, которое может включать несколько кодов адреса (КАД).

КОП определяет действие, выполняемое процессором, а КАД — операнды. КАД может отсутствовать, если команда является безадресной, например, команды проверки флагов процессора или управ-

Рис. 2.2.3. Формат команды микропроцессора

ляющие его работой. Команды, работающие с данными (пересылки, арифметические, логические и сдвига), могут иметь один и более операндов и, соответственно, полей КАД (см. рис. 2.2.3). Так, операция побитовой инверсии является одноадресной, команды сложения и вычитания — двухадресными. В более сложных системах команд встречаются трехадресные инструкции, например, чтение из регистров двух операндов, их сложение и размещение результата в третьем ($C = A + B$) или инструкция умножения с накоплением ($C = A \cdot B + C$).

Различают шесть основных способов адресации операндов:

- непосредственная;
- прямая;
- регистровая;
- косвенно-регистровая;
- косвенно-регистровая со смещением;
- относительная.

При *непосредственной* адресации операнд содержится в поступившей команде и используется для введения различных констант. Такой способ вызывает увеличение размера команды на число байтов заданного операнда.

Разновидностью такой адресации является адресация, при которой операнд подразумевается, а не указывается явным образом. Примером может служить операция сравнения с нулем, в которой нуль не задается явным образом, но подразумевается.

При *прямой* адресации операнд выбирается из ячейки памяти, адрес которой указан в команде (рис. 2.2.4). Достоинство данного способа — простота, недостаток — большая разрядность адресного поля, зависящая от размера адресного пространства.

При *регистровой* адресации операнд выбирают из ячейки регистрового запоминающего устройства (РЗУ), номер (или имя) которой указан в команде (рис. 2.2.5). Регистровая адресация отличается

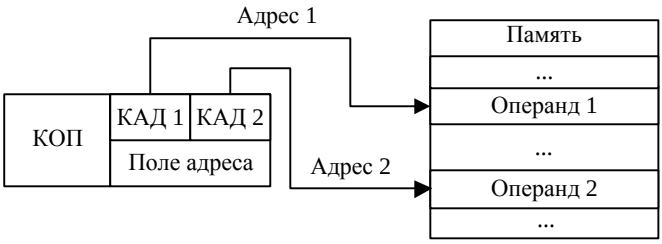

Рис. 2.2.4. Прямая адресация операндов

простотой и быстродействием и весьма распространена. Такой способ адресации не требует обращения к внешней памяти из-за того, что РЗУ находится на кристалле процессора. Как следствие, выполнение операций при регистровой адресации занимает минимальное время, однако необходимо периодическое обращение к памяти для сохранения результатов и записи операндов в РЗУ, что несколько уменьшает выгоду от использования такого способа адресации. Дешифрация адреса также относительно проста: объем РЗУ ограничен, а для задания номера регистра требуется всего несколько битов (3 – 8).

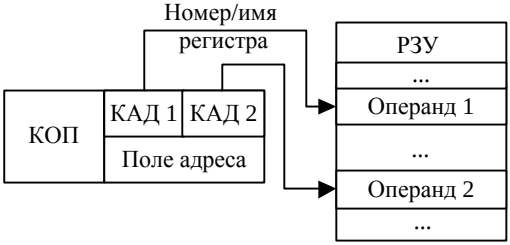

Рис. 2.2.5. Регистровая адресация операндов

При *косвенно-регистровой* адресации операнд выбирается из ячейки памяти, адрес которой содержится в регистре, указанном в команде (рис. 2.2.6).

При *косвенно-регистровой адресации со смещением* операнд выбирают из ячейки памяти, адрес которой является суммой содержимого регистра, адрес которого указан в команде, и операнда смещения (смещение может быть положительным или отрицательным).

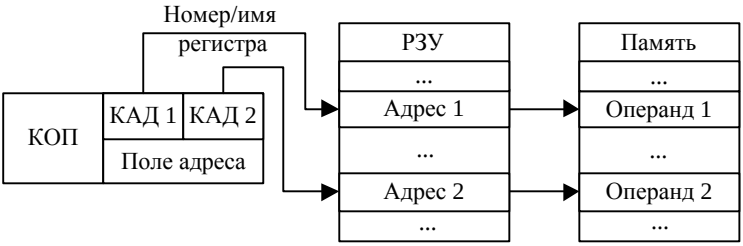

Рис. 2.2.6. Косвенно-регистровая адресация операндов

Разновидностями косвенно-регистровой адресации со смещением являются автоинкрементная и автодекрементная адресации. Как и при косвенно-регистровой адресации, адрес ячейки операнда хранится в регистре, но при каждом обращении к регистру его значение увеличивается или уменьшается на единицу. Эти способы адресации удобны при обработке массивов данных в цикле, поскольку нет необходимости каждый раз загружать адрес обрабатываемого значения массива в регистр.

При *относительной* адресации операнд выбирается из ячейки памяти, адрес которой является суммой текущего содержимого программного счетчика и заданного в команде смещения — базового адреса (рис. 2.2.7). В большинстве случаев такой способ адресации используется не для адресации операнда, а для формирования адреса команды, к которой переходит программа при выполнении команд ветвления. Сформированный адрес загружается в счетчик программ, обеспечивая выборку требуемой команды. Достоинством данного способа является то, что адресное поле команды может быть меньше шины адреса процессора.

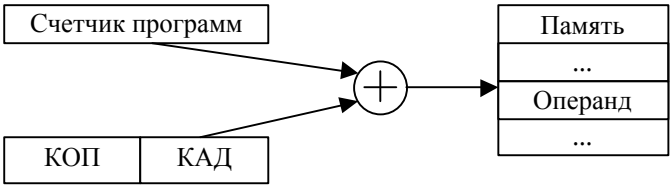

Рис. 2.2.7. Относительная адресация операндов

Отметим, что наиболее распространенными являются различные варианты косвенно-регистровой адресации. При этом в команде указывается только номер регистра, используемого в качестве адресного, следовательно, размер команды оказывается небольшим (как при регистровой адресации). Однако выборка операнда из ОЗУ требует выполнения циклов передачи данных по системной шине, что снижает производительность МП.

Принцип совмещения выполнения операций во времени известен давно и широко применяется на практике. Этот общий принцип включает два понятия: *параллелизм* и *конвейеризация*. Хотя у них много общего и их зачастую трудно различить, эти термины отражают два различных подхода: при параллелизме совмещение операций достигается воспроизведением их в нескольких копиях аппаратной структуры, этот подход рассмотрен в разделе 5, далее подробнее рассмотрим конвейеризацию.

Конвейерный принцип выполнения команд

Конвейеризация (instruction pipelining) в общем случае основана на разделении выполняемой операции на части, называемые *ступенями*, и выделении для каждой из них отдельного блока аппаратуры. Таким образом, обработку любой машинной команды можно разделить на несколько этапов, организовав передачу данных от одного этапа к следующему. При этом конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность процессора в данном случае возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Конвейерная обработка широко применяется во всех современных быстродействующих процессорах.

Конвейеризация увеличивает количество команд, завершающихся в единицу времени, следовательно, программа будет выполняться быстрее. Отметим, что при этом конвейеризация не сокращает время выполнения отдельной команды. Производительность процессора прямо пропорциональна его тактовой частоте и обратно пропорциональна длительности самой медленной ступени конвейера.

Если в процессоре без применения конвейера одна команда выполняется за N тактов, то говорят о производительности $1/N$ команд за такт. Если в этот процессор ввести конвейер длиной N ступеней, то

при полной загрузке конвейера производительность процессора увеличится в N раз. Таким образом, эффективнее разбивать команды на большее количество ступеней и делать конвейер более длинным, повышая общую производительность процессора.

Конвейер позволяет увеличить производительность процессора в N раз только тогда, когда его загрузка близка к полной, а скорость подачи новых команд и операндов соответствует максимальной производительности конвейера. Если произойдет задержка, то параллельно будет выполняться меньшее количество команд, и суммарная производительность конвейера, а следовательно, и всего МП снизится.

Кроме того, введение конвейерной обработки возможно для больших комбинационных схем. В качестве примера можно привести умножитель 32×32 бита, в котором самый старший бит результата является функцией от всех 64 битов операндов, поэтому расчет произведения занимает значительное время. Такой умножитель можно реализовать, объединив четыре умножителя 16×16 битов, работающих последовательно, что позволяет увеличить частоту работы схемы.

Однако в процессе конвейерной обработки иногда возникают ситуации, которые препятствуют выполнению очередной команды из потока команд в предназначенном для нее такте. Такие ситуации называются *конфликтами*. Конфликты приводят к необходимости приостановки выполнения команд и снижают производительность конвейера. В самом простом случае, если приостанавливается какая-либо команда, то все следующие за ней команды также приостанавливаются. Команды, предшествующие приостановленной, могут продолжать выполняться, но во время приостановки не выбирается ни одна новая команда.

Состояние ожидания (ОЖ) — состояние, при котором конвейер пропускает один или несколько тактов из-за того, что не готовы операнды.

Состояние простоя (ПР) — состояние, при котором конвейер пропускает один или несколько тактов потому, что данный этап конвейера не используется в данной команде.

Рассмотрим пример разбиения команды на ступени:

1) ВК — выборка команды;
2) ДК — дешифрация команды;
3) ФА — формирование адреса;

4) ПО — получение операндов;
5) ВО — выполнение операции;
6) РР — размещение результата в РЗУ или памяти.

На рис. 2.2.8, *а* представлены временные диаграммы функционирования конвейера при идеальной загрузке на примере выполнения трех команд: инструкция K_1 находится на стадии РР, в то время как последняя команда K_3 находится на стадии получения операндов.

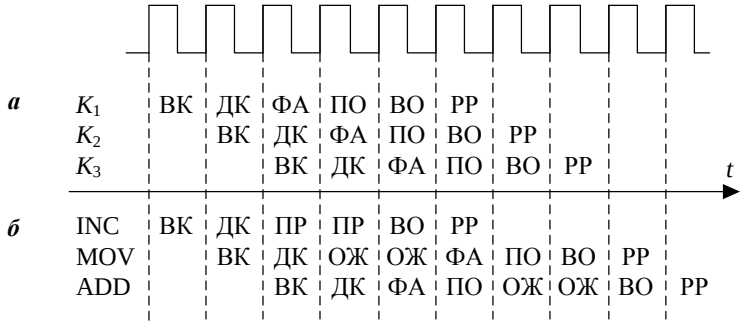

Рис. 2.2.8. Временные диаграммы функционирования 6-ступенчатого конвейера команд: *а* — при идеальной загрузке; *б* — при наличии зависимостей по данным

На рис. 2.2.8, *б* показано, как снижается производительность конвейерной обработки команд при выполнении трех команд, имеющих зависимости по данным:

1) INC R2 — увеличение содержимого регистра R2 на единицу;
2) MOV R3, (R2) — пересылка содержимого ячейки памяти, адрес которой находится в R2, в регистр R3;
3) ADD R3, (R4) — сложение содержимого регистра R3 с содержимым ячейки памяти, адрес которой находится в R4, сохранение результата в R3.

На рис. 2.2.8, *б* показано также, что при выполнении реальной программы возникают состояния ожидания и простоя конвейера. Первая команда вызывает состояние простоя конвейера, потому что команда инкремента не проходит стадии ПО. Вторая и третья команды вызывают состояние ожидания конвейера из-за того, что неизвестен результат выполнения предыдущей операции.

По причинам возникновения различают следующие типы конфликтов:

1) структурные — когда аппаратные средства не могут поддерживать все возможные комбинации команд в режиме одновременного выполнения с совмещением;

2) конфликты по данным — когда выполнение текущей команды зависит от результата выполнения предыдущей команды;

3) конфликты по управлению — когда команда переходов либо какая-то другая изменяет значение программного счетчика.

Контрольные вопросы

1. Дайте определение системы команд процессора.
2. По каким признакам можно классифицировать команды микропроцессора?
3. Какие способы адресации вы знаете?
4. В чем состоит конвейерный принцип выполнения команд?
5. Какие типы конфликтов выделяют в конвейере?

Литература

1. Микропроцессоры. В 3-х кн. Кн. 1. Архитектура и проектирование микроЭВМ: учебник для втузов / ***Под ред. Л.Н. Преснухина***. — М.: Высшая школа, 1986. — 495 с.

2. Микропроцессорные системы: учеб. пособие для вузов / ***Е.К. Александров, Р.И. Грушвицкий, М.С. Куприянов и др.***; ***под ред. Д.В. Пузанкова***. — СПб.: Политехника, 2002. — 935 с.