Глава 2
Организация компьютерных систем

Цифровой компьютер состоит из связанных между собой процессоров, модулей памяти и устройств ввода-вывода. Глава 2 призвана познакомить читателя с этими компонентами и с тем, как они взаимосвязаны. Данная информация послужит основой для подробного рассмотрения каждого уровня в последующих пяти главах. Процессоры, память и устройства ввода-вывода — ключевые понятия, они будут упоминаться при обсуждении каждого уровня, поэтому изучение компьютерной архитектуры мы начнем с них.

Процессоры

На рис. 2.1 показана структура обычного компьютера с шинной организацией. **Центральный процессор** — это мозг компьютера. Его задача — выполнять программы, находящиеся в основной памяти. Для этого он вызывает команды из памяти, определяет их тип, а затем выполняет одну за другой. Компоненты соединены **шиной**, представляющей собой набор параллельно связанных проводов для передачи адресов, данных и управляющих сигналов. Шины могут быть внешними (связывающими процессор с памятью и устройствами ввода-вывода) и внутренними. Современный компьютер использует несколько шин.

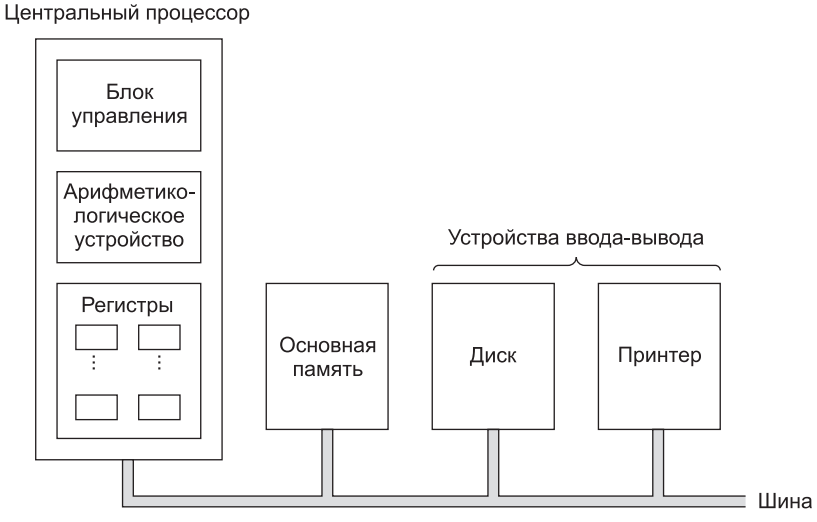

Рис. 2.1. Схема компьютера с одним центральным процессором и двумя устройствами ввода-вывода

Процессор состоит из нескольких частей. Блок управления отвечает за вызов команд из памяти и определение их типа. Арифметико-логическое устройство выполняет арифметические операции (например, сложение) и логические операции (например, логическое И).

Внутри центрального процессора находится быстрая память небольшого объема для хранения промежуточных результатов и некоторых команд управления. Эта память состоит из нескольких регистров, каждый из которых выполняет определенную функцию. Обычно размер всех регистров одинаков. Каждый регистр содержит одно число в диапазоне, верхняя граница которого зависит от размера регистра. Операции чтения и записи с регистрами выполняются очень быстро, поскольку они находятся внутри центрального процессора.

Самый важный регистр — **счетчик команд**, который указывает, какую команду нужно выполнять следующей. Название «счетчик команд» выбрано неудачно, поскольку он ничего не *считает*, но этот термин употребляется повсеместно[1]. Еще есть **регистр команд**, в котором находится выполняемая в данный момент команда. У большинства компьютеров имеются и другие регистры, одни из них многофункциональны, другие служат лишь какие-либо конкретным целям. Третьи регистры используются операционной системой для управления компьютером.

Устройство центрального процессора

Внутреннее устройство тракта данных типичного фон-неймановского процессора иллюстрирует рис. 2.2. **Тракт данных** состоит из регистров (обычно от 1 до 32), **арифметико-логического устройства (АЛУ)** и нескольких соединительных шин. Содержимое регистров поступает во входные регистры АЛУ, которые на рис. 2.2 обозначены буквами A и B. В них находятся входные данные АЛУ, пока АЛУ производит вычисления. Тракт данных — важная составная часть всех компьютеров, и мы обсудим его очень подробно.

АЛУ выполняет сложение, вычитание и другие простые операции над входными данными и помещает результат в выходной регистр. Содержимое этого выходного регистра может записываться обратно в один из регистров или сохраняться в памяти, если это необходимо. Не во всех архитектурах есть регистры A, B и выходные регистры. На рис. 2.2 представлена операция сложения, но АЛУ может выполнять и другие операции.

Большинство команд можно разделить на две группы: команды типа регистр-память и типа регистр-регистр. Команды первого типа вызывают слова из памяти, помещают их в регистры, где они используются в качестве входных данных АЛУ (слова — это такие элементы данных, которые перемещаются между памятью и регистрами[2]). Словом может быть целое число. Организацию памяти мы обсудим далее в этой главе. Другие команды этого типа помещают регистры обратно в память.

Команды второго типа вызывают два операнда из регистров, помещают их во входные регистры АЛУ, выполняют над ними какую-нибудь арифметическую или логическую операцию и переносят результат обратно в один из реги-

[1] Используется также термин «указатель команд». — *Примеч. науч. ред.*

[2] На самом деле размер слова обычно соответствует разрядности регистра данных. Так, у 16-разрядных микропроцессоров 8086 и 8088 слово имеет длину 16 бит, а у 32-разрядных микропроцессоров — 32 бита. — *Примеч. науч. ред.*

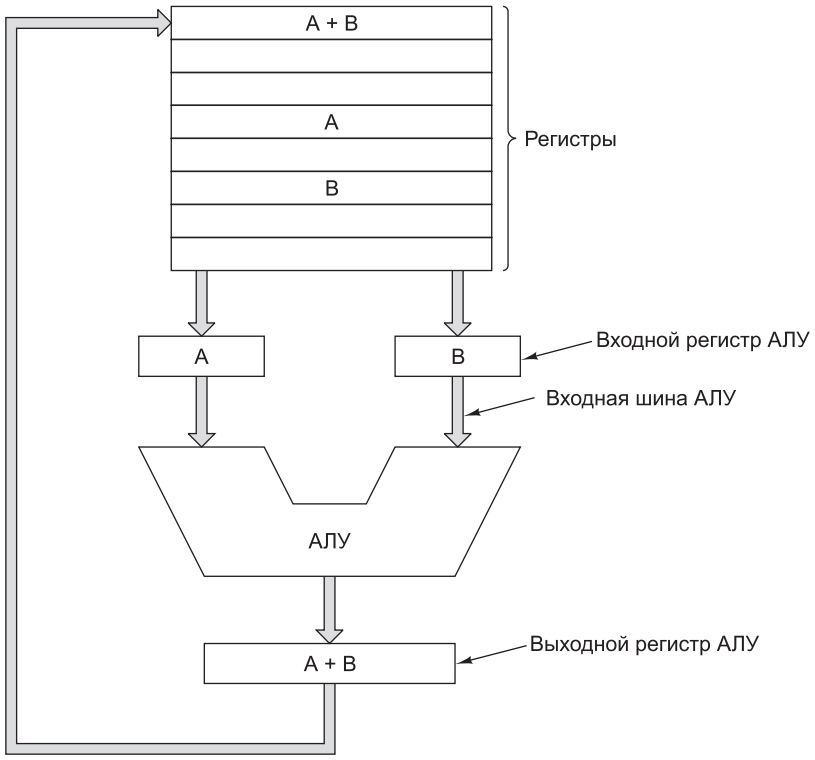

Рис. 2.2. Тракт данных обычной фон-неймановской машины

стров. Этот процесс называется **циклом тракта данных**. В какой-то степени он определяет, что может делать машина. Современные компьютеры оснащаются несколькими АЛУ, работающими параллельно и специализирующимися на разных функциях. Чем быстрее происходит цикл тракта данных, тем быстрее компьютер работает.

Выполнение команд

Центральный процессор выполняет каждую команду за несколько шагов. Он делает следующее:

1. Вызывает следующую команду из памяти и переносит ее в регистр команд.
2. Меняет положение счетчика команд, который после этого указывает на следующую команду[1].
3. Определяет тип вызванной команды.
4. Если команда использует слово из памяти, определяет, где находится это слово.
5. Переносит слово, если это необходимо, в регистр центрального процессора[2].

[1] Это происходит после декодирования текущей команды, а иногда и после ее выполнения. — *Примеч. науч. ред.*

[2] Следует заметить, что бывают команды, которые требуют загрузки из памяти целого множества слов и их обработки в рамках единственной команды. — *Примеч. науч. ред.*

6. Выполняет команду.
7. Переходит к шагу 1, чтобы начать выполнение следующей команды.

Такая последовательность шагов (**выборка** — **декодирование** — **исполнение**) является основой работы всех компьютеров.

Описание работы центрального процессора можно представить в виде программы. В листинге 2.1 приведена такая программа-интерпретатор на языке Java. В описываемом компьютере есть два регистра: счетчик команд (PC) с адресом следующей команды и аккумулятор (AC), в котором хранятся результаты арифметических операций. Кроме того, имеются внутренние регистры, в которых хранится текущая команда (`instr`), тип текущей команды (`instr_type`), адрес операнда команды (`data_loc`) и сам операнд (`data`). Каждая команда содержит один адрес ячейки памяти. В ячейке памяти хранится операнд — например, фрагмент данных, который нужно добавить в аккумулятор.

Листинг 2.1. Интерпретатор для простого компьютера (на языке Java)

```
public class Interp{
  static int PC;         // PC содержит адрес следующей команды
  static int AC;         // Аккумулятор, регистр для арифметики
  static int instr;      // Регистр для текущей команды
  static int instr_type; // Тип команды (код операции)
  static int data_loc;   // Адрес данных или -1, если его нет
  static int data;       // Текущий операнд
  static boolean run_bit = true; // Бит, который можно сбросить,
                                 // чтобы остановить машину
  public static void interpret(int memory[], int starting_address{
// Эта процедура интепретирует программы для простой машины,
// которая содержит команды только с одним операндом из
// памяти. Машина имеет регистр AC (аккумулятор). Он
// используется для арифметических действий - например,
// команда ADD суммирует число из памяти с AC. Интерпретатор
// работает до тех пор, пока не будет выполнена команда
// HALT, вследствие чего бит run_bit поменяет значение на
// false. Машина состоит из блока памяти, счетчика команд, бита
// run bit и аккумулятора AC. Входные параметры представляют собой
// копию содержимого памяти и начальный адрес.

    PC=starting_address;
    while (run_bit) {
      instr=memory[PC]; // Вызывает следующую команду в instr
      PC=PC+1;          // Увеличивает значение счетчика команд
      instr_type=get_instr_type(instr);      // Определяет тип команды
      data_loc=find_data(instr, instr_type); //  Находит данные (-1,
                                             // если данных нет)
      if(data_loc>=0)   // Если data_loc=-1, значит, операнда нет
        data=memory[data_loc];               // Вызов данных
      execute(instr_type,data);              // Выполнение команды
    }
  }

  private static int get_instr_type(int addr) {...}
  private static int find_data(int instr, int type) {...}
  private static void execute(int type, int data) {...}
}
```

Сам факт того, что можно написать программу, имитирующую работу центрального процессора, показывает, что программа не обязательно должна выполняться реальным процессором (устройством). Напротив, вызывать из памяти, определять тип команд и выполнять эти команды может другая программа. Такая программа называется **интерпретатором**. Об интерпретаторах мы говорили в главе 1.

Эквивалентность аппаратных процессоров и интерпретаторов имеет важные последствия для организации компьютера и проектирования компьютерных систем. После того как разработчик выбрал машинный язык (Я) для нового компьютера, он должен решить, разрабатывать ли ему процессор, который будет выполнять программы на языке Я, или написать специальную программу для интерпретации программ на том же языке. Если он решит написать интерпретатор, ему потребуется разработать аппаратное обеспечение для исполнения этого интерпретатора. Возможны также гибридные конструкции, когда часть команд выполняется аппаратным обеспечением, а часть интерпретируется.

Интерпретатор разбивает команды на более мелкие (элементарные). В результате машина, предназначенная для исполнения интерпретатора, может быть гораздо проще по строению и дешевле, чем процессор, выполняющий программы без интерпретации. Такая экономия особенно важна при большом количестве сложных команд с различными параметрами. В сущности, экономия проистекает из самого факта замены аппаратного обеспечения программой (интерпретатором), тогда как создание копий программного продукта обходится дешевле, чем создание копий аппаратных элементов.

Первые компьютеры поддерживали небольшое количество команд, и эти команды были простыми. Однако разработка более мощных компьютеров привела, помимо всего прочего, к появлению более сложных команд. Вскоре разработчики поняли, что при наличии сложных команд программы выполняются быстрее, хотя выполнение каждой отдельной команды занимает больше времени. (В качестве примеров таких сложных команд можно назвать выполнение операций с плавающей точкой, обеспечение прямого доступа к элементам массива и т. п.) Если обнаруживалось, что пара тех или иных команд часто выполняется последовательно, нередко вводилась новая команда, заменяющая эти две.

Сложные команды оказались лучше еще и потому, что некоторые операции иногда перекрывались. Подобные операции могли выполняться параллельно, но для этого требовалась дополнительная аппаратура. Для дорогих компьютеров с высокой производительностью приобретение такого дополнительного аппаратного обеспечения было вполне оправданным. Таким образом, у дорогих компьютеров было гораздо больше команд, чем у дешевых. Однако растущая стоимость разработки и требования совместимости команд привели к тому, что сложные команды стали использоваться и в дешевых компьютерах, хотя там во главу угла ставилась стоимость, а не быстродействие.

К концу 50-х годов компания IBM, которая лидировала тогда на компьютерном рынке, решила, что производство семейства компьютеров, каждый из которых выполняет одни и те же команды, выгоднее и для самой компании, и для покупателей. Чтобы охарактеризовать этот уровень совместимости, компания IBM ввела термин **архитектура**. Новое семейство компьютеров должно было иметь единую архитектуру и много разных моделей, отличающихся по цене и скорости,

но «умеющих» выполнять одни и те же программы. Но как построить дешевый компьютер, который сможет выполнять все сложные команды, предназначенные для высокоэффективных дорогостоящих машин?

Решением проблемы стала интерпретация. Эта технология, впервые предложенная Уилксом в 1951 году, позволяла разрабатывать простые дешевые компьютеры, которые, тем не менее, могли выполнять большое количество команд. В результате компания IBM создала архитектуру System/360, семейство совместимых компьютеров, различающихся по цене и производительности почти на два порядка. Аппаратное обеспечение, позволяющее работать без интерпретации, использовалось только в самых дорогих моделях.

Простые компьютеры с интерпретаторами команд имели свои достоинства. Наиболее важными среди них являлись:

- возможность исправлять неправильно реализованные команды «на месте» или даже компенсировать ошибки аппаратного обеспечения на уровне аппаратного обеспечения;
- возможность добавлять новые команды при минимальных затратах, причем при необходимости уже после покупки компьютера;
- возможность (благодаря структурированной организации) разработки, проверки и документирования сложных команд.

В 70-е годы компьютерный рынок быстро разрастался, новые компьютеры могли выполнять все больше и больше функций. Вследствие повышенного спроса на дешевые компьютеры предпочтение отдавалось компьютерам с интерпретаторами. Возможность разрабатывать аппаратное обеспечение с интерпретатором для определенного набора команд привела к появлению дешевых процессоров. Полупроводниковые технологии быстро развивались, низкая стоимость брала верх над высокой производительностью, и интерпретаторы стали применяться при разработке компьютеров все шире и шире. Интерпретация использовалась практически во всех компьютерах, выпущенных в 70-е годы, от мини-компьютеров до самых больших машин.

К концу 70-х годов интерпретаторы стали применяться практически во всех моделях, кроме самых дорогих машин с очень высокой производительностью (например, Cray-1 и компьютеров серии Control Data Cyber). Интерпретаторы обеспечивали реализацию сложных команд без использования дорогостоящей аппаратуры, поэтому разработчики могли вводить все более и более сложные команды, а также (и даже в особенности) расширять способы определения операндов.

Эта тенденция достигла своего апогея в компьютере VAX, разработанном компанией DEC; у него было несколько сот команд и более 200 способов определения операндов в каждой команде. К несчастью, архитектура VAX с самого начала ориентировалась на интерпретацию, а производительности уделялось мало внимания. Это привело к появлению большого количества второстепенных команд, которые сложно было выполнять непосредственно. Данное упущение стало фатальным как для VAX, так и для его производителя (компании DEC). Компания Compaq купила DEC в 1998 году (правда, тремя годами позже сама компания Compaq вошла в структуру Hewlett-Packard).

Хотя самые первые 8-разрядные микропроцессоры были очень простыми и поддерживали небольшой набор команд, к концу 70-х годов даже они стали

разрабатываться с ориентацией на интерпретаторы. В этот период основной проблемой для разработчиков стала возрастающая сложность микропроцессоров. Главное преимущество интерпретации заключалось в том, что можно было разработать очень простой процессор, а вся самое сложное реализовать с помощью интерпретатора. Таким образом, вместо разработки сложной аппаратуры требовалась разработка сложного программного обеспечения.

Успех системы Motorola 68000 с большим набором интерпретируемых команд и одновременный провал компьютера Zilog Z8000, у которого был столь же обширный набор команд, но не было интерпретатора, продемонстрировали все преимущества интерпретации при разработке новых машин. Этот успех был довольно неожиданным, учитывая, что компьютер Z80 (предшественник Zilog Z8000) пользовался большей популярностью, чем Motorola 6800 (предшественник Motorola 68000). Конечно, важную роль здесь сыграли и другие факторы — например то, что компания Motorola много лет занималась производством микросхем, а торговая марка Zilog принадлежала Exxon — крупной нефтяной компании.

Еще один фактор в пользу интерпретации — существование быстродействующих постоянных запоминающих устройств для хранения интерпретаторов (так называемых командных ПЗУ). Предположим, что для выполнения обычной интерпретируемой команды интерпретатору компьютера Motorola 68000 нужно выполнить 10 команд (они называются **микрокомандами**), по 100 нс на каждую, и произвести два обращения к оперативной памяти, по 500 нс на каждое. Общее время выполнения команды составит, следовательно, 2000 нс — всего лишь в два раза больше, чем в лучшем случае заняло бы непосредственное выполнение этой команды без интерпретации. А если бы не было специального быстродействующего постоянного запоминающего устройства, выполнение этой команды заняло бы целых 6000 нс. Шестикратное возрастание времени выполнения вынести намного сложнее.

Системы RISC и CISC

В конце 70-х годов проводилось много экспериментов с очень сложными командами, появление которых стало возможным благодаря интерпретации. Разработчики пытались уменьшить «семантический разрыв» между тем, что компьютеры способны делать, и тем, что требуют языки высокого уровня. Едва ли кто-нибудь тогда думал о разработке более простых машин, так же как сейчас мало кто (к сожалению) занимается разработкой менее мощных электронных таблиц, сетей, веб-серверов и т. д.

В компании IBM этой тенденции противостояла группа разработчиков во главе с Джоном Коком (John Cocke); они попытались воплотить идеи Сеймура Крея, создав экспериментальный высокоэффективный мини-компьютер **801**. Хотя компания IBM не занималась сбытом этой машины, а результаты эксперимента были опубликованы только через несколько лет [Radin, 1982], весть быстро разнеслась по свету, и другие производители тоже занялись разработкой подобных архитектур.

В 1980 году группа разработчиков в университете Беркли во главе с Дэвидом Паттерсоном (David Patterson) и Карло Секвином (Carlo Séquin) начала раз-

работку не ориентированных на интерпретацию процессоров VLSI [Patterson, 1985; Patterson and Séquin, 1982]. Для обозначения этого понятия они придумали термин **RISC**, а новый процессор назвали RISC I, вслед за которым вскоре был выпущен RISC II. Немного позже, в 1981 году, Джон Хеннеси (John Hennesy) в Стенфорде разработал и выпустил другую микросхему, которую он назвал **MIPS** [Hennessy, 1984]. Эти две микросхемы развились в коммерчески важные продукты SPARC и MIPS соответственно.

Новые процессоры существенно отличались от коммерческих процессоров того времени. Поскольку они были несовместимы с существующей продукцией, разработчики вправе были включать туда новые наборы команд, которые могли бы повысить общую производительность системы. Первоначально основное внимание уделялось простым командам, которые могли быстро выполняться. Однако вскоре разработчики осознали, что ключом к высокой производительности компьютера является разработка команд, которые можно быстро *запускать*. То есть не так важно, как долго выполняется та или иная команда, важнее то, сколько команд в секунду может быть запущено.

В то время когда разрабатывались эти простые процессоры, всеобщее внимание привлекало относительно небольшое количество команд (обычно около 50). Для сравнения: число команд в компьютерах VAX производства DEC и больших компьютерах производства IBM в то время составляло от 200 до 300. Компьютер RISC (Reduced Instruction Set Computer — **компьютер с сокращенным набором команд**) противопоставлялся системе CISC (Complex Instruction Set Computer — **компьютер с полным набором команд**) — слабо завуалированный намек на компьютер VAX, который доминировал в то время в университетской среде. На сегодняшний день мало кто считает, что размер набора команд так уж важен, но названия сохранились до сих пор.

С этого момента началась грандиозная идеологическая война между сторонниками RISC и «консерваторами» (VAX, Intel, мэйнфреймы IBM). По мнению первых, наилучший способ разработки компьютеров — включение туда небольшого количества простых команд, каждая из которых выполняется за один цикл тракта данных (см. рис. 2.2), то есть производит над парой регистров какую-либо арифметическую или логическую операцию (например, сложение или операцию логического И) и помещает результат обратно в регистр. В качестве аргумента они утверждали, что даже если системе RISC приходится выполнять 4 или 5 команд вместо одной, которую выполняет CISC, RISC все равно выигрывает в скорости, так как RISC-команды выполняются в 10 раз быстрее (поскольку они не интерпретируются). Следует также отметить, что к этому времени быстродействие основной памяти приблизилась к быстродействию специальных командных ПЗУ, потому недостатки интерпретации были налицо, что еще более поднимало популярность компьютеров RISC.

Учитывая преимущества RISC в плане производительности, можно было предположить, что на рынке такие компьютеры, как UltraSPARC компании Sun, должны доминировать над компьютерами CISC (Intel Pentium и т. д.). Однако ничего подобного не произошло. Почему?

Во-первых, компьютеры RISC несовместимы с другими моделями, а многие компании вложили миллиарды долларов в программное обеспечение для продукции Intel. Во-вторых, как ни странно, компания Intel сумела воплотить те же

идеи в архитектуре CISC. Процессоры Intel, начиная с процессора 486, содержат RISC-ядро, которое выполняет самые простые (и обычно самые распространенные) команды за один цикл тракта данных, а по обычной технологии CISC интерпретируются более сложные команды. В результате обычные команды выполняются быстро, а более сложные и редкие — медленно. Хотя при таком «гибридном» подходе производительность ниже, чем в архитектуре RISC, новая архитектура CISC имеет ряд преимуществ, поскольку позволяет использовать старое программное обеспечение без изменений.

Принципы проектирования современных компьютеров

Прошло уже более двадцати лет с тех пор, как были сконструированы первые компьютеры RISC, однако некоторые принципы их функционирования можно перенять, учитывая современное состояние технологии разработки аппаратного обеспечения. Если происходит очень резкое изменение в технологии (например, новый процесс производства делает время обращения к памяти в 10 раз меньше, чем время обращения к центральному процессору), меняются все условия. Поэтому разработчики всегда должны учитывать возможные технологические изменения, которые могли бы повлиять на баланс между компонентами компьютера.

Существует ряд принципов разработки, иногда называемых **принципами RISC**, которым по возможности стараются следовать производители универсальных процессоров. Из-за некоторых внешних ограничений, например требования совместимости с другими машинами, приходится время от времени идти на компромисс, но эти принципы — цель, к которой стремится большинство разработчиков.

- *Все команды должны выполняться непосредственно аппаратным обеспечением*. То есть обычные команды выполняются напрямую, без интерпретации микрокомандами. Устранение уровня интерпретации повышает скорость выполнения большинства команд. В компьютерах типа CISC более сложные команды могут разбиваться на несколько шагов, которые затем выполняются как последовательность микрокоманд. Эта дополнительная операция снижает быстродействие машины, но может использоваться для редко применяемых команд.
- *Компьютер должен запускать как можно больше команд в секунду*. В современных компьютерах используется много различных способов повышения производительности, главный из которых — запуск как можно большего количества команд в секунду. В конце концов, если процессор сможет запустить 500 млн команд в секунду, то его производительность составляет 500 MIPS, сколько бы времени ни занимало выполнение этих команд. (**MIPS** — сокращение от Millions of Instructions Per Second — миллионов команд в секунду.) Этот принцип предполагает, что параллелизм должен стать важным фактором повышения производительности, поскольку запустить на выполнение большое количество команд за короткий промежуток времени можно только в том случае, если есть возможность одновременного выполнения нескольких команд.

  Хотя команды любой программы всегда располагаются в памяти в определенном порядке, компьютер изменить порядок их запуска (так как необходимые ресурсы памяти могут быть заняты) и (или) завершения. Конечно,

если команда 1 устанавливает значение в регистр, а команда 2 использует этот регистр, нужно действовать с особой осторожностью, чтобы команда 2 не считала значение из регистра раньше, чем оно там окажется. Чтобы не допускать подобных ошибок, необходимо хранить в памяти большое количество дополнительной информации, но благодаря возможности выполнять несколько команд одновременно производительность все равно оказывается выше.

- *Команды должны легко декодироваться*. Предел количества запускаемых в секунду команд зависит от темпа декодирования отдельных команд. Декодирование команд позволяет определить, какие ресурсы им необходимы и какие действия нужно выполнить. Все, что способствует упрощению этого процесса, полезно. Например, можно использовать единообразные команды с фиксированной длиной и с небольшим количеством полей. Чем меньше разных форматов команд, тем лучше.
- *К памяти должны обращаться только команды загрузки и сохранения*. Один из самых простых способов разбить операцию на отдельные шаги — сделать так, чтобы операнды большей части команд брались из регистров и возвращались туда же. Операция перемещения операндов из памяти в регистры и обратно может осуществляться в разных командах. Поскольку доступ к памяти занимает много времени, длительность которой невозможно спрогнозировать, выполнение этих команд могут взять на себя другие команды, единственное назначение которых — перемещение операндов между регистрами и памятью. То есть к памяти должны обращаться только команды загрузки и сохранения (`LOAD` и `STORE`).
- *Регистров должно быть много*. Поскольку доступ к памяти происходит относительно медленно, в компьютере должно быть много регистров (по крайней мере 32). Если слово было однажды загружено из памяти, при наличии большого числа регистров оно может содержаться в регистре до тех пор, пока не потребуется. Возвращение слова из регистра в память и новая загрузка этого же слова в регистр нежелательны. Лучший способ избежать излишних перемещений — наличие достаточного количества регистров.

Параллелизм на уровне команд

Разработчики компьютеров стремятся к тому, чтобы повысить производительность своих машин. Один из способов заставить процессоры работать быстрее — повышение их тактовой частоты, однако при этом существуют некоторые технологические ограничения на то, что можно сделать методом «грубой силы» на данный момент. Поэтому большинство проектировщиков для повышения производительности при данной тактовой частоте процессора используют параллелизм (выполнение двух или более операций одновременно).

Существует две основные формы параллелизма: параллелизм на уровне команд и параллелизм на уровне процессоров. В первом случае параллелизм реализуется за счет запуска большого количества команд каждую секунду. Во втором случае над одним заданием работают одновременно несколько процессоров. Каждый подход имеет свои преимущества. В этом разделе мы рассмотрим параллелизм на уровне команд, а в следующем — параллелизм на уровне процессоров.

Конвейеры

Уже много лет известно, что главным препятствием высокой скорости выполнения команд является необходимость их загрузки из памяти. Для разрешения этой проблемы можно вызывать команды из памяти заранее и хранить в специальном наборе регистров. Эта идея использовалась еще в 1959 году при разработке компьютера Stretch компании IBM, а набор регистров был назван **буфером выборки с упреждением**. Таким образом, когда требовалась определенная команда, она вызывалась прямо из буфера, а обращения к памяти не происходило.

В действительности при выборке с упреждением команда обрабатывается за два шага: сначала происходит выборка команды, а затем ее выполнение. Дальнейшим развитием этой стратегии стала концепция **конвейера**. При использовании конвейера команда обрабатывается уже не за два, а за большее количество шагов, каждый из которых реализуется определенным аппаратным компонентом, причем все эти компоненты могут работать параллельно.

На рис. 2.3, *а* изображен конвейер из 5 блоков, которые называются **ступенями**. Первая ступень (блок С1) вызывает команду из памяти и помещает ее в буфер, где она хранится до тех пор, пока не потребуется. Вторая ступень (блок С2) декодирует эту команду, определяя ее тип и тип ее операндов. Третья ступень (блок С3) определяет местонахождение операндов и вызывает их из регистров или из памяти. Четвертая ступень (блок С4) выполняет команду, обычно проводя операнды через тракт данных (см. рис. 2.2). И наконец, блок С5 записывает результат обратно в нужный регистр.

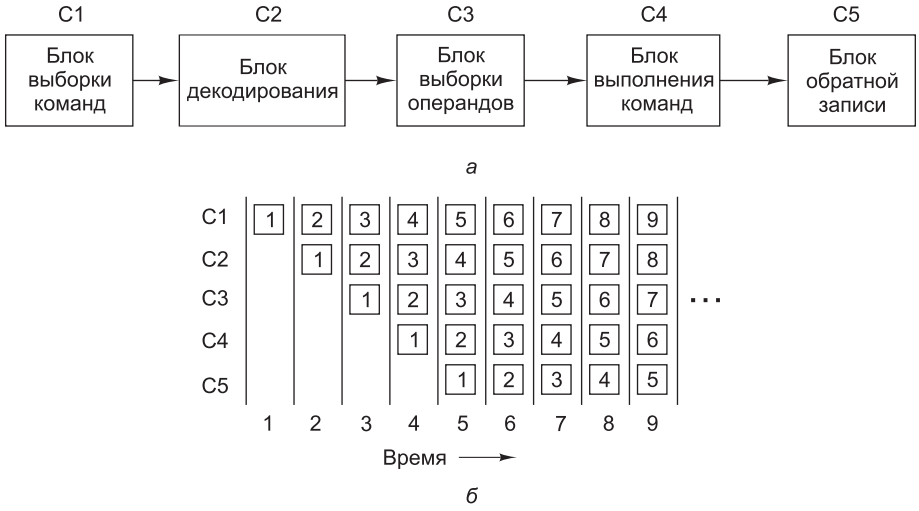

Рис. 2.3. Пятиступенчатый конвейер (*а*); состояние каждой ступени в зависимости от количества пройденных циклов (*б*). Показано 9 циклов

На рис. 2.3, *б* мы видим, как действует конвейер во времени. Во время цикла 1 блок С1 обрабатывает команду 1, вызывая ее из памяти. Во время цикла 2 блок С2 декодирует команду 1, в то время как блок С1 вызывает из памяти команду 2. Во время цикла 3 блок С3 вызывает операнды для команды 1, блок С2 декодирует команду 2, а блок С1 вызывает команду 3. Во время цикла 4 блок С4

выполняет команду 1, С3 вызывает операнды для команды 2, С2 декодирует команду 3, а С1 вызывает команду 4. Наконец, во время цикла 5 блок С5 записывает результат выполнения команды 1 обратно в регистр, тогда как другие ступени конвейера обрабатывают следующие команды.

Чтобы лучше понять принципы работы конвейера, рассмотрим аналогичный пример. Представим себе кондитерскую фабрику, на которой выпечка тортов и их упаковка для отправки производятся раздельно. Предположим, что в отделе отправки находится длинный конвейер, вдоль которого стоят 5 рабочих (или ступеней обработки). Каждые 10 секунд (это время цикла) первый рабочий ставит пустую коробку для торта на ленту конвейера. Эта коробка отправляется ко второму рабочему, который кладет в нее торт. После этого коробка с тортом доставляется третьему рабочему, который закрывает и запечатывает ее. Затем она поступает к четвертому рабочему, который ставит на ней штамп. Наконец, пятый рабочий снимает коробку с конвейерной ленты и помещает ее в большой контейнер для отправки в супермаркет. Примерно таким же образом действует компьютерный конвейер: каждая команда (в случае с кондитерской фабрикой — торт) перед окончательным выполнением проходит несколько ступеней обработки.

Возвратимся к нашему конвейеру на рис. 2.3. Предположим, что время цикла у этой машины — 2 нс. Тогда для того, чтобы одна команда прошла через весь конвейер, требуется 10 нс. На первый взгляд может показаться, что такой компьютер будет выполнять 100 млн команд в секунду, в действительности же скорость его работы гораздо выше. В течение каждого цикла (2 нс) завершается выполнение одной новой команды, поэтому машина выполняет не 100, а 500 млн команд в секунду!

Конвейеры позволяют добиться компромисса между **временем запаздывания** (время выполнения одной команды) и **пропускной способностью процессора** (количество команд, выполняемых процессором в секунду). Если время обращения составляет T нс, а конвейер имеет n ступеней, время запаздывания составит nT н.

Поскольку одна команда выполняется за одно обращение, а за одну секунду таких обращений набирается $10^9/T$, количество команд в секунду также составляет $10^9/T$. Скажем, если T = 2 нс, то каждую секунду выполняется 500 млн команд. Для того чтобы получить значение MIPS, нужно разделить скорость исполнения команд на один миллион; таким образом, $(10^9/T)/10^6 = 1000/T$ MIPS. В принципе, скорость исполнения команд можно измерять и в миллиардах операций в секунду (Billion Instructions Per Second, BIPS), но так никто не делает, и мы не будем.

Суперскалярные архитектуры

Один конвейер — хорошо, а два — еще лучше. Одна из возможных схем процессора с двумя конвейерами показана на рис. 2.4. В ее основе лежит конвейер, изображенный на рис. 2.3. Здесь общий блок выборки команд вызывает из памяти сразу по две команды и помещает каждую из них в один из конвейеров. Каждый конвейер содержит АЛУ для параллельных операций. Чтобы выполняться параллельно, две команды не должны конфликтовать из-за ресурсов (например, регистров) и ни одна из них не должна зависеть от результата выполнения другой. Как и в случае с одним конвейером, либо компилятор должен гарантировать

отсутствие нештатных ситуаций (когда, например, аппаратура не обеспечивает проверку команд на несовместимость и при обработке таких команд выдает некорректный результат), либо конфликты должны выявляться и устраняться дополнительным оборудованием непосредственно в ходе выполнения команд.

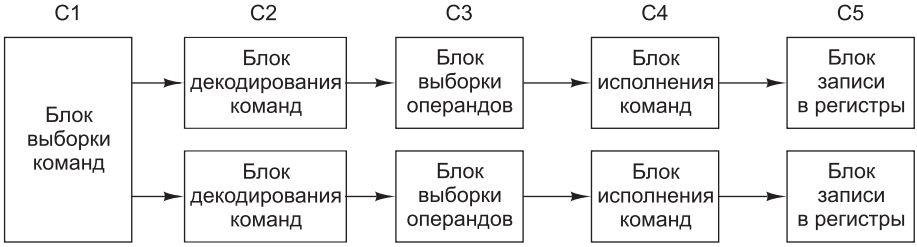

Рис. 2.4. Сдвоенный пятиступенчатый конвейер с общим блоком выборки команд

Сначала конвейеры (как сдвоенные, так и обычные) использовались только в RISC-компьютерах. У процессора 386 и его предшественников их не было. Конвейеры в процессорах компании Intel появились, только начиная с модели 486[1]. Процессор 486 имел один пятиступенчатый конвейер, а Pentium — два таких конвейера. Похожая схема изображена на рис. 2.4, но разделение функций между второй и третьей ступенями (они назывались декодер 1 и декодер 2) было немного другим. Главный конвейер (**u-конвейер**) мог выполнять произвольные команды. Второй конвейер (**v-конвейер**) мог выполнять только простые команды с целыми числами, а также одну простую команду с плавающей точкой (FXCH).

Проверка совместимости команд для параллельного выполнения осуществляется по жестким правилам. Если команды, входящие в пару, были сложными или несовместимыми, выполнялась только одна из них (в u-конвейере). Оставшаяся вторая команда сопоставлялась со следующей командой. Команды всегда выполнялись по порядку. Специальные компиляторы для процессора Pentium объединяли совместимые команды в пары и могли генерировать программы, выполняющиеся быстрее, чем в предыдущих версиях. Измерения показали, что программы, в которых применяются операции с целыми числами, при той же тактовой частое на Pentium выполняются вдвое быстрее, чем на 486 [Pountain, 1993]. Выигрыш в скорости достигался благодаря второму конвейеру.

Переход к четырем конвейерам возможен, но требует громоздкого аппаратного обеспечения (отметим, что компьютерщики, в отличие от фольклористов, не верят в счастливое число три). Вместо этого используется другой подход. Основная идея — один конвейер с большим количеством функциональных блоков, как показано на рис. 2.5. Intel Core, к примеру, имеет сходную структуру (подробно мы рассмотрим ее в главе 4). В 1987 году для обозначения этого подхода был введен термин **суперскалярная архитектура** [Agerwala and Cocke, 1987]. Однако подобная идея нашла воплощение еще более 40 лет назад в компьютере CDC 6600. Этот компьютер вызывал команду из памяти каждые 100 нс и помещал ее в один из 10 функциональных блоков для параллельного выполнения. Пока команды выполнялись, центральный процессор вызывал следующую команду.

[1] Необходимо отметить, что параллельное функционирование отдельных блоков процессора имело место и в предыдущем микропроцессоре (386). Этот механизм стал прообразом 5-ступенчатого конвейера микропроцессора 486. — *Примеч. науч. ред.*

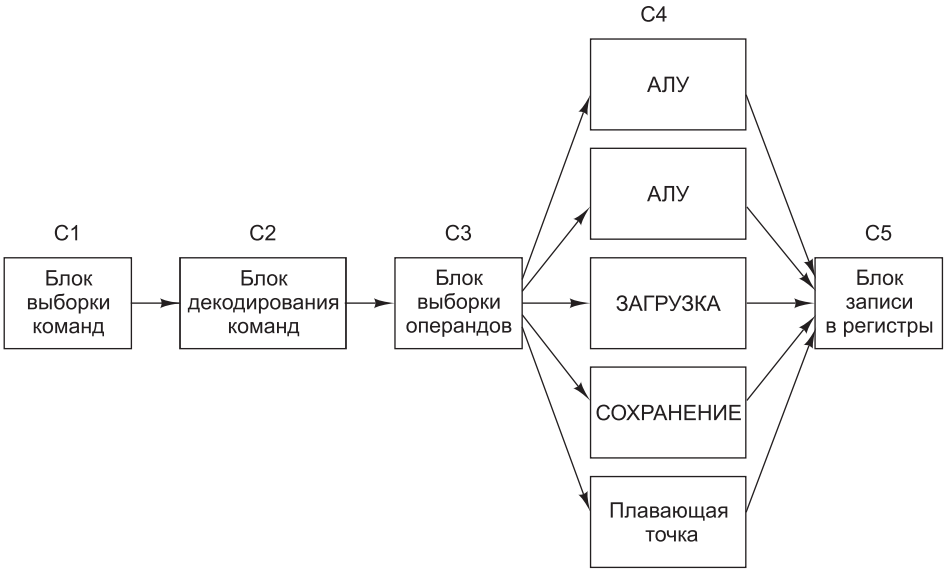

Рис. 2.5. Суперскалярный процессор с пятью функциональными блоками

Со временем определение «суперскалярности» несколько изменилось. Теперь суперскалярными называют процессоры, способные запускать несколько команд (зачастую от четырех до шести) за один тактовый цикл. Естественно, для передачи всех этих команд в суперскалярном процессоре должно быть несколько функциональных блоков. Поскольку в процессорах этого типа, как правило, предусматривается один конвейер, его устройство обычно соответствует рис. 2.5.

В соответствии с этим определением компьютер 6600 формально не был суперскалярным с технической точки зрения — ведь за один тактовый цикл в нем запускалось не больше одной команды. Однако при этом был достигнут аналогичный результат — команды запускались быстрее, чем исполнялись. На самом деле, разница в производительности между ЦП с циклом в 100 нс, передающим за этот период по одной команде четырем функциональным блокам, и ЦП с циклом в 400 нс, запускающим за это время четыре команды, трудноуловима. В обоих процессорах соблюдается принцип превышения скорости запуска над скоростью управления; при этом рабочая нагрузка распределяется между несколькими функциональными блоками.

Отметим, что на выходе ступени 3 команды появляются значительно быстрее, чем ступень 4 способна их обрабатывать. Если бы на выходе ступени 3 команды появлялись каждые 10 нс, а все функциональные блоки делали свою работу также за 10 нс, то на ступени 4 всегда функционировал бы только один блок, что сделало бы саму идею конвейера бессмысленной. В действительности большинству функциональных блоков ступени 4 (точнее, обоим блокам доступа к памяти и блоку выполнения операций с плавающей точкой) для обработки команды требуется значительно больше времени, чем занимает один цикл. Как видно из рис. 2.5, на ступени 4 может быть несколько АЛУ.