Информатика и компьютерные науки.

Информатика - это наука и техника, связанные с машинной обработкой, хранением и передачей информации. Поэтому центральное понятие в информатике - информация. Точное выяснение понятия "информация" существенно необходимо для понимания систем обработки информации. Вообще, понятие "информация" используется нами в разных смыслах. Мы обычно под информацией понимаем высказывания относительно событий, взаимосвязей или состояний реального мира. Н/р "Волга впадает в Каспийское море."

В информатике информацией называется абстрактное содержание какого-либо высказывания, описания, указания (оператора), сообщения. Необходимо отличать информацию, т.е. абстрактное содержание от изображения информации. Н/р математическое понятие "целое число" является абстрактным понятием. Мы изображаем целые числа в виде последовательности цифр из множества {0, 1,... 9}. Можно изобразить целое число римскими цифрами или палочками, насечками. Все это изображения, т.е. внешние формы информации - представления.

Опр. Информацией называют абстрактное содержание ("содержательное значние", "семантика") какого-либо высказывания, описания, указания сообщения или известия. Внешнюю форму изображения называют представлением.

Для машинной обработки информации существует много форм представления информации от условных знаков (сигналов) и произносимых слов до рисунков или последовательностей символов. Все представления (изображения) информации не будут иметь смысла, если не будет известно о значениях представлений. Н/р древние надписи и рисунки. Это внешняя форма. Однако нам неизвестны значения этих изображений, поэтому нам недоступен их смысл, т.е. абстрактное значение информации. Поэтому важно установить способ для выявления значения представления.

Опр. Переход от представления к абстрактной информации, т.е. к значению представления, называют интерпретацией.

Многие формы представления информации допускают различное толкование. Н/р "красный". Только в том случае, когда существуют единые, согласованные интерпретации, возможен обмен информацией. Н/р дорожные знаки.

С другой стороны одно и то же абстрактное содержание м.б. представлено различными способами. Н/р числа.

Выявление подходящих систем представления (языков) для определенных классов информации является одной из задач информатики.

Вопрос об отношении информации к реальности, т.е. является ли информация истинной не рассматривается и не решается в информатике, т.к. ответ на него зависит от субъективного восприятия. Итак, мы различаем в связи с информацией:

- ее представление или изображение (внешняя форма)

- ее значение (собственно "абстрактная" информация)

- ее отношение к реальному миру (связь абстрактной информации с

действительностью.

Информатика включает в себя науку о машинной обработке информации и поэтому в ней рассматриваются вопросы:

- схематизированного представления (изображения) информации: структуры объектов и данных, а также их взаимосвязи;

- правил и предписаний для обработки информации (алгоритмы, вычислительные предписания) и их представления, включая описание протекания работы (процессы).

Оба эти пункта тесно связаны между собой. Программа, например, в качестве своей внешней формы имеет текстовую или графическую структуру. Эта структура, в свою очередь, является объектом для информационной обработки. Но программа, также, представляет собой предписание для обработки информации. При ее выполнении в компьютере протекает процесс действий, который преобразует определенные исходные данные в определенный результат.

В информатике рассматриваются информационные системы вида (A, R, I). Где R - множество представлений с интерпретацией I во множестве A элементов (информаций). Т.о. интерпретации соответствует отображение: I : R -> A (интерпретация I ставит в соответствие данному представлению r некоторое абстрактное информационное содержание I[r].) R также называют системой представления, а A - семантической моделью.

Пример (система представления для натуральных чисел). Пусть N - множество нат. чисел (включая число "нуль"), представляемых числом штрихов, т.е.:

ε, ⏐, ⏐⏐, ⏐⏐⏐ …

где через ε обозначена пустая последовательность. Интерпретацией I будет отображение десятичного представления в последовательность штрихов.

I : {0, 1, …, 9}+ → N

I[0]=ε, I[1]= ⏐, I[2]= ⏐⏐, …

Этот пример демонстрирует фундаментальную проблему информационной обработки: информация в ее абстрактном виде не может быть записано непосредственно, она всегда может быть только изображена.

Часто в какой-нибудь системе представления имеется много различных изображений одной и той же информации. Эти изображения называются эквивалентными. Точнее говоря, в информационной системе (A, R, I) справедливо: два изображения r1, r2∈R называются семантически эквивалентными, если они несут одинаковую информацию: I[r1]=I[r2]

Как правило, мы больше заинтересованы в информации, чем в ее представлении. Поэтому часто бывает удобным обходиться с представлением так, как если бы оно было непосредственно информацией. Например: классическая математика.

Обработка информации означает, строго говоря, обработку или преобразование информации. Для этого необходимо, чтобы в применяемой информационной системе была представима любая информация.

Пусть (A, R, I) - информационная система. Если I сюръективно, т.е. для каждой информации а∈А существует представление r∈R с I[r]=a, то каждая информация имеет представление. Обычно представляют интерес ИС обладающие этим свойством.

Отображение на множестве представлений при определенных предположениях индуцирует и отображение информации. Пусть f: R→R отображение на множестве представлений R. Если для всех x, y∈R справедливо:

I[x]=I[y] → I[f(x)]=I[f(y)] и I сюръективно, то вследствие интерпретации I:R→A однозначным образом устанавливается отображение информации f' : A→A по следующему правилу:

f'(a)=b, если для r∈R справедливо I[r]=a и I[f(r)]=b.

Связь между f и f' и интерпретацией пояснить коммутирующей диаграммой:

R I A

f f'

R I A

Также справедливо I[f(r)]=f'(I[r]). f' называют абстракцией f.

Итак информация представляется не непосредственно, а лишь изображается каким-либо образом. Однако не все эквивалентные изображения определенной информации одинаково легко интерпретируются или обрабатываются.

Пример: Σ1/2^i, 0.(9), 0!, 1

Все эти термы имеют значение "один". Однако они различаются с точки зрения простоты, чтения, интерпретации и понимания.

Простота конкретного изображения информации имеет важное значение. Часто подмножество S (изображений простой внешней формы) изображений R выделяется как множество нормальных форм. Тогда S называют системой нормальных форм(СНФ). Если в такой системе для каждого изображения существует по меньшей мере 1 семантически эквивалентная НФ, то СНФ называется полной.

Пусть S⊆R - СНФ. Если любое множество изображений с одинаковой интерпретацией имеет единственную НФ, т.е. отображение IS - инъективно, то СНФ называется однозначной.

Т.к. на множестве однозначных НФ интерпретация есть инъективное отображение, то соответствующую информацию по мере надобности можно отождествить с ее НФ.

Формальное понятие алгоритма. Рекурсивные функции. Системы текстовых замен.(СТЗ)

Алгоритмами являются способы решения, описанные с помощью предписаний по обработке, которые удовлетворяют определенным требованиям.

Опр. Алгоритм - это способ с точным (т.е. выраженным в точно определенном языке) конечным описанием применения эффективных (т.е. практически выполнимых) элементарных шагов (переработки).

Это интуитивное понятие алгоритма.

Алгоритмы типичным образом решают не только частные задачи, но и классы задач. Подлежащие решению частные задачи, выделяемые по мере надобности из рассматриваемого класса, определяются с помощью параметров. Параметры играют роль исходных данных для алгоритма. Алгоритмы, как правило, по этим исходным данным доставляют результаты. Эти результаты в случае задач информационной обработки могут быть информацией (вернее, представлением информации) или последовательностью указаний (управляющих сигналов), по которым осуществляются определенные преобразования.

Независимо от формы описания для алгоритмов важно различать следующие аспекты:

постановку задачи, которая должна быть решена с помощью алгоритма;

специфичный способ, каким решается задача, при этом для алгоритма различают:

а) элементарные шаги обработки, которые имеются в распоряжении;

б) описание выбора отдельных подлежащих выполнению шагов.

Свойства алгоритма.

Массовость.

Дискретность - алгоритм представляется в виде конечной последовательности шагов.

Конечность.

Определенность.

Эффективность.

Пример: алгоритм Евклида.

если а=в то НОД(а, в)=а

если а>в то НОД(а, в)=НОД(а-в, в)

если а<в то НОД(а, в)=НОД(а, в-а)

Формальное понятие алгоритмов тесно связано с понятиями рекурсивных функций, машин Тьюринга, нормальных алгоритмов Маркова (или СТЗ).

Алгоритм называется терминистическим, если он завершается за конечное число шагов. Детерминистическим, если нет свободы в выборе очередного шага алгоритма. Детерминированным, если независимо от последовательности выполняемых шагов, результат определяется однозначно.

Существование или не существование алгоритма может быть установлено, если найти такой математический объект, который будет существовать в точности тогда, когда и алгоритм. Таким математическим объектом может быть рекурсивная функция. Функция определяется однозначно, если известен закон, по которому каждому набору х1, …, хn ставится в соответствие 1 значение функции y. Закон может быть произвольным, в т.ч. это может быть алгоритм вычисления значения функции. В этом случае функцию называют вычислимой, а алгоритм, по которому вычисляется функция, называется алгоритмом, сопутствующим рекурсивной функции. Рекурсивные функции являются частным классом вычислимых функций.

РФ строится здесь на множестве целых неотрицательных чисел следующим образом: задаются 3 базовые РФ, для которых сопутствующие алгоритмы - одношаговые. Затем используются 3 приема, называемые операторами подстановки, рекурсии и минимизации, с помощью которых на основе базовых функций строятся более сложные РФ. По существу эти операторы - алгоритмы, комбинируя которые получают более сложные алгоритмы.

Простейшие базовые функции:

Функция произвольного количества аргументов тождественно =0. Знак функции ϕn - где n - количество аргументов.  
Сопутствующий алгоритм: если знак функции ϕn то значение функции 0.  
Например: ϕ1(3)=0, ϕ3(4, 56, 78)=0;

Тождественная функция. Знак функции ψn,i . 0<i<=n. n - количество аргументов, i - номер аргумента. Сопутствующий алгоритм: если знак функции ψn,i то значение функции - значение i-го аргумента, считая слева направо.  
Например: ψ3,2(3,22,54)=22;

Функция получения последователя. Функция одного независимого аргумента. Знак функции - λ. Сопутствующий алгоритм: если знак функции λ то значение функции - число, следующее за значением аргумента.  
Например: λ(5)=6 или 5'=6.

3 приема построения сложных РФ.

Оператор подстановки. F(f1, …, fn).  
Вычисляются значения функций f1, …, fn и используются как аргументы при вычислении F.  
τ(y)= λ(λ(y))= λ(y')=y''.

Оператор рекурсии R. f::= R[f1, f2, x(y)].

f - функция n аргументов, f1 - n-1 аргумента, f2 - n+1 аргумента, причем n-1 аргументов функций совпадают, а 2 следующих аргументов называются дополнительными. Один из них - х - называется главным доп. аргументом(ГДЭ). Он войдет в определяющую функцию. Другой y - вспомогательный доп. аргумент(ВДЭ), использующийся при построении новой функции.

Говорят, что оператор рекурсии строит новую функцию по 2 условиям:

f(0)=f1, f(i')=f2(i, f(i)).

Значением искомой функции при нулевом значении ГДЭ считать значение функции f1, а значением новой функции для каждого последующего значения ГДЭ считать значение функции f2 для предыдущего значения ГДЭ и для значения ВДЭ, совпадающего со значением искомой функции на предыдущем шаге.

Например:

PR(x) ::= R[ϕ0,ψ2,1(x,y),x(y)]

PR(0) = ϕ0 = 0

PR(1) = ψ2,1(0,PR(0))= 0

PR(2) = ψ2,1(1,PR(1)) = 1

Оператор минимизации или построение по первому нулю.

f ::=μ[f1, (x)] f(x1, …, xn)=μ(f1(x1, …, xn, y), (y))

с помощью функции f1 n+1 аргументов и дополнительного исчезающего аргумента.

Придавать последнему аргументу значения начиная с нуля, вычисляя при этом значение функции f1. Как только значение функции f1=0 значение дополнительного аргумента принимаем за значение искомой функции для тех главных аргументов, для которых вычислялось значение функции f1.

Например:

r(x)::=μ(ψ2,1(x, y), (y)]

r(0)=0

r(i) = ψ2,1(i, y) - не определено для i<>0.

Все базовые функции и построенные без оператора минимизации определены для всех значений дополнительных аргументов. Функции, построенные с помощью оператора минимизации могут быть определены не для всех значений исходных данных. Большинство известных математических функций - рекурсивные.

Например:

y=x+1 - совпадает с базовой.

w=x+y - подстановка в (z+1) вместо z функции ψ3,3 (x, y, z). Получим f\*(x, y, z)

Затем воспользуемся следующим:

S(x, y) ::= f1[ψ1,1(x), f\*(x, y, z); y(z)]

S(x, 0) =ψ1,1 (x) = x

s(x, 1) = f\*(x, 0, S(x, 0)) = s(x, 0) + 1

Класс вычислимых функций исчерпывается классом РФ. Каков бы ни был алгоритм, перерабатывающий последовательность целых неотрицательных чисел в целые неотрицательные числа найдется сопутствующий некоторой РФ алгоритм, эквивалентный данному и наоборот. Если нельзя построить РФ, то нельзя разработать алгоритм решения задачи.

. Формальное описание алгоритма через замену текстов

Для точного описания алгоритма (которое допускает машинную обработку, переработку и выполнение) требуется формальный язык (подмножество из V\* с заданным набором знаков V) для записи алгоритмов и точное определение понятия эффективности (выполнимости) элементарных шагов переработки. В простейшем случае алгоритмы в качестве входа и выхода используют слова над некоторым набором знаков. Поскольку в виде слов может быть представлена самая различная информация, можно считать, что алгоритмы всегда оперирует со словами.

Одной из простейших концепций элементарных шагов переработки последовательностей знаков является замена определенных подслов (образцов) в обрабатываемом слове другими словами. Эта концепция ведет к алгоритмам в форме систем текстовых замен на последовательнос­тях знаков.

Пусть V - запас знаков. Пара (v, w) e V\* х V\* называется заменой над V. Замена часто записывается в виде

v → w

Конечное множество R замен будем в дальнейшем называть системой текстовых замен (СТЗ) над V. Элементы этой системы будем называть также правилами текстовых замен (ПТЗ). СТЗ служат для представления алгоритмов. Отдельные шаги этих алгоритмов состоят, таким образом, в применении правил замен.

Замена

s → t

называется применением правила v → w, если имеются слова a, v, w, z ∈ V\* такие, что справедливо

s = а ° v ° z, t = а ° w ° z.

Слово s ∈ V\* называется терминальным (или терминалом} в R, если не существует слова t ∈ V\* такого, что справедливо следующее: замена

s→t

является применением какого-либо правила из R. Таким образом, к терминальному слову s нельзя больше применить никакого правила замены.

Через повторное применение ПТЗ, исходя из начально заданного слова t0, возникают вычисления. Если t0, t1, ..., tn принадлежат V\* и

ti-->ti+1

есть применение правила г из R для всех i,0<=i<=n, то последо­вательность (tj) 1<=i<=n называют (конечным) вычислением (последовательностью вычислении) над R для t0. Часто вычисление записывается следующим образом:

t0 → t1 →…→ tn

Слово t0 называется также входом для вычисления. Если tn есть терминал, то вычисление называется завершающимся (конечным) с результатом tn. Слово tn называется также выходом для R при входе t0. Бесконечное вычисление (последовательность вычислений) (ti)i∈N из слов ti ∈V\*, для которых ti-->ti+1 есть применение правил замен из R для всех i ∈ N, называется незавершающимся (бесконечным) вычислением.

Пример (вычисления по СТЗ).

(1) Для системы текстовых замен Q над множеством символов {L, О}, которая состоит из следующих правил:

LL → е, О → е, через последовательность

LOLL → LO → L

задается завершающееся вычисление для входного слова <LOLL> с результатом <L>.

(2) Для СТЗ над {L, О}, состоящей из правил О → OO, О→ L,

для входа <O> последовательность вычислений О → OO → OL → LL

является завершающимся вычислением с выходом <LL>, а последовательность

о ->. оо -> ооо ->. оооо ->...

является незавершающимся вычислением.

Система текстовых замен R в силу следующего предписания определяет алгоритм текстовых замен (АТЗ), который использует слова над V в качестве входа и выхода. Для входного слова t ∈ V\* алгоритм работает следующим образом:

"Если одно из правил множества R применимо к слову t (т. e. существует слово s ∈ V\*, для которого имеет место: l -» s есть применение правила из R), то примени правило к t и затем примени снова этот же алгоритм к слову s; в противном случае прекрати выполнение алгоритма".

Слово t служит входом для алгоритма; если (после конечного числа шагов) возникает терминальное слово, т. e. слово, к которому неприменимо никакое правило, то это слово является выходом (результатом вычислений). Если такая ситуация никогда не возникает, то алгоритм не завершается. Алгоритмы, определенные таким образом с помощью СТЗ, всегда являются последовательными. При этом выбор применяемого правила является недетермннистическим.

Часто для АТЗ в качестве входов используют только слова совершенно определенной формы (нормальная форма). Определенные знаки не входят в эти слова (а также и в выходные слова) - эти знаки, используются исключительно как вспомогательные знаки в словах, возникающих в процессе вычислений.

Примеры (алгоритмы текстовых замен).

(1) Сложение двух натуральных чисел, представленных в виде количества штрихов

Натуральное число представляется в виде количества штрихов с ограничительными скобками, т. e. число n∈N представляется словом <| |...|>, причем внутрь скобок входит n штрихов. В этом случае алгоритм состоит из одного единственного правила замены (е обозначает пустое слово):

> + < → е

Для входа <|...|> + <|...|> алгоритм дает сумму штрихов.

(2) Умножение двух натуральных чисел (в таком же представлении)

Применяются вспомогательные знаки d, e, m. Алгоритм состоит из следующих правил замен:

|>\*< → >\*<d

d| →. |md

dm → md

d> → >

<>\*< → <e

e| → e

em → |e

e> → >

Для входного слова <|...|> \* <|...|> с п1 штрихами в первом операнде и п2 штрихами во втором операнде алгоритм дает выходное слово <|...1> с п1 \* п2 штрихами.

Последовательности, образованные из вспомагательных знаков, пред­ставляют вполне определенные ситуации в вычислениях. Возникающие слова могут трактоваться снова как (усложненные) представления чисел.

Для входа <||> \* <|||> возникает показанный на рисунке граф возможных вычислении. Все вычисления заканчиваются получением слова <|||||| >. Этот алгоритм подстановки недетерминистический, но детерминированный, т. е. дает, несмотря на различные вычисления, всегда один и тот же результат.



Алгоритмы в виде текстовых замен в общем случае являются недетерминистическимн и недетерминированными. Для входного слова t суще­ствуют, как правило, многие разные вычисления с различными результа­тами. При этом для одной и той же задачи могут существовать как завер­шающиеся, так и незавершающиеся вычисления.

2.1.3. Детерминистические алгоритмы текстовых замен

Часто предпочитают рассматривать детерминистические АТЗ, т. е. алгоритмы, которые для каждого входного слова однозначно задают вычисления и тем самым в случае их завершения порождают вполне определенный результат. Это может быть обеспечено, например, установлением приоритетов применения правил. Такие приоритеты могут быть заданы просто порядком описаний правил.

Примером детерминистических алгоритмов являются так называемые алгоритмы Маркова (по имени русского математика А. А. Маркова). В алгоритмах Маркова правила замен линейно упорядочены (этот порядок определяется последовательностью описания правил). Тогда применение правил устанавливается следующим образом.

Определение (марковская стратегия применения). Если применимо несколько правил, то фактически применяется то из этих правил, которое в описании алгоритма встречается первым. Если правило применимо в нескольких местах обрабатываемого слова, то выбирается самое левое из этих мест.

Таким образом, марковские алгоритмы всегда являются детерминиро­ванными и детерминистическими. В частности, справедливо следующее:

- каждое вычисление по марковской стратегии является также общим вычислением в СТЗ;

- для каждого входного слова существует точно одно конечное или же бесконечное марковское вычисление; алгоритмы Маркова являются детерминистическими (а отсюда результат, если он существует, однозначно определен).

Пример (алгоритмы Маркова). Пусть задана система текстовых замен R на множестве символов {v, ~, true, false, (, )} для редукции булевских термов, которые построены только из символов данного множества, к нормальной форме. Эта система состоит из следующих правил:

(1) ~ ~ → е

(2) ~ true → false

(3) ~.false → true

(4) (true) → true

(5) (false) → false

(6) false v → e

(7) v false → e

(8) true v true → true

Алгоритм, определенный данной системой, работает по марковской стратегии корректно для замкнутых булевских термов (т. е. замкнутые булевские термы переводятся в семантически эквивалентные однознач­ные нормальные формы, состоящие из true и false). Это справедливо лаже для замкнутых термов с неполностью расставленными скобками. Однако в этом случае возможны такие применения, которые не являются эквивалентными преобразованиями. Для терма

-.true v true по марковской стратегии получаются вычисления

-.true v true → (правило (2) )

false v true → (правило (6) )

true

В общей недетерминистической стратегии дополнительно получают (в смысле постановки задачи корректное) вычисление

- true v true → (правило (8) )

- true → (правило (2))

false

Благодаря марковской стратегии однозначно определяется выбор приме­няемого правила, что для многих задач упрощает формулирование алгоритма. В определенных случаях может также оказаться полезным введение частичного упорядочения правил (частичный порядок над правилами замен).

2.1.4. Отображения, индуцируемые алгоритмами текстовых замен

Путем сопоставления выходного слова каждому входному слову при конечном вычислении детерминистические алгоритмы вычисляют час­тичные функции. Функции являются частичными, так как иногда при некоторых исходных данных алгоритмы не завершаются и потому результат вычислений не определен. Это имеет место также и для АТЗ. Явного использования частичных функций можно избежать путем введения особого символа -⊥ ("дно"), который символизирует отсутству­ющий "результат" незавершающегося ("расходящегося") вычисления.

Каждый детерминированный алгоритм R в форме СТЗ на после­довательностях символов V\* определяет отображение:

fR: V\* → V\* U ⊥ вследствие следующих правил. Пусть справедливо:

(1) fR (t) = г, если слово г есть результат вычислений по R для входного слова t;

(2) fR (t) = ⊥, если вычисление по R для входного слова t не заканчи­вается.

Тогда мы говорим: алгоритм R вычисляет функцию fp .

Обратим внимание, что для слов t, для которых выполнение СТЗ не завершается, отображение fR дает результат ⊥. Символ ⊥, таким образом, обозначает (псевдо)результат незавершающегося вычисления. С помо­щью введения этого символа обходят явную работу с частичными отображениями.

Если слова t ∈ V\* понимать как представления определенных инфор­мации из множества А, т. е. существует функция интерпретации такая, что (А, V\*, I) образует информационную систему, и если функция fR , индуцируемая алгоритмом R, согласована с интерпретацией, то R индуцирует также функцию между информациями, т. е. R индуцирует отображение интерпретаций.

Пример (умножение чисел, представленных количеством штрихов). Пусть интерпретация числа штрихов определяется отображением

I:{<, |,>}\* → N

с 1(<|...|>) = п для слова <|...|> с п штрихами. Тогда алгоритм умножения так представленных чисел со входом

<|...|> \* <|...|> —» <|...|>

n m n\*m

I↓ I↓ I↓

mult (n, m) = n\*m

индуцирует отображение пары чисел на их произведение» т. е. отображе­ние умножения.

Детерминистические АТЗ порождают частичные отображения на словах и тем самым, поскольку слова служат для представления информации и отображение согласовано с интерпретацией, частичные отображения ме­жду соответствующими информациями. Недетерминированные алгорит­мы определяют отношения. С системой замен R на V мы связываем от­ношение (граф, который вычисляется через алгоритм)

GR ⊆ V\*x(V\* U ⊥), определяемое следующим образом:

GR = { (t, r) ∈ V\* х V\* : г - выходное слово вычисления по R с входным словом t}

U {(t, ⊥) ∈ V\* х {⊥} : существует бесконечное вычисление по R для входного слова t}.

Обратим внимание, что в отношении gr для каждого входа t существует выход r ∈ V\*U{⊥}.

Пример (недетерминистический алгоритм с неоднозначным результатом). Пусть вход имеет форму ∠<|||...|>. Пусть задана система замен R с правилами:

∠< → <∠

∠| → |∠|

∠| → ||

∠> → |>

Так заданный алгоритм R по любому натуральному числу n (представлен­ному в виде числа штрихов) порождает натуральное число m, большее, чем n, либо не завершается. На рис. 2.2 приведена схема дерева всевозможных вычислений для входа V<| ||>.

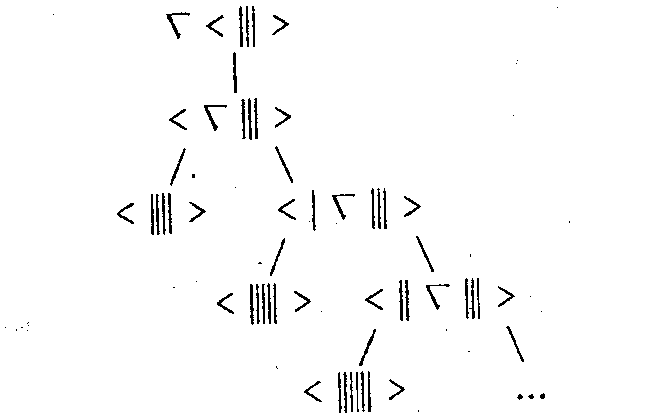


Рис. 2.2. Дерево вычислений

Получается следующее отношение (при интерпретации последователь­ности штрихов как натурального числа):

GR = { (n, m): n ∈ N^((m ∈ N ^ n<m) v m=⊥

Детерминистические алгоритмы вычисляют функции. Это ставит вопрос о том, могут ли все математические функции вычисляться с помощью алгоритмов. Один из фундаментальных результатов современной математической логики принадлежат логику Курту Геделю и отвечает на вопрос о вычислимых функциях, представимых через алгоритмы. Упрощая фopмулировку, результат Геделя говорит, что не всякая функция допускает ее вычисление с помощью алгоритма. Те функции, для вычисления которых может быть задан алгоритм, называют вычислимыми.

Имеется много различных формализации понятия "алгоритм" Некоторыми примерами этого являются:

- текстовые и термовые замены (редукция);

- рекурсивные функции;

-(тьюринговые) машины (а также регистровые машины).

Однако все эти формализации ведут к тому же самому понятию "вычис­лимые функции". Каждая вычислимая функция может быть вычислена с помощью АТЗ.

2.2. Вычислительные структуры

Алгоритмы работают над элементами данных, которые могут быть объе­динены в так называемый носитель (множества данных). Для формулиро­вания алгоритмов, наряду с используемыми элементами данных, весьма существенны имеющиеся в распоряжении эффективные функции над этими элементами. Фигурирующие в алгоритмах носители и операции могут трактоваться вместе как вычислительные структуры. Вычислительная структура охватывает тем самым семейство носителей (данные) и семейство отображений между этими носителями. Вычислительные структуры обнаруживаются в самых различных проявлениях. К примеру, карманный калькулятор, так же как и мощная ЭВМ, могут математичес­ки восприниматься и описываться как вычислительные структуры.

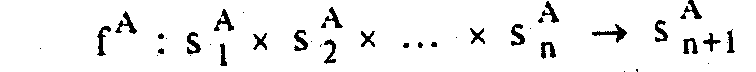
2,2.1, Семейства функций и множеств как вычислительные структуры

Понятие вычислительной структуры близко к понятию математической структуры или алгебры. Вычислительная структура состоит из семейства множеств, называемых носителями, и семейства отображений между носителями.

Определение (вычислительная структура). Пусть S и F - множества обо­значений; вычислительная структура А состоит из семейства {sА: s ∈ S} носителей sА и семейства {fА: f ∈ F} отображений fА между этими носителями. Мы пишем:

А=({sА: s ∈ S}, {fА: f ∈ F}).

Элементы s ∈ S есть обозначения для носителей и называются типами. Элементы f ∈ F есть обозначения для отображений и называются символами функции или знаками операций. Для каждого f ∈ F существует 1 одно n ∈ N такое, что имеет место: fА есть n-местная функция, и существуют типы s1…. sn+1 ∈ S такие, что



Может также быть и n = 0, т. e. допускаются и "нульместные" отображе­ния. Это такие отображения, которые (с пустым списком аргументов) получают в точности один элемент из области значений.

Для устранения частичных отображений снова используется специаль­ный элемент ⊥ ("дно") для представления неопределенного значения функции. Пусть М - множество, не содержащее ⊥. Множество M1 определяется как

M'=M U {⊥}.

Элемент ⊥ представляет "неопределенный" результат функции, например в случае незавершающегося алгоритма.

Отображение f:M1 x...x Mn → Mn+1

называется строгим, когда справедливо: если одним из аргументов функции является ⊥, то результат функции тоже есть ⊥. Это соответствует простому предположению, что результат применения функции к списку аргументов определен только в том случае, когда определены все аргументы. Распространение частичных отображений на все отображения путем добавления ⊥ к носителям приводит к строгим отображениям.

Пример

(1) Вычислительная структура BOOL булевских значений Множество S типов вычислительной структуры BOOL задано так:

S = {Ьоо1}. Множество F символов функций структуры BOOL задано так:

F = {true, false, -, v, ^}. Множество носителей В1 сопоставлено типу Ьоо1, т. e. имеет место

boo|BOOL=B'= {L, О, ⊥}.

Символы из F обозначают следующие функции:

trueBOOL: →. B',

faiseBOOL: →B',

-bool : B' → B' (бесскобочный префикс),

^ ВооL: B' х В' → В' (инфикс),

v BOOL: B' х В' → В' (инфикс).

Причем для a, b ∈ В имеет место

trueBOOL = L,

falseBOOL =o,

- bool b = not(b),

a vBOOL b = or(a, b),

а ^BOOL b = and(a, b).

Функции являются строгими и потому их значения для случая, когда один из аргументов есть ⊥, также установлены.

2.2.2. Сигнатуры

Чтобы установить множество символов функций и типов, которые встре­чаются в вычислительной структуре, а также установить, каким образом символы функций содержательно могут быть связаны между собой, используются сигнатуры.

Определение (сигнатура). Сигнатура Σ- это пара (S, F) множеств S и F, причем

S обозначает множество типов, т. е. имен для носителей;

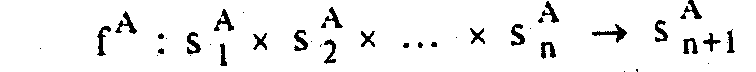
F обозначает множество символов (имен) функций;

для каждого символа функции f ∈ F пусть задана ее функциональное fct f ∈ S+.

В дальнейшем для улучшения читаемости при задании функциональности для f мы будем писать

fct f=(S1,..., Sn)Sn+l, .

чтобы выразить, что fА в вычислительной структуре А с соответствующей сигнатурой Σ используется для обозначения отображения



Пример (сигнатуры). Сигнатура вычислительной структуры BOOL булевс­ких значении и натуральных чисел из вышеприведенного примера дает пример сигнатуры.

smat = (bool,),

fbool = {true, false, -i, v, ^},

fct true = bool,

fct false = bool,

fct ~ = (bool) bool (бесскобочный префикс),

fct v = (bool, bool) bool (инфикс),

fct ^ = (bool, bool) bool (инфикс),

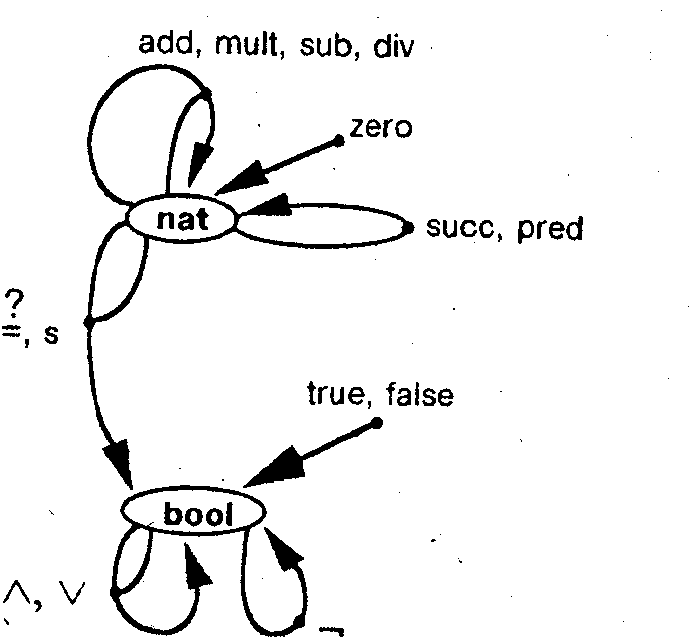


Рис.2.3. Диаграмма сигнатуры

Сигнатуры допускают наглядное графическое представление в виде диаграммы сигнатуры. Такая диаграмма для каждого типа содержит узел и для каждого n-местного символа операции - ребро с n входными узлами и одним выходным узлом. Для вычислительной структуры bool мы получаем диаграмму сигнатуры, показанную на рис. 2.3.

Задания одной только сигнатуры, конечно, недостаточно для того, чтобы однозначно охарактеризовать вычислительную структуру. Имеется много различных вычислительных структур с одной и той же сигнатурой.

Вычислительные структуры указывают на структурные свойств информационных систем. В частности, имеет место:

({sА : s ∈ S}, S, T)

образует снова информационную систему с Т: S → {sА : s ∈ S, причем T[s] = sА

Также ({fА: f ∈ F}, F, I) с I: F→ {fА: f ∈ F}, причем I[f] = fА являет информационной системой.

Для заданной вычислительной структуры с определенной сигнатуре всегда существует специальная структура, алгебра термов, состоящая множества термов, которые могут быть сформулированы над сигнатурой.

2.2.3. Основные термы

При заданной сигнатуре существует множество основных термов, кото­рые могут быть образованы с помощью символов функций сигнатуры. Пусть Σ = (S, F) есть сигнатура. Множество основных термов типа s с s ∈ S определяется следующим образом:

(i) каждый нульместиый символ функции f ∈ F с fct f = s образует основной терм типа s;

(ii) каждая последовательность символов f(t1, ..., tn) с f ∈ F и fct f= (S1, ..., Sn) s есть основной терм типа s, если для всех i, 1 <= i <= n, ti есть основной терм типа si.

Множество всех основных термов сигнатуры Σ обозначим через WΣ, a множество основных термов типа s - через WΣs. Если не существует нульместных символов функций, то множество WΣ пусто.

Если имеется вычислительная структура А с сигнатурой Σ, то основные термы в А допускают интерпретацию. Переход от основного терма (представления) типа s к соответствующему элементу а из множества А называют интерпретацией t в А. Интерпретация IA поэтому означает отображение

IA: WΣ → {а ∈ sA; s ∈ S}.

Для каждого основного терма t запись IA[t] обозначает интерпретацию I в А. Пишут также tA вместо IA[t]. Интерпретация получается заменой в основном терме символов функций на соответствующие функции:

IA[f(t1,...,tn)]=fA (IA[t1] ,.., IA[tn]).

В классической математике часто заданная интерпретация опускается и вместо tA просто записывается t; разницей между основным термом и его интерпретацией там сознательно пренебрегают.

Для каждой вычислительной структуры А сигнатуры Σ основные термы типа s ∈ S могут использоваться как представления элементов из множества sA, которые связаны с типом s в А. Если для каждого элемента а (<>⊥) носителей из А имеется представление терма, т. е. для каждого s и каждого а ∈ sA (<>⊥) существует основной терм типа s с tA = а, то А называется термопроизводимой, это равносильно требованию, что в соот­ветствующей информационной системе отображение интерпретаций является сюръективным.

Интерпретация ("значение") основного терма допускает соответствен­но вычисление терма. Один из простых способов организации такого вычисления представляют собой схемы.

2.2.4. Вычисления основных термов: схемы

Основные термы имеют характерную внутреннюю структуру. Основной терм образуется из символов функций и последовательности (иногда пустой) основных термов ("подтермов"), являющихся термами-аргументами.

Схема (или формуляр) для основного терма - это графическое представление вычислений при интерпретации этого терма; схема состоит из прямоугольников, в которые заносится интерпретация основных термов, и из подсхем для вычислений подтермов. Вычисление интерпретации основного терма допускает удобное его проведение по схеме. Поскольку интерпретация основного терма производится через значения интерпретаций его подтермов, эта интерпретация подтермов упорядочивается с помощью схемы, структура которой аналогична структуре самого основного терма.

Примеры (схемы).

(1) Основному терму ((1 + 2) \* 3) - 4 с интерпретацией в N соответствует схема, показанная на рис. 2.4.

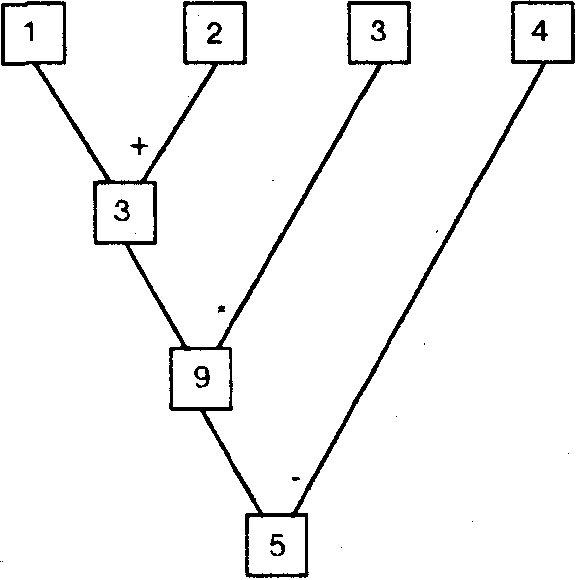


Рис. 2.4. Вычислительная схема

(2) Основному тepмy (true ^ false) v (false v ((~false ^ true)) с интерпретацией в BOOL соответствует схема, приведенная на рис. 2.5.

true

false

false

false

true

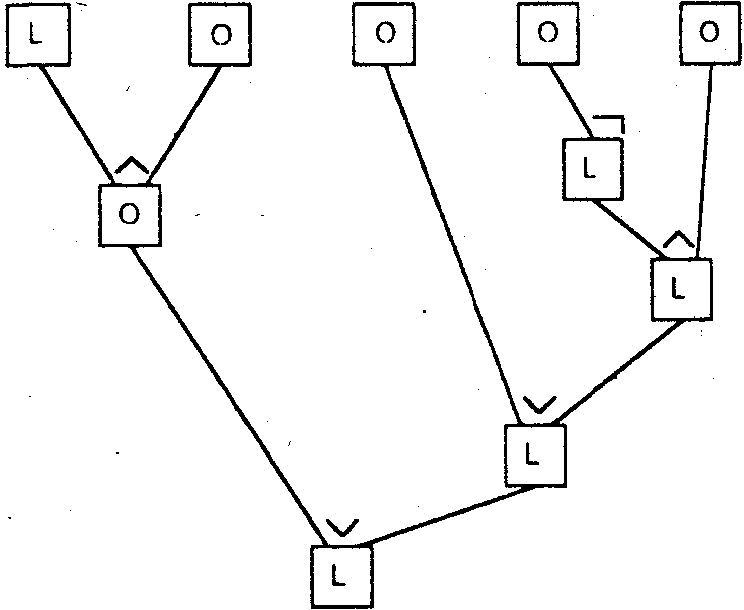


Рис. 2.5. Вычислительная схема

Как уже говорилось, основные термы могут очень легко применяться для представления элементов из множества носителей вычислительной структуры. Чтобы можно было определить отображение между этими элементами, используют термы со свободными идентификаторами.

2.2,5. Термы с (свободными) идентификаторами

Идентификатор ("обозначатель", "переменная", "неизвестное") - это держатель места ("имя") для терма (или элемента), который (позднее) может быть подставлен на это место. Идентификаторы могут пониматься как имена термов или элементов, которые будут конкретизированы только позднее.

Пусть Σ = (S, F) - сигнатура, и Х = {Xs: s ∈ S} - семейство множеств идентификаторов. Пусть множества Xs идентификаторов попарно не пересекаются и отличны от символов функций в F. WΣ(X) обозначает алгебру термов, распространенную на X, т. е. WΣI с ΣI = (S, F u {х ∈ Xs: s ∈ S}) и fct x = s для х ∈ Хs, где Xs обозначает множество идентификаторов (держателей мест, обозначений) типа s.

Примеры (термы с идентификаторами).

Уравнения с "неизвестными" в математике - это уравнения между термами с идентификаторами, например ax2 + bx + с = 0.

(2) Часто термы снабжаются свободными идентификаторами, чтобы определять функции. Функция f может быть определена через:

f: N → N с f(x) = 2х+ 1

Термы со свободными идентификаторами называются также полиномами. Вместо идентификаторов в термы могут подставляться другие термы. Соответствующее отображение называется подстановкой в термы с (сво­бодными) идентификаторами.

Пусть t - терм с идентификаторами, х - идентификатор типа s и г -терм типа s; через t [г/х] обозначается терм, который получается, когда идентификатор х заменяется на г. Этот процесс называется подстановкой.

Подстановки описываются формально, аналогично построению термов, с помощью следующих уравнении:

x [t/x] = t,

у [t/x] = у, если х и у - различные идентификаторы,

f(tl,...,tn)[t/x]=f(t1[t/x],...,tn[t/x]), где f ∈ F с fct f= (S1, .... Sn) Sn+1 и термы ti имеют типы si;.

Через t [t1|/X], ..., tn/Xn] обозначается терм, который возникает из терма t при одновременной подстановке ti вместо (попарно различных) иденти­фикаторов Xi.

Пусть t - терм с (свободными) идентификаторами. Терм г назовем экземпляром t, если г получается из t путем замены (подстановки) в нем определенных (свободных) идентификаторов.

2.2.6. Интерпретация термов с (свободными) идентификаторами

Пусть А - вычислительная структура с сигнатурой Σ = (S, F), а Х - семей­ство множеств идентификаторов. Отображение

b: {х ∈ Xs: s ∈ S} → {а ∈ sA : s ∈ S},

которое каждому идентификатору х в X типа s ставит в соответствие элемент а ∈ sA структуры данных sA типа s, называется конкретизацией Х (в А).

Для каждой конкретизации b определяется интерпретация IAb терма t со свободными идентификаторами из Х с помощью следующих равенств:

IAb[x]=b(x), IAb [f(t1,…,tn)]=fA(IAb [t1],…, IAb [tn]),

лтя n = 0 получается IAb [f] = fA .

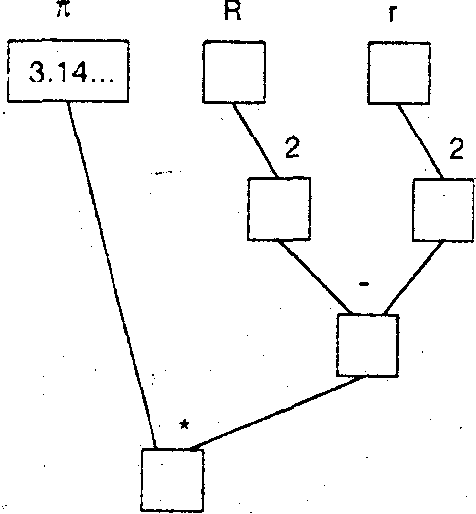
2.2.7. Термы с (свободными) идентификаторами как схемы

Термы со свободными идентификаторами могут играть роль схем, в кото­рых не все значения определены.

Пример (из геометрии). Площадь S кольца s с внутренним радиусом г и внешним радиусом R получается по формуле

S = π(R2 - r').

Терму в правой части формулы соответствует схема, приведенная на рис. 2.7.



Pw. 2.7. Вычислительная схема \_-:

Терм со свободными идентификаторами определяет вычислительную схему. Схемы можно найти (в несколько иной форме) во многих местах в управлении. Например, есть схемы для начисления зарплаты.

Если в терме определенные идентификаторы встречаются многократ­но, то вместо схем типа дерева получают "Наsse"-диаграммь1 (т. е. ацик­лические ориентированные графы).

Пример (схема с многократно применяемым промежуточным резуль­татом). Терм (х - у) \* (х + у) обладает схемой, показанной на рис. 2.8.

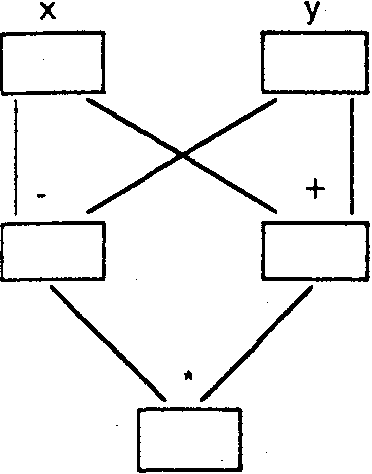


Рис.2.8. Вычислительная схема

Термы могут использоваться в системах замены термов для представле­ния алгоритмов.

2.3. Алгоритмы как системы подстановки термов

Наиболее наглядный метод описания алгоритмов как системы текстовых замен предлагают системы подстановки термов. Как было показано в предыдущем разделе, имеет место: термы могут строиться над заданной сигнатурой по определенным, четким правилам. К сигнатуре и термам над нею могут быть заданы интерпретации над вычислительной структу­рой, т. е. над алгеброй. Как и в случае булевских термов, возникает ин-4)ормационная система, при которой термы выступают как представле­ния. Через интерпретацию задается семантическая эквивалентность на гермах. Правила преобразования термов могут тогда быть определены так, что термы всегда будут переводиться в семантически эквивалентные термы.

Множества правил для алгоритмов могут быть предъявлены в форме системы подстановки термов. Специальная структура термов, которая получается из их построения, может использоваться для задания правил подстановок и их применения.

2.3.1. Правила подстановки термов

Для заданной сигнатуры Σ и заданного семейства Х множеств идентификаторов пара (t, r) термов t, r одинакового типа с (свободными) идентификаторами из Х называется подстановкой термов (правилом подстановки термов - ППТ) или также схемой подстановки термов. Правило записывается в виде t → г.

В большинстве случаев для ППТ требуется, чтобы все идентификато­ры, встречающиеся в г, также входили в t. Если в t и г заменяют опреде­ленные идентификаторы Х1, ..., Xn на термы t1, ..., tn подходящих типов, то получают экземпляр (частный, конкретный случаи) правила. Соот­ветственно

t [ t1/X1, ..., tn/Xn ] → r [ t1/X1, .... tn/Xn]

называется экземпляром правила t → г. Если t и г - основные термы, то экземпляр называется полным.

Пример (экземпляры правил подстановок термов). Для правила

pred (succ (x)) → х примерами экземпляров являются:

pred (succ (zero)) → zero, pred (succ (succ (у))) → succ (у).

Из правила получаются шаги подстановки термов благодаря тому, что правило применяется к любому подтерму имеющегося терма.

Пусть t —> г есть экземпляр правила. Пусть задан терм с, в котором встречается свободный идентификатор х; тогда

С [t/Х] → С [r/Х]

называется (безусловным) применением правила (к. терму c[t/x]). Tepм t называется редексом, а вхождение х в с - местом применения.

Пример (применение правила замены термов). К терму

succ (succ (pred (succ (zero)))) может быть применено правило предыдущего примера. Это дает:

succ (succ (pred (succ (zero)))) —> succ (succ (zero)).

Аналогом алгоритмов в виде подстановки текстов являются алгоритмы в виде систем подстановки термов (алгоритмы подстановки термов - АПТ).

2.3.2. Система подстановки термов

Множество (в общем случае конечное) R правил подстановки термов на сигнатуре Σ называется системой подстановки термов (СПТ) над Σ. Если для последовательности термов ti (0 <= i <= n) справедливо для i=0,...,n-l:

(\*). ti, —> ti+1 есть применение правила из системы подстановки термов R, то последовательность термов является вычислением в R для to.

Терм t называется терминальным для системы R, если не существует герма г такого, что t →г

есть применение правила из R.

Если в вычислении, заданном с помощью термов ti 0 <= i <= n, терм tn является терминальным, то вычисление называется терминированным (завершающимся), a tn -результатом или выходом вычисления для входа to.

Бесконечная последовательность (ti)i∈N термов, удовлетворяющих приведенному выше условию (\*), называется нетерминированным (незавершающимся) вычислением R для to. Система R называется в общем случае терминированной, если не существует незавершающихся вычислений.

Терминальные основные термы определяют нормальную форму. Они часто также называются термами в нормальной форме относительно R. Над системой R мы можем терму t поставить в соответствие терминальный терм г в качестве нормальной формы, если г есть результат вычислений с входом t. Как правило, система нормальных форм, индуцированных через СПТ, не является ни однозначной, ни полной. Термы, для которых существуют только бесконечные вычисления, не имеют нормальных форм. Для определенных термов могут существовать вычисления с различными результатами.

Пример (СПТ для вычислительной структуры BOOL). Ниже символы

функций ~, v, ^ будут использоваться в скобочной префиксной и инфиксной формах записи. Правила подстановок термов гласят:

(~true) → false, (~false) → true, (false v x) → x, (x v false) → x, (true v true) → true Эта СПТ редуцирует каждый основной терм типа bool к терму true или false.

Снова повторное применение ППТ из заданного множества правил мы можсм трактовать как алгоритм, который работает с термами в качестве входа и выхода.

2.3.3. Алгоритмы подстановки термов

Пусть задана система подстановок R; R определяет алгоритм в силу следующего предписания (алгоритм получает в качестве входного слова основной терм t):

(1) Если R содержит правило подстановки с применением t → г, то далее алгоритм продолжается с использованием г вместо t.

(2) Если R не содержит правила подстановки с применением t → г, то алгоритм закашивается с г в качестве результата.

Алгоритм подстановки термов (АПТ) тем самым выполняет вычисление в R для каждого основного терма t. Часто вычисление состоит в решении задачи преобразовать заданный основной терм в определенную заранее заданную нормальную форму.

Если АПТ начинает работать с заданным основным термом t, то t называют также входом для алгоритма; если алгоритм завершается основным термом г, то г называется также выходом или результатом.

Алгоритмы подстановки термов работают над основными термами сита-туры. Они осуществляют преобразование основных термов посредством вычислений. При этом не рассматривается, насколько эти преобразова­ния соответствуют определенным численным свойствам символов функ­ций. Все же благодаря концепции вычислительных структур возможна особенно ясная, простая концепция корректности СПТ.

ОСНОВЫ МАШИННОЙ АРИФМЕТИКИ

Системы счисления и способы перевода чисел из одной системы в другую.

Системой счисления называют систему приемов и правил, позволяющих устанавливать взаимно-однозначное соответствие между любым числом и его представлением в виде совокупности конечного числа символов. Множество символов, используемых для такого представления, называют цифрами.

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

В непозиционных системах любое число определяется как некоторая функция от численных значений совокупности цифр, представляющих это число. Цифры в непозиционных системах счисления соответствуют некоторым фиксированным числам. Пример непозиционной системы - римская система счисления. В вычислительной технике непозиционные системы не применяются.

Систему счисления называют позиционной, если одна и та же цифра может принимать различные численные значения в зависимости от номера разряда этой цифры в совокупности цифр, представляющих заданное число. Пример такой системы - арабская десятичная система счисления.

В позиционной системе счисления любое число записывается в виде последовательности цифр:



Позиции, пронумерованные индексами k (-l < k < m-1) называются разрядами числа. Сумма m+l соответствует количеству разрядов числа (m - число разрядов целой части числа, l - дробной части).

Каждая цифра в записываемой последовательности может принимать одно из N возможных значений. Количество различных цифр (N), используемых для изображения чисел в позиционной системе счисления, называется основанием системы счисления. Основание N указывает, во сколько раз единица k+1-го разряда больше единицы k-го разряда, а цифра соответствует количеству единиц k–го разряда, содержащихся в числе.



Таким образом, число может быть представлено в виде суммы:



Основание позиционной системы счисления определяет ее название. В вычислительной технике применяются двоичная, восьмеричная, десятичная и шестнадцатеричная системы. В дальнейшем, чтобы явно указать используемую систему счисления, будем заключать число в скобки и в индексе указывать основание системы счисления.

В двоичной системе счисления используются только две цифры: 0 и 1.Любое двоичное число может быть представлено в следующей форме:

(1)



Например, двоичное число

В восьмеричной системе счисления для записи чисел используется восемь цифр (0, 1, 2, 3, 4, 5, 6, 7), а в шестнадцатеричной - шестнадцать (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).



Таблица для перевода чисел из одной системы счисления в другую.

|  |  |  |  |
| --- | --- | --- | --- |
| Двоичные числа | Восьмеричные числа | Десятичные числа | Шестнадцатиричные числа |
| 0,0001 | 0,04 | 0,0625 | 0,1 |
| 0,001 | 0,1 | 0,125 | 0,2 |
| 0,01 | 0,2 | 0,25 | 0,4 |
| 0,1 | 0,4 | 0,5 | 0,8 |
| 1 | 1 | 1 | 1 |
| 10 | 2 | 2 | 2 |
| 11 | 3 | 3 | 3 |
| 100 | 4 | 4 | 4 |
| 101 | 5 | 5 | 5 |
| 110 | 6 | 6 | 6 |
| 111 | 7 | 7 | 7 |
| 1000 | 10 | 8 | 8 |
| 1001 | 11 | 9 | 9 |
| 1010 | 12 | 10 | A |
| 1011 | 13 | 11 | B |
| 1100 | 14 | 12 | C |
| 1101 | 15 | 13 | D |
| 1110 | 16 | 14 | E |
| 1111 | 17 | 15 | F |
| 10000 | 20 | 16 | 10 |

Для хранения и обработки данных в ЭВМ используется двоичная система, так как она требует наименьшего количества аппаратуры по сравнению с другими системами. Все остальные системы счисления применяются только для удобства пользователей.

В двоичной системе очень просто выполняются арифметические и логические операции над числами.

Таблица сложения:

|  |  |  |
| --- | --- | --- |
| Слагаемое | Слагаемое | Сумма |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 10 |

Таблица умножения:

|  |  |  |
| --- | --- | --- |
| Множитель | Множитель | Произведение |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Многоразрядные числа складываются, вычитаются, умножаются и делятся по тем же правилам, что и в десятичной системе счисления.

Перевод числа из одной системы в другую выполняется по универсальному алгоритму, заключающемуся в последовательном делении целой части числа и образующихся целых частных на основание новой системы счисления, записанное в исходной системе счисления, и в последующем умножении дробной части и дробных частей получающихся произведений на то же основание, записанное в исходной системе счисления.

При переводе целой части получающиеся в процессе последовательного деления остатки представляют цифры целой части числа в новой системе счисления, записанные цифрами исходной системы счисления. Последний остаток является старшей цифрой переведенного числа.

При переводе дробной части числа целые части чисел, получающихся при умножении, не участвуют в последующих умножениях. Они представляют собой цифры дробной части исходного числа в новой системе счисления, изображенные числами старой системы. Значение первой целой части является первой цифрой после запятой переведенного числа.

Пример перевода числа 30,6 из десятичной системы в двоичную:

Перевод целой части Перевод дробной части

Последовательное Остатки Целые части - Последовательное

деление разряды пере- умножение

веденной дроби

0, 6

X

2

30 / 2 0 ──────┐ ───────────────────

15 / 2 1 ─────┐│ ┌──── 1, 2

7 / 2 1 ────┐││ │ X

3 / 2 1 ───┐│││ │ 2

1 / 2 1 ──┐││││ │ ───────────────────

0 │││││ │┌─── 0, 4

│││││ ││ X

│││││ ││ 2

│││││ ││ ───────────────────

│││││ ││┌── 0, 8

│││││ │││ X

│││││ │││ 2

│││││ │││ ───────────────────

│││││ │││┌─ 1, 6

│││││ ││││

Результат: 11110,1001

Если при переводе дробной части получается периодическая дробь, то производят округление, руководствуясь заданной точностью вычислений.

Пример перевода числа 111110,01 из двоичной системы в десятичную.

Перевод целой части Перевод дробной части

0, 0100

X

1010

111110 / 1010 ───────────────────

1010 110 ────────┐ ┌───── 10, 1000

1011 │ │ X

1010 │ │ 1010

10 ────────────┼┐ │ ───────────────────

││ │┌─── 101, 0000

││ ││

Результат: 62,25

Примечание 1 1010 - основание десятичной системы счисления в двоичной записи.

Примечание 2: десятичные эквиваленты разрядов искомого числа находим по таблице.

При переводе чисел из любой системы счисления в десятичную удобнее пользоваться непосредственно формулой (1):



Для осуществления автоматического перевода десятичных чисел в двоичную систему счисления необходимо вначале каким-то образом ввести их в машину, Для этой цели обычно используется двоично-десятичная запись чисел или представление этих чисел в кодах ASCII.

При двоично-десятичной записи каждая цифра десятичного числа заменяется четырехзначным двоичным числом (тетрадой):



При записи чисел в кодах ASCII цифрам от 0 до 9 поставлены в соответствие восьмиразрядные двоичные коды от 00110000 до 00111001.

ЭВМ, предназначенные для обработки экономической информации, например IBM AT, позволяют производить арифметические операции в десятичной системе счисления над числами, представленными в двоично-десятичных кодах и кодах ASCII.

Шестнадцатеричная и восьмеричная системы счисления используются только программистами и операторами ЭВМ, так как представление чисел в этих системах более компактное, чем в двоичной, и перевод из этих систем в двоичную и обратно выполняется очень просто (основания этих систем представляют собой целую степень числа

Для перевода восьмеричного числа в двоичное достаточно каждый восьмеричный разряд представить тремя двоичными (триадой), а для перевода шестнадцатиричного числа - четырьмя (тетрадой):



Формы представления чисел в ЭВМ.

Разряд двоичного числа представляется в ЭВМ некоторым техническим устройством, например, триггером, двум различным состояниям которого приписываются значения 0 и 1. Группа таких устройств, предназначенная для представления в машине многоразрядного числа, называется регистром.

Структура двоичного регистра, представляющего в машине n-разрядное слово:

┌───┬───┬───┬───┬───┐

│n-1│n-2│...│ 1 │ 0 │

└───┴───┴───┴───┴───┘

Отдельные запоминающие элементы пронумерованы от 0 до n-1. Количество разрядов регистра определяет точность представления чисел. Путем соответствующего увеличения числа разрядов регистра может быть получена любая точность вычислений, однако это сопряжено с увеличением количества аппаратуры (в лучшем случае зависимость линейная, в худшем - квадратичная).

В ЭВМ применяются две основные формы представления чисел:

полулогарифмическая с плавающей запятой и естественная с фиксированным положением запятой.

При представлении чисел с фиксированной запятой положение запятой закрепляется в определенном месте относительно разрядов числа и сохраняется неизменным для всех чисел, изображаемых в данной разрядной сетке. Обычно запятая фиксируется перед старшим разрядом или после младшего. В первом случае в разрядной сетке могут быть представлены только числа, которые по модулю меньше 1, во втором - только целые числа.

Для кодирования знака числа используется старший ("знаковый") разряд.

При выполнении арифметических действий над правильными дробями могут получаться двоичные числа, по абсолютной величине больше или равные единице, что называется переполнением разрядной сетки. Для исключения возможности переполнения приходится масштабировать величины, участвующие в вычислениях.

Диапазон представления правильных двоичных дробей:

.



Числа, которые по абсолютной величине меньше единицы младшего разряда разрядной сетки, называются машинным нулем.

Диапазон представления целых двоичных чисел со знаком в n-разрядной сетке:



Использование представления чисел с фиксированной запятой позволяет упростить схемы машины, повысить ее быстродействие, но представляет определенные трудности при программировании. В настоящее время представление чисел с фиксированной запятой используется как основное только в микроконтроллерах.

В универсальных ЭВМ основным является представление чисел с плавающей запятой. Представление числа с плавающей запятой в общем случае имеет вид:



где

N - основание системы счисления,

p - целое число, называемое порядком числа A,

m - мантисса числа A (│m│<1).

Так как в ЭВМ применяется двоичная система счисления, то



причем порядок и мантисса представлены в двоичной форме.

Двоичное число называется нормализованным, если его мантисса удовлетворяет неравенству



Неравенство показывает, что двоичное число является нормализованным, если в старшем разряде мантиссы стоит единица. Например, число 0,110100\*10100 - нормализованное, а 0,001101\*10110 - ненормализованное.Нормализованное представление чисел позволяет сохранить в разрядной сетке большее количество значащих цифр и, следовательно, повышает точность вычислений. Однако современные ЭВМ позволяют, при необходимости, выполнять операции также и над ненормализованными числами.

Широкий диапазон представления чисел с плавающей запятой удобен для научных и инженерных расчетов. Для повышения точности вычислений во многих ЭВМ предусмотрена возможность использования формата двойной длины, однако при этом происходит увеличение затрат памяти на хранение данных и замедляются вычисления.

Представление отрицательных чисел в ЭВМ.

Для кодирования знака двоичного числа используется старший ("знаковый") разряд (ноль соответствует плюсу, единица - минусу).

Такая форма представления числа называется прямым кодом.

Формула для образования прямого кода правильной дроби имеет вид:

[A]пр=A, если A>=0,

[A]пр=1-A, если A<0.

Примеры:

A = 0,110111 🡪 [A]пр = 0,110111

A = -0,110111 🡪 [A]пр = 1 - (-0,110111) = 1,110111

Прямой код целого числа получается по формуле:

[A]пр=A, если A>=0,

[A]пр=10n-1-A, если A<0.

где 10 - число 2 в двоичной системе счисления, n - количество позиций в разрядной сетке.

Например, при n=8

A = 110111 🡪 [A]пр = 00110111A = -110111 🡪 [A]пр = 10000000 - (-110111) = 10110111

В ЭВМ прямой код применяется только для представления положительных двоичных чисел. Для представления отрицательных чисел. применяется либо дополнительный, либо обратный код, так как над отрицательными числами в прямом коде неудобно выполнять арифметические операции.

Формула для образования дополнительного кода дроби:

[A]доп = 10 + А

Формула для образования обратного кода дроби:

[A]обр = 10 - 10-(n-1) + A.

Например, при n=8, A=-0,1100001[A]доп = 10 + (-0,1100001) = 1,0011111

[A]обр = 10-10-7+(-0,1100001) = 1,1111111-0,1100001 = 1,0011110.

Формула для образования дополнительного кода целого числа:

[A]доп = 10n + A.

Формула для образования обратного кода целого числа:

[A]обр = 10n - 1 + A.

Например, при n = 8, A = -1100001 🡪

[A]доп = 100000000 + (-1100001) = 10011111

[A]обр = 100000000-1+(-1100001) = 11111111-1100001 = 10011110.

Таким образом, правила для образования дополнительного и обратного кода состоят в следующем:

для образования дополнительного кода отрицательного числа необходимо в знаковом разряде поставить единицу, а все цифровые разряды инвертировать (заменить 1 на 0, а 0 - на 1), после чего прибавить 1 к младшему разряду;

для образования обратного кода отрицательного числа необходимо в знаковом разряде поставить единицу, а все цифровые разряды инвертировать.

Примечание: при данных преобразованиях нужно учитывать размер разрядной сетки.

Замена вычитания двоичных чисел A1-A2 сложением с дополнениями [A1]пр+[-A2]доп или [A1]пр+[-A2]обр позволяет оперировать со знаковыми разрядами так же, как и с цифровыми. При этом перенос из старшего знакового разряда, если он возникает, учитывается по разному для обратного и дополнительного кодов:

при использовании дополнительного кода единица переноса из знакового разряда отбрасывается;

при использовании обратного кода единица переноса из знакового разряда прибавляется к младшему разряду суммы (осуществляется так называемый циклический перенос).

Пример: складываем числа A1=0,10010001 и A2=-0,01100110

При использовании обратного кода получим:

[A1]пр = 0,10010001

+

[A2]обр = 1,10011001

───────────

10,00101010

└─────── +1

Результат: 0,00101011

При использовании дополнительного кода получим:

[A1]пр = 0,10010001

+

[A2]доп = 1,10011010

───────────

Результат: 0,00101011

Если знаковый разряд результата равен нулю, то получено положительное число, которое представлено в прямом коде. Если в знаковом разряде единица, то результат отрицательный и представлен в обратном или дополнительном коде.

Для того, чтобы избежать ошибок при выполнении бинарных операций, перед переводом чисел в обратные и дополнительные коды необходимо выравнивать количество разрядов прямого кода операндов.

При сложении чисел, меньших единицы, в машине могут быть получены числа, по абсолютной величине большие единицы. Для обнаружения переполнения разрядной сетки в ЭВМ применяются модифицированные прямой, обратный и дополнительный коды. В этих кодах знак кодируется двумя разрядами, причем знаку "плюс" соответствует комбинация 00, а знаку "минус" - комбинация 11.

Правила сложения для модифицированных кодов те же, что и для обычных. Единица переноса из старшего знакового разряда в модифицированном дополнительном коде отбрасывается, а в модифицированном обратном коде передается в младший цифровой разряд.

Признаком переполнения служит появление в знаковом разряде суммы комбинации 01 при сложении положительных чисел (положительное переполнение) или 10 при сложении отрицательных чисел (отрицательное переполнение). Старший знаковый разряд в этих случаях содержит истинное значение знака суммы, а младший является старшей значащей цифрой числа. Для коррекции переполнения число нужно сдвинуть в разрядной сетке на один разряд вправо, а в освободившийся старший знаковый разряд поместить цифру, равную новому значению младшего знакового разряда. После корректировки переполнения мантиссы результата необходимо увеличить на единицу порядок результата.

Система вещественных чисел, применяемая при ручных вычислениях, предполагается бесконечной и непрерывной, т.е. не существует никаких ограничений на диапазон используемых чисел и точность их представления.

Однако в компьютерах реализация такой системы на аппаратном уровне была бы нецелесообразной, хотя программно может быть реализована любая точность вычислений. Нецелесообразность аппаратной реализации вычислений с произвольной точностью вызвана тем, что такие вычисления требуют неоправданно большого расхода основных машинных ресурсов: памяти и процессорного времени.

Во всех компьютерах размеры регистров и ячеек памяти фиксированы, что ограничивает систему представления чисел. Ограничения касаются как диапазона, так и точности представления чисел, т.е. система машинных чисел оказывается конечной и дискретной.

В любой универсальной ЭВМ существует несколько различных форматов представления как для чисел с фиксированной, так и для чисел с плавающей запятой. На некоторые из форматов имеются международные стандарты, и поэтому такие форматы являются общими для ЭВМ, построенных различными фирмами на различной элементной базе. Следует отметить, что нестандартные форматы обычно являются неявно специализированными для определенных областей применения, причем разработчики аппаратуры могут не указать в документации, для чего был предназначен тот или иной формат.

С точки зрения программиста важно, какие из форматов данных обрабатываются аппаратными средствами данной ЭВМ, а какие - только программными средствами. Операции над данными любого формата, который не поддерживается аппаратурой, выполняются очень медленно. Любой формат данных, который превышает размер регистров процессора, не пригоден для быстрых вычислений.

Для представления целых чисел в ЭВМ обычно применяются 8-, 16-, 32- и 64-битовый стандартные форматы, причем интерпретация чисел как знаковых или беззнаковых обычно возлагается на программиста или на компилятор с языка высокого уровня.

Для представления чисел с плавающей запятой также существует несколько стандартных форматов, различающихся по точности, но имеющих одинаковую структуру следующего вида:

n-1 n-2 0

┌───╥───┬───┬─────┬───╥───┬───┬─────┬───┐

│ ║ │ │ ... │ ║ │ │ ... │ │

└───╨───┴───┴─────┴───╨───┴───┴─────┴───┘

│ └────────╥────────┘└────────╥───────┘

│ смещенный модуль

знак порядок мантиссы

мантиссы

Порядок p задается в так называемой смещенной форме: если для задания порядка выделено k разрядов, то к истинному значению порядка прибавляют смещение, равное (2k-1-1). Использование смещенной формы позволяет производить операции над порядками, как над беззнаковыми числами, что упрощает операции сравнения, сложения и вычитания порядков. Кроме того, использование смещенного порядка упрощает операцию сравнения нормализованных чисел с плавающей запятой, сводя ее к операции сравнения целых чисел.

Следует отметить, что вещественный формат с m-разрядной мантиссой позволяет абсолютно точно представлять m-разрядные целые числа, т.е. любое двоичное целое число, содержащее не более m разрядов, может быть без искажений преобразовано в вещественный формат.

ОСНОВНЫЕ КОМПОЗИЦИИ ДЕЙСТВИЙ И ИХ ПРАВИЛА ВЫВОДА

1. СООТНОШЕНИЯ ДЛЯ КОРРЕКТНОСТИ ПРОГРАММ

Каждый алгоритм имеет как статическую, так и динамическую струк­туры. Статическая структура представляется текстом (или структурной схемой) программы, описывающим действия и проверки, которые должны быть выполнены при решении данной задачи. Текст не зависит от значений исходных данных. Напротив, динамическая структура в значительной степени определяется выбором исходных дан­ных, поскольку при выполнении алгоритма будут сделаны различные пе­реходы в зависимости от значений входных переменных. Динамическая структура отражает процесс вычислений и состоит из последовательности состояний вычислений.

Состояние вычислений в любой момент времени включает в себя значения всех программных переменных в этот момент и, таким образом, зависит от начальных значений переменных и этапов вычислений алгоритма, которые ему предшествовали. Текущая инструкция или изменяет состояние вычисле-

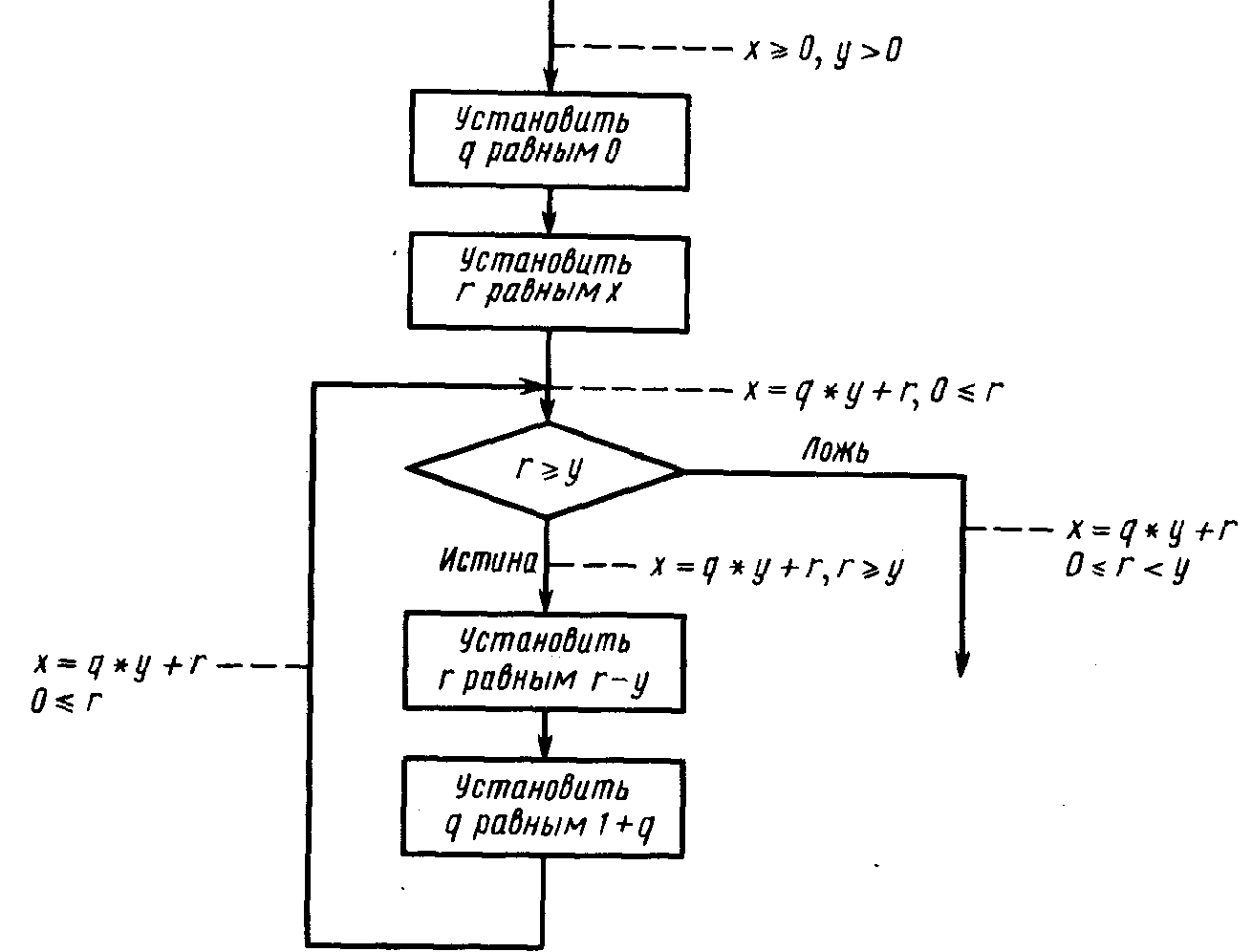


Рис. 1. Алгоритм для нахождения q и r-, удовлетворяющих условиям x=q\*y+r и 0 <= г < y. Таким образом, на выходе q = х div y и r=х mod y.

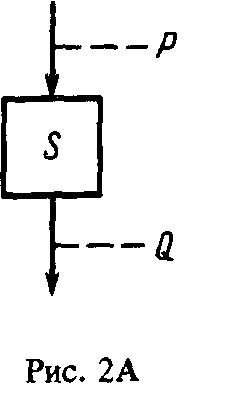
ний (значения некоторых переменных) и передает управление следующей инструкции в последовательности, или производит проверку состояния вы­числений (сравнивает значения определенных переменных) и на основе ре­зультата этой проверки передает управление другой инструкции.

Язык программирования предоставляет как простые операторы, так и методы композиции, которые позволяют формировать структурные опера­торы из других простых или составных операторов. Перед нами стоят две связанные между собой задачи.

Определить виды используемых в языке Паскаль простых операторов, а также часто применяемые методы композиции решений подзадач.

2. Обеспечить правила вывода, позволяющие определить эффект воздей­ствия простого оператора на состояние вычисления, а также вывести опре­деленные свойства составного оператора из свойств составляющих его ком­понент.

Рассмотрим рис. 1, который воспроиз­водит алгоритм для определения х div у и х mod у. Заметим, что состояние вычислений, связанное с точкой входа, определяется значениями входных переменных х и у; но после выполнения первых двух операторов состояние вычислений расширяется, включая зна­чения четырех переменных: х, у, q и г. (Значения х и у могут быть различными



в зависимости от применения алгоритма, но остаются постоянными в течение одного применения). Ясно, что сле­дующая за проверкой r >= у передача управления будет за­висеть от текущего состояния вычислений.

Фундаментальное свойство всех видов композиции, которые мы будем рассматривать, заключается в том, что они дают возможность объединить в одну сложную структур­ную схему с одним входом и одним выходом одну или более схем также с одним входом и одним выходом. Как показано на рис. 1, наша процедура проектирования объединяет с этими точками некоторые соотношения, которые описывают существенные аспек­ты состояний вычислений в этих точках. Каждая такая структурная схема имеет вид, приведенный на рис. 2А, где S может быть отдельным действи­ем ЭВМ или сложной схемой. Чтобы выразить ее в более краткой форме, используем нотацию:

{Р} S {Q}. (1)

Это - спецификация программы со следующим смыслом: если соотношение Р истинно перед выполнением S, то Q будет истинно после завершения выполнения S. Другими словами, если Р истинно при входе, то Q будет истинно при выходе.

Если S — программа, чью корректность мы хотим установить, то нотация (1) в том смысле, как мы ее определили, — это то, что мы хотим доказать, причем Р — соотношение, которому должны удовлетворять начальные зна­чения переменных, а Q — соотношение, которому должны удовлетворять конечные значения переменных. Например, если S - алгоритм, изображен­ный на рис. 1, то соотношение, которое мы хотим доказать, таково:

{(х >= 0)^(y > 0)} S {(х = q \*у + r)^(0<= r < у)}, (2)

где знак ^ используется в качестве формальной нотации для "и".

Как уже отмечалось, соотношение {Р} S {Q} означает, что если P истинно перед выполнением S, то Q должно быть истинно при завершении S. Это оз­начает, что {Р} S {Q} тождественно истинно, если S не завершается, т. е. если на рис. 2А точка выхода никогда не достигается. Другими словами, (1) определяет только частичную корректность S. Частично корректной является такая программа, в которой гарантируется получение требуемого резуль­тата при условии ее завершения. Но завершается ли программа действитель­но для некоторых исходных значений — это другой вопрос. Если дополни­тельно можно показать, что программа завершается для всех исходных значений, удовлетворяющих соотношению Р, то говорим, что программа полностью корректна. Чтобы доказать завершение программы, получаемой применением правил композиции, введенных ранее, необходим анализ циклов, т.е. операторов итерации.

Можно подвести итог нашему подходу к проектированию программ.

1. Проектирование должно начинаться со спецификации {Р} S {Q}, кото­рой должна удовлетворять проектируемая нами программа. Мы начинаем, таким образом, .с ясного и недвусмысленного определения того, когда программа должна использоваться (предусловие Р), и результата ее исполь­зования при этом (постусловие Q).

2. Процесс проектирования сверху вниз определяет спецификации [Рi} Si {Qi} для компонентов Si, из которых строится программа.

3. Проектирование программы осуществляется одновременно с доказа­тельством корректности указанных спецификаций.

2.3. ПРАВИЛА ВЫВОДА ДЛЯ ПРОСТЫХ ОПЕРАТОРОВ

Правила вывода - это схемы рассуждений, позволяющие доказывать свойства программ. Правила вывода имеют вид

Н1,…,Hn (,2.26)

H

Если H1, . . . , Hn, - истинные утверждения, то Н - также истинное ут­верждение.

Два простейших правила вывода формулируются в (2.27) и (2.30). Первое утверждает, что если выполнение программы S обеспечивает ис­тинность утверждения R, то оно также обеспечивает истинность каждого утверждения, которое следует из R, т. е.

[P}S{R}, R → Q

{P}S{Q}

Например, из

{(х > 0)^(у > 0)} S{(z+u \*у = х \*у)^(и = 0)}, (2.28)

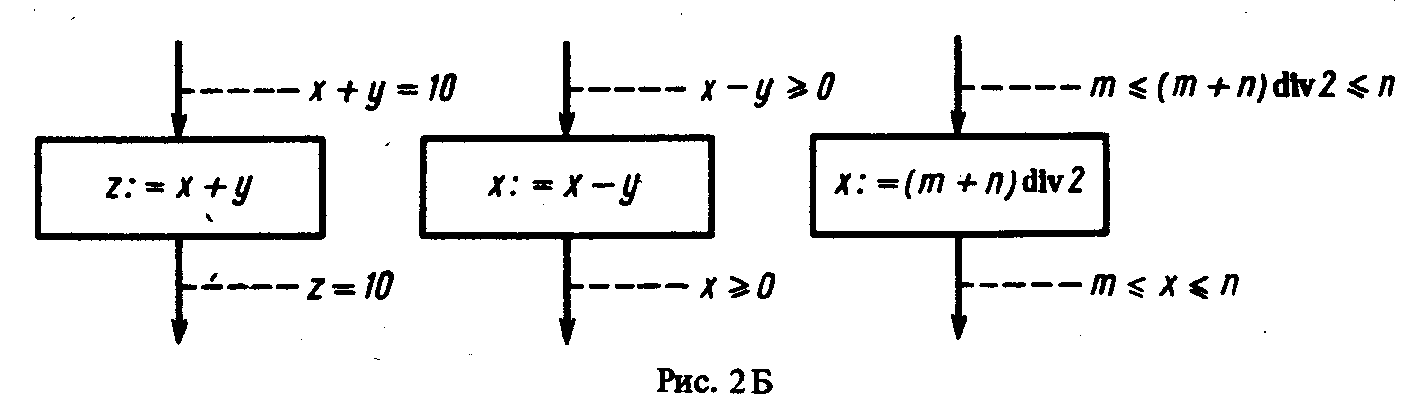
(z+u\*y=x\*y)^(u=0)→(z=x\*y) (2.29)

заключаем, используя (2.27), что {(х > 0) ^ (у > Q)} S {z = х \* у].

Второе правило утверждает, что если R - известное предусловие програм­мы S, приводящей к результату Q после завершения своего выполнения, то это же относится к любому другому утверждению, из которого следует R:

P→R, [R}S{Q}

{P}S{Q}



Правила (2.27) и (2.30) называются правилами консеквенции.

Теперь обратимся к специфическому виду, который принимает {Р} S {Q}, если что-то известно о S. Будем изучать действие структурных операторов в последующих разделах. Здесь рассмотрим случай, когда S - простой опе­ратор. Простейшей формой простого оператора является пустой оператор — тот, который не оказывает никакого воздействия на значения программных переменных. Для любого Р имеем правило вывода

{Р}{Р}. (2.32)

Из простых операзйэров, оказывающих влияние на значения программных переменных и, следовательно, на Булевы значения утверждений Р и Q в [Р) S {Q}, ранее введено только присваивание. Итак, пусть х:=е есть оператор присваивания, который устанавливает х равным значению выра­жения е. Тогда можем сделать вывод, что для любого Р

{PXe}x:=e{P}. (2.33)

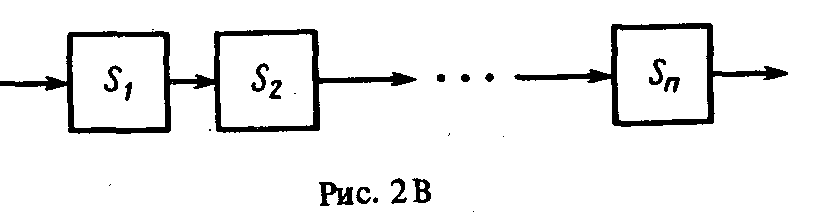
Здесь утверждается, что если Р истинно для подстановки е вместо х перед выполнением присваивания, то Р должно быть истинным, когда перемен­ной х присвоили ее новое значение. Это правило лучше пояснить приме­рами, данными на рис. 2Б.

2.4. СОСТАВНЫЕ И УСЛОВНЫЕ ОПЕРАТОРЫ

Предположим, мы хотим доказать, что имеет место {Р} S {Q}, когда S является структурным оператором. Нам необходимо правило для каждого типа композиции операторов, которое позволит вывести свойства сложного (структурного) оператора на основе установленных свойств его компонент. В этом разделе обсуждаются такие правила для составных и условных опера­торов.

Составные операторы.

Простейшей формой структурирования является создание так называ­емых составных операторов путем последовательной композиции, состо-



ящей из действия S1, за которым следует действие S2. Можно обобщить это определение последовательной композиции на случай произвольного конечного числа действий S1, S2, . . . Sn. В языке Паскаль последователь­но соединенные операторы обычно заключают в скобки begin и end, кото­рые указывают, что полученный таким образом оператор является единым, хотя и структурным. Такой оператор имеет вид begin S1;S2,.. . Sn end (2.34)

и называется составным оператором. Он может быть представлен струк­турной схемой (рис. 2В).

Правило вывода для последовательной композиции гласит, что если S есть begin S1; S2 end и если имеют место {Р} S1 {R} и {R}S2{Q}, то истинно и {Р} begin S1 ;S2 end {Q}. Формально это правило может быть выражено следующим образом:

{Р} S, [R], {R} S, {Q}

{Р}begin Si;S, end {Q} (2-36)

Правило вывода (2.36) обобщается следующим очевидным образом:

{Рi-1}S1{Pi} дпя i=1,....n

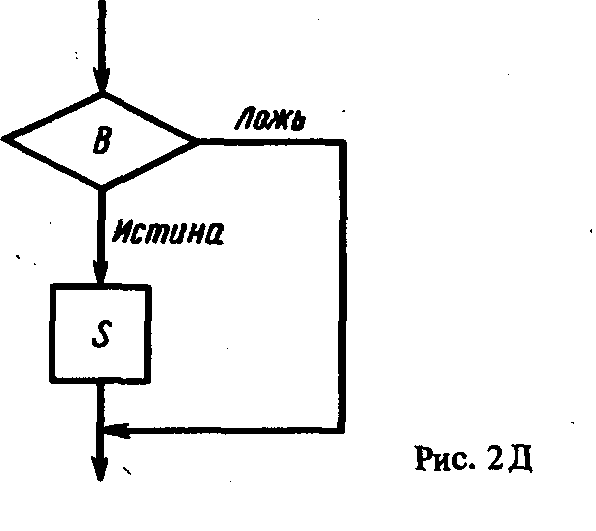
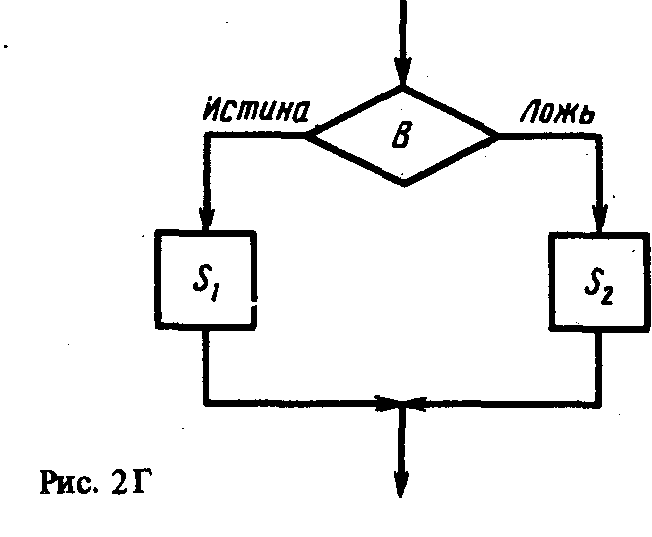
{P0} begin S1;. ..Sn. end {Pn}

Условные операторы.

Если S1 и S2 - операторы, а В — Булево выражение (т.е. выражение, имеющее значение Булевого типа), то

if В then S1 else S2 (2.38)

есть оператор, обозначающий следующее действие: вычисляется В; если его значение есть истина, то должно быть выполнено действие, описываемое S1, в противном случае — действие, описываемое S2 Выражение (2.38) может быть графически представлено структурной схемой (рис. 2Г).



Выработаем правило вывода для условного оператора (2.38). Если требуется установить истинность [Р] if В then S1 else S2 {Q},то необхо­димо доказать два утверждения.

1. Если В истинно, то выполняется S1. Так как Р справедливо перед вы­полнением (2.38), то делаем вывод, что в этом случае "Р^В также спра­ведливо перед выполнением S1. Если Q справедливо после выполнения (2.38), то должно быть справедливо и [Р ^ В} S1 {Q}. Итак, мы должны доказать {Р^В} S1 {Q}.

2. Если В ложно, то будет выполняться S2. Так как Р было истинно перед выполнением (2.38), делаем вывод, что Р ^ ~В справедливо перед выпол­нением S2. С этим утверждением в качестве предусловия S2 требуется до­казать, что после выполнения S2 будет справедливо Q, т.е. доказать [p^~b}s2{q].

Если мы доказали как [Р^В] S1 {Q}, так и{Р^ ~В} S2 {Q}, то можно утверждать, что если Р справедливо перед выполнением (2.38), то Q будет справедливо по окончании его выполнения независимо от того, какой опе­ратор (S1 или S2) был выбран для выполнения. Итак, можно сформулиро­вать следующее правило вывода:

{P^B}S1{Q},{P^~B}S2{Q}.

{Р} if В then S1 else S2 {Q}

Теперь рассмотрим два варианта условного оператора и обеспечим для каждого правила вывода. Первый вариант получается, когда мы замечаем, что в if B then S1 else S2 S2 может быть любым и, в частности, пустым опера­тором. Пустой оператор определяет тождественную операцию, т. е. операцию, не воздействующую на значения переменных. Если S2 — пустой оператор, то (2.38) запишется так:

if B then S. (2.41)

Действие, которое определяется этим оператором, вначале состоит в вычислении В. Если В истинно, то должно быть выполнено действие, опре­деляемое S, иначе должна быть выполнена тождественная операция. Выра­жение (2.41) можно представить рис. 2Д.

Теперь определим соответствующее правило вывода. Если мы хотим до­казать {P} if B then S{Q}, то необходимо обеспечить два условия. Первое из них заключается в том, что если S выполняется, то Q справедливо после его завершения. Так как при условии истинности В для выполнения выби­рается S, то выводим, что Р /\В справедливо перед выполнением S, если Р справедливо перед выполнением (2.41). Итак, требуется доказать {p^b}s{q}. Второй случай имеет место, когда S не выполняется, т.е. когда после вычисления В получается значение ложь. Итак, в этой точке справедливо Р ^ ~В, и если Q справедливо после выполнения (2.41), то необходимо доказать, что Р ^ ~В → Q. Альтернативой получения правила вывода является рассмотрение (2.41) в эквивалентной форме:

if B then S else. (2.42)

В (2.42) после else идет пустой оператор. Но в любом случае правило вывода формулируется следующим образом:

{P^B}S{Q},P^~B→Q .

{Р} if В them S {Q}

2.5. ОПЕРАТОРЫ ИТЕРАЦИИ

Теперь обратимся к структурам, которые приводят к циклам. Сначала изучим конструкцию while-do, а затем repeat-until.

Операторы while-do

Если В — Булево выражение и S — оператор, то

while В do S (2.47)

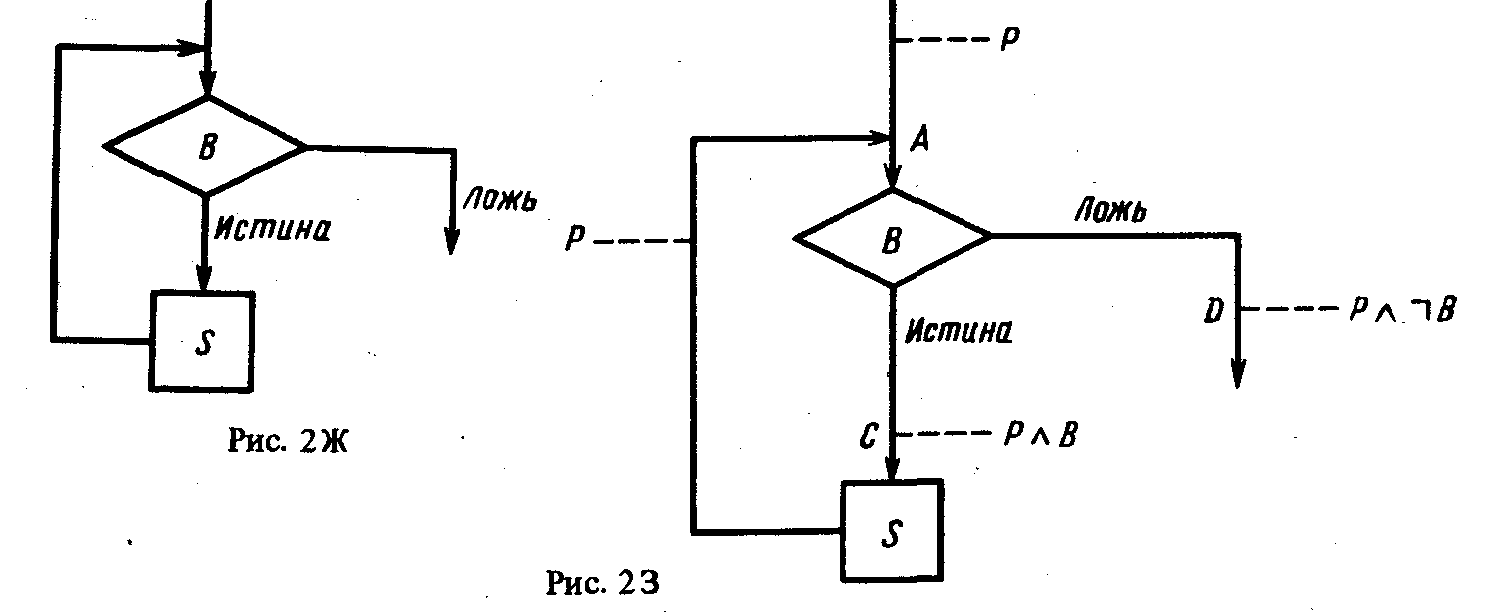
обозначает итерационное выполнение оператора S, пока В истинно. Если В ложно с самого начала, то S не будет выполняться совсем. Процесс итера­ционного выполнения S оканчивается, когда В становится ложным. Эта ситуация изображена на рис. 2Ж.

Задав любое начальное неотрицательное целое значение п, можно восполь­зоваться этим оператором для вычисления суммы I2 + 22 + ... +п2, полу­чаемой как конечное значение h в следующей программе:

begin h := 0;

while n > 0 do begin h: = h + n \*n; n: = п - 1 end

Теперь необходимо изучить утверждение {Р} while В do S {Q}, для чего рассмотрим рис. 2Ж более подробно, т. е. перейдем к рис. 23. Если Р спра­ведливо, когда мы впервые входим в рис. 23, то ясно, что Р^В будет справедливо, если мы достигнем точки С Тогда если мы хотим быть уве­-



ренными, что Р снова выполняется при возврате в точку А, нам необходимо обеспечить истинность для S утверждения {Р ^ В} S {Р}.

В этом случае ясно, что Р будет выполняться не только тогда, когда точ­ка А на рис. 23 достигается первый или второй раз, но и после произволь­ного числа итераций. Аналогично Р ^ В выполняется всякий раз, когда достигается точка С. Когда достигается точка выхода D (см. рис. 23), то справедливо не только Р, но и ~В. Итак, получаем следующее правило вы­вода:

{p^b}s{p}

{Р} while В do S {Р ^ ~В}

Заметим, что (2.49) устанавливает свойство инвариантности Р для цикла в (2.49). Если Р выполняется для начального состояния вычисления, то оно будет выполняться для состояния вычисления, соответствующего каждо­му прохождению цикла. Р составляет сущность динамических процессов, которые происходят при выполнении (2.47). Далее будем называть Р ин­вариантом цикла.

Операторы repeat-until.

Другая форма оператора итерации:

repeat S until В, (2.50)

где S — последовательность операторов и В — Булево выражение. Опера­тор (2.50) определяет, что S выполняется перед вычислением B. Затем, если В ложно, процесс итерационного выполнения S продолжается, в противном случае он завершается. Структура оператора итерации (2.50) представлена на рис. 2И.

Сравнивая рис. 2Ж и 2И, замечаем, что на рис. 2И S должен быть выпол­нен по меньшей мере один раз, в то время как на рис. 2Ж он может не вы­полняться совсем.

Пример оператора repeat-until дан в программе (2.51), которая вычис­ляет сумму h=12+22+...+n2 для заранее заданного значения п:

begin h:=0;

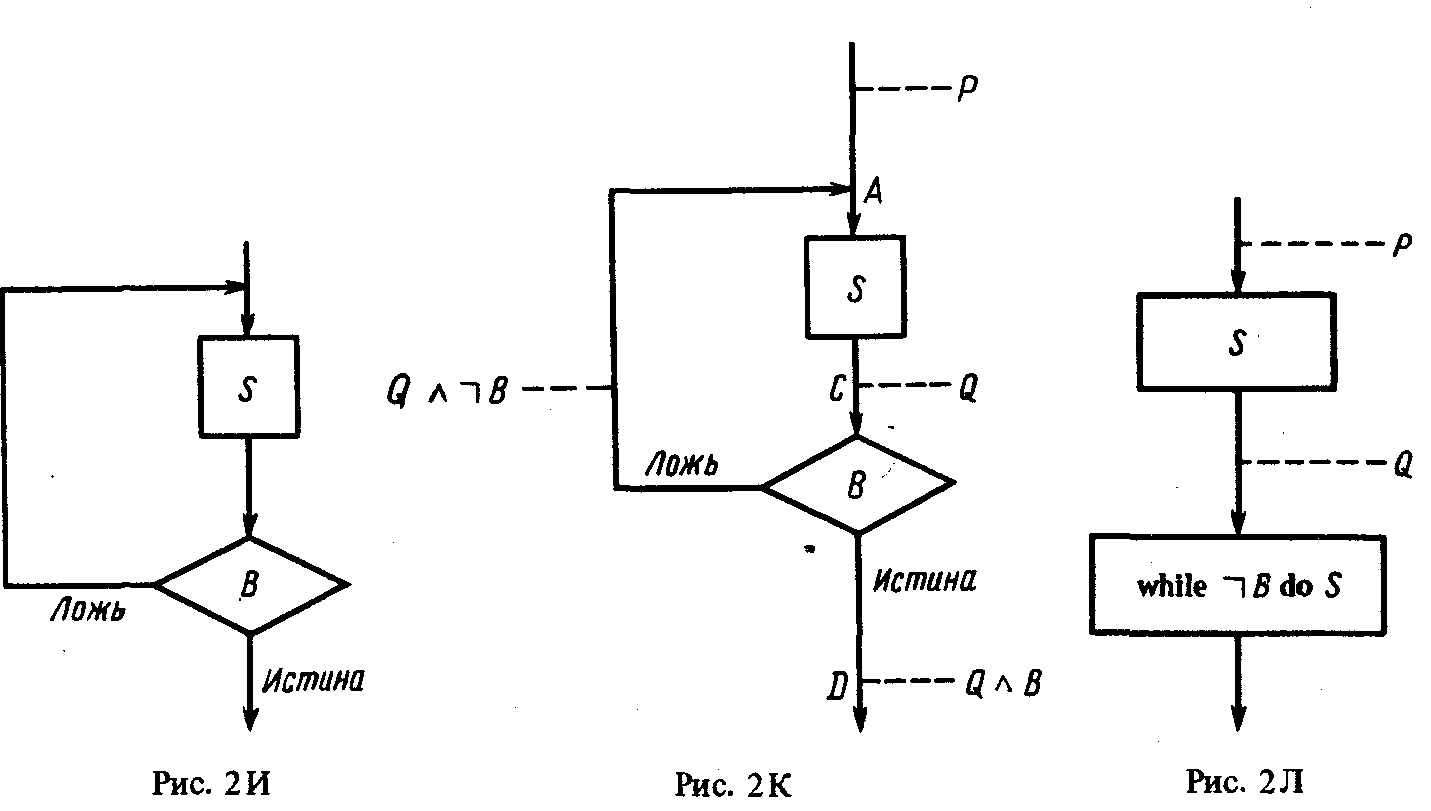
repeat h: = h + n \* п;

п := п — 1

until n= 0 (2.51) end

Предположим, что допускаются отрицательные значения п. Тогда можно использовать (2.48) и (2.51) для иллюстрации фундаментальной проблемы, связанной с итерационной композицией, когда мы заинтересованы в полной, а не в частичной корректности. Если n<=0, то (2.51) является бесконечным циклом. С другой стороны, программа (2.48) завершается, даже если n<=0. Таким образом, видим, что неправильно спроектированный оператор ите­рации может фактически описывать бесконечное вычисление.

Чтобы установить правило вывода для repeat S until В, рассмотрим рис. 2К. Предположим, доказано, что {Р} S {Q} и Q ^~B → Р. Тогда если Р истинно в точке входа, то Q будет истинно, когда точка С достигается в первый раз.



Если В ложно, то истинно Q ^ ~В, и при прохождении через цикл вновь до­стигается точка А. Так как Q/\~В →.Р, то, следовательно, Р удовлетво­рится при достижении точки А во второй раз (если это вообще происходит). Но тогда вновь на основании [Р] S {Q} будет удовлетворяться Q, когда точка С достигается во второй раз, и т.д. Итак, видно, что при сделанных предположениях Р будет истинно всякий раз, когда достигается точка А, и Q будет истинно всякий раз, когда достигается точка С, независимо от числа повторений цикла. Когда итерационный процесс заканчивается, т.е. когда достигается точка выхода D на рис. 2К, то должно быть истинно ут­верждение Q^B. Это рассуждение приводит к следующему правилу вывода:

{P}S{Q},Q^~B→P

{Р} repeat S until B{Q^B]

Покажем, как за­менить оператор repeat-until структурной схемой, в которой цикл задается оператором while-do. Вспомним идею repeat S until В — выполнить S, а затем проверить В, затем выполнить снова S, если В ложно, и повторять этот про­цесс. Другими словами, выполнить S, повторяя затем выполнение S до тех пор, пока В не станет истинным. Таким образом, имеем эквивалентность:

repeat Suntil В = begin S; while ~Bdo S end. (2.53)

Теперь проверим, что (2.53) позволяет вывести правило для repeat-until (2.52) из правила вывода while-do

{P^B} S {P} (2.49)

{P}while B do S{P^~B}

и правила вывода

{P}S1{R},{R}S2{Q]

{Р} begin S1; S2 end {Q}' (2.54)

Нам потребуется также правило консеквенции

Р → R, {R} S {Q}

{Р} S {Q}

Если задано {Р} S {Q}, то естественно берем Q в качестве соотношения в точке входа в while ~В do S (рис. 2Л), которое приводит к использова­нию (2.49) в модифицированном виде

{Q^~B}S[Q}

{Q} while ~В do S {Q ^ В}

где применяется равенство ~~B =В. Теперь, вместе с соответствующей заменой переменных, из правила консеквенции (2.55) вытекает:

Q^~B→P,{P}S[Q}

{Q^~B}S{Q]

комбинация которого с (2.56) дает

{P}S{Q},Q^~B→P

{Q} while ~В do S {Q ^ В}

Затем, используя (2.54), получаем искомый результат

[p}s{q},q^ ~b→p

{Р} begin S; while ~В do S end {Q ^ B}

который с учетом заданной эквивалентности (2.53) и есть то, что требова­лось доказать.

2.7. ИСПОЛЬЗОВАНИЕ ОСНОВНЫХ ПРАВИЛ ВЫВОДА

Алгоритм деления.

В качестве примера доказательства корректности программы применим полученные нами правила вывода для доказательства корректности програм­мы деления неотрицательных целых чисел (см. рис. 2.1). Пусть читатель теперь проверит, что структурная схема действительно отображена в следу­ющей программе на языке Паскаль. Заметим, что соотношения (коммен­тарии, заключенные в { }) не являются частью собственно программы, а описывают существенные аспекты состояния вычислений в соответствующих точках, определенных в процессе проектирования сверху вниз:

{(х>=0)^(y>0)} begin q:= 0;

r := х;

{(x=q\*y+r)^(0<=r)}

while r > у do

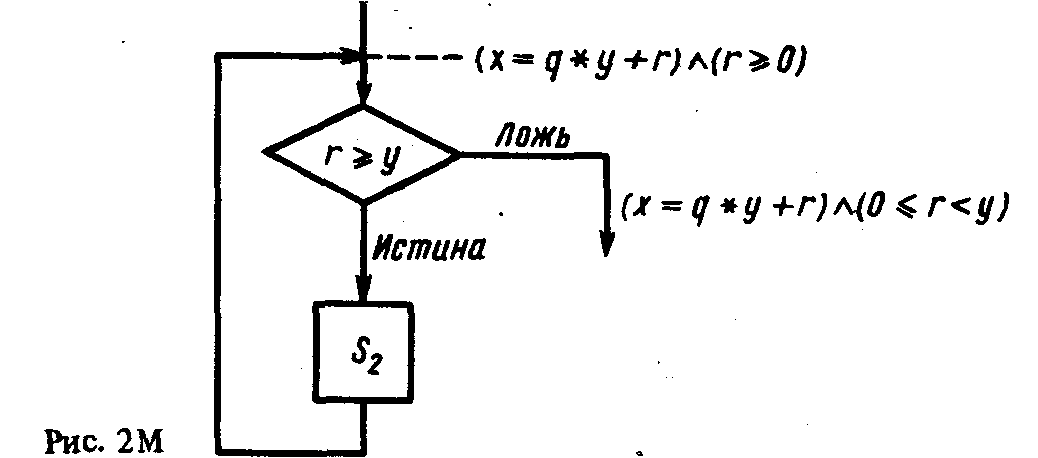
{(x=q\*y+r)n(0<y<=r)}

begin r : = r — у;

q:=1+q

end end {(x=q\*y+r)^(0<=r<y)} (2,58)

Критерий корректности (2.58) выражается соотношением (х= q \*у + r) ^ (0 <= r <у), которое должно иметь место при завершении (2.58),



если (х>=0) ^ (у>0) справедливо перед выполнением (2.58). Это мы и должны доказать.

Рассмотрим сначала цикл в нашей программе, представленный Sз на рис. 2.2 в (рис. 2М). В этом примере В имеет вид (r>= у). Если теперь за­метить, что выходное соотношение на рис. 2.2в может быть переписано как

(х=q\*у+r)^(r>=0)^~(r>=у), то ясно, что это как раз и есть Р ^ ~В, где P — это

(x=q\*y+r)^(r>=O). (2.59)

Другими словами, наша задача при верификации рис. 2М — показать, что

{P} while (r >= у) do S2 {Р ^ ~(г >= у)}, где инвариант цикла Р задается (2.59). Но правило вывода

{P^B}S{P}

{Р} while В do S {Р ^ ~В}

гарантирует, что это утверждение должно быть истинным постольку, по­скольку можно доказать

{P^(r>=y)}S2{P} (2.61) для Р,

задаваемого (2.59), и S2, задаваемого

begin r := r —у; q := 1 + q end,

которое является только версией на языке Паскаль на рис. 2.26. Теперь (x=q\*y+r)^(r>=0)^(r>=y→(x=(1+q)\*y+(r-y))^(r-y>=0). Вспоминая правило вывода для оператора присваивания, получим:

{(x=(l+q)\*y+(r-y))^((r-y)>=0)}r:=r-y{(x=(l+q)\*y+r)^(r>=0)},

{(х = (1 + q) \* у + r) ^ (г >= 0)} q : = 1 + q {(х = q \* у + r) ^ (r >= 0)}.

Из правила вывода для составного оператора и правила консеквенции следует:

{(x=q\*y+r)^(r>=0)^(r>=y)} begin r := r — y',q := 1 + q end

{(x=q\*y+r)^(r>=0)}, (2.62)

которое есть в точности выражение (2.61), необходимое для доказательства. Таким образом, правило вывода (2.60) позволяет вывести требуемое условие

{(х = q \* у + r)^{r >= 0)} S3 {(x= q \*у + г)^(r >= 0)^ ~(г >=у)}. (2.63)

Следующий шаг состоит в проверке того, что требуемое условие

{(х >= 0) ^ (у > 0)} begin q : = 0; r : = х end {(х = q \* у + r) ^ (r >= 0)} (2.64)

выполняется для S1 на рис. 2.2. Действительно,

(х >= 0) ^ (у > 0) → {х = 0 \* у + х) ^ (х >= 0)

{(х = 0 \* у + х) ^(х >= 0)} q := 0 {(х = q \* у + х)^(х >= 0)}

{(x=q\*y+x)^(x>=0)} r:=x {(x=q\*y+r)^(r>=0)}. (2.65)

Из (2.65), используя правило консеквенции и правило для составного оператора, выводим, что (2.64) выполняется.

Теперь из (2.63), (2.64) и правила вывода для составного оператора следует:

{(х>=0)^(у>0)} begin q := 0; r := х;

while r >= у do

begin r := r — у, q := 1 + q end end {(x=q\*y+r)^(0<=r<y)],

что мы и хотели доказать. Так как это было доказательство частичной кор­ректности, условие у > 0 не использовалось при доказательстве. Это усло­вие, однако, будет играть существенную роль при доказательстве того, что алгоритм действительно завершается.

Аппаратные средства персональных компьютеров

Создание современных персональных компьютеров (ПК) стало возможным благодаря серьезным научным прорывам в областях материаловедения, микроминиатюризации радиоэлектронных элементов, информатики, робототехники, прикладной математики. За каких-то пять-шесть лет маленькие дешёвые электронные модули полностью заменили огромные стойки, начинённые дорогостоящими компьютерными компонентами, а современный ПК впитал все накопившиеся в вычислительной технике за долгие годы разработки, принципы, идеи.

Однако чисто внешний эффект миниатюризации современных компьютеров ничто по сравнению с их преимуществом над электронно-вычислительными машинами (ЭВМ) предыдущих поколений (серии ЕС ЭВМ, СМ и т.д.). Прежде всего, это возросшая во много раз производительность, малое потребление электроэнергии, дешевизна технического обслуживания и эксплуатации. Компьютеры трансформировались из ЭВМ, занимавших пространства площадью свыше 100 м2, в ПК, которые размещаются теперь на рабочих столах и даже в карманах пиджаков пользователей.

История создания персонального компьютера

В истории вычислительной техники можно выделить два больших этапа развития. Для первого этапа характерно применение в компьютерах не микросхем, а радиоламп и транзисторов, не оперативной памяти на динамических элементах, а памяти на ферритовых кольцах, не миниатюрных накопителей на магнитных дисках, а огромных стоек с накопителями на сменных магнитных дисках.

Второй этап развития компьютерной техники ознаменовался началом выпуска ЭВМ на модулях микропроцессоров. В 1971 г. был выпущен первый 4-разрядный микропроцессор Intel 4004, который обладал рядом положительных особенностей, свойственных современным микропроцессорам, - универсальностью, гибкостью, миниатюрностью, малой потребляемой мощностью и низкой стоимостью. Подобные микропроцессоры свыше десятилетия использовались как в зарубежной, так и в отечественной вычислительной технике.

Первыми персональными компьютерами принято считать микроЭВМ Altair 8800 производства MITS, которые поступили в продажу в 1975 г. в виде наборов для радиолюбителей "Сделай сам". В этих компьютерах использовались интегральные микросхемы малой (до 100 транзисторов на кристалл) и средней (до 1000 транзисторов на кристалл) степени интеграции, а также 8-разрядный микропроцессор Intel 8080.

В 1977 г. корпорация Apple выпустила в продажу персональный компьютер Apple II, который отличался совершенным (по тем временам) программным обеспечением, рассчитанным на самых неподготовленных пользователей. Простота освоения работы на компьютере позволяла использовать эту технику не только специалистам-радиолюбителям и профессионалам, но и всем желающим приобщиться к миру электроники. Компьютер Apple II был собран на микросхемах средней степени интеграции и имел микропроцессор Motorola 6800.

Начало 80-х знаменуется разделением мира ПК на две фракции, несовместимых друг с другом ПК, - Macintosh (наследники Apple II) и IBM PC. Первые ПЭВМ имели существенные недостатки, ограничивающие распространение ПК в среде неподготовленных пользователей и, в основном, отвечали интересам узкого круга профессионалов. Все они имели закрытую архитектуру. Это означает, что аппаратные средства были представлены одним неразъемным устройством. Совершенствование компьютера было уделом профессионалов-разработчиков. В августе 1981 года фирма IBM выпустила первый ПК, в котором осуществила модульный принцип построения, использовавшийся до этого в больших вычислительных системах. Такое построение, называющееся открытой архитектурой, обеспечивало возможность сборки ЭВМ из независимо изготовленных частей. На системной (материнской) плате размещены только те блоки, которые осуществляют обработку информации. Схемы, управляющие всеми остальными устройствами компьютера, реализованы на отдельных платах, которые вставляются в стандартные разъемы на системной плате - слоты. Это позволяет легко заменять дополнительные устройства на новые при старении прежних, увеличивать объем памяти, добавлять, по мере необходимости, новые устройства.

Начиная с 1981 г. лидерство на компьютерном рынке захватила корпорация IBM с компьютером IBM PC, который в дальнейшем стал стандартом для производителей персональных компьютеров. Эти ПК разрабатывались на базе 8-разрядного процессора Intel 8088. Первые IBM PC имели динамическую память 256 Кбайт, источники питания мощностью 63,5 Вт и накопители на двусторонних гибких магнитных дисках (НГМД) на 360 Кбайт.

В 1983 г. фирма IBM представила расширенный (eXTended) вариант персонального компьютера IBM PC XT. В этой модели появился накопитель на жестких магнитных дисках (НЖМД) на 10 или 20 Мбайт, двусторонние НГМД на 360 Кбайт (5,25 дюйма) или 720 Кбайт (3,5 дюйма), а ёмкость оперативной памяти была расширена до 640 Кбайт. Все модели IBM PC XT обладали более мощными источниками питания (до 150 Вт) и располагали 8-разрядным микропроцессором Intel 8088 и сопроцессором Intel 8087.

В 1984 г. фирма IBM предложила открытый стандарт IBM PC Advanced Technologies (AT), который, как и все предыдущие, был открыт для дублирования всеми ведущими корпорациями-производителями компьютеров. IBM PC AT были разработаны на базе новых 16-разрядных процессоров Intel 80286, которые по сравнению с Intel 8088 имели несколько новых режимов работы. В AT использовался новый сопроцессор Intel 80287. Кроме того, элементная база обрамления процессора впервые содержала модули комплектов микросхем (так называемых чипсетов) с очень высокой степенью интеграции (свыше 100 000 транзисторов на кристалл). ПК IBM PC AT располагал мощным блоком питания (200 Вт), жестким диском (30 Мбайт), оперативной динамической памятью 640 Кбайт с возможностью расширения до 1 Мбайт, двусторонними НГМД (360 Кбайт) и 1,2 Мбайт (5,25 дюйма) или 720 Кбайт (3,5 дюйма).

Так как открытый стандарт AT стал достоянием всех фирм, желающих продвигать свои изделия в ряду самых наукоемких продуктов, то фирма IBM утратила лидирующие позиции в мире ПК. Поэтому она предприняла попытку создания закрытого стандарта. В 1987 г. был представлен новый ПК семейства PS/2, модель 25. Основным отличием PS/2 от PC являлось наличие новой 32-разрядной системной архитектуры МСА (Micro Channel Architecture), несовместимой с шиной ISA (Industry Standard Architecture), применявшейся в PC AT. Магистрали МСА имеют более высокие скоростные данные, чем у ISA, кроме того, разъемы расширения шин не совпадают. Впервые IBM интегрировала на материнскую плату некоторые контроллеры - параллельные и последовательные порты, видеоконтроллер.

ПК IBM PS/2 имели успех очень непродолжительное время, поскольку усилиями корпораций-производителей персональных компьютеров семейства PC, в противовес шине МСА, была разработана более технологичная системная архитектура EISA (Enhanced ISA - усовершенствованная шина стандартной промышленной архитектуры). По своим характеристикам она адекватна МСА. Кроме того, разъемы шины позволяли устанавливать не только платы нового формата, но и ISA, что импонировало широкому кругу пользователей, которые в новом компьютере могли бы использовать старые периферийные устройства и не затрачивать средства на покупку новых. Обе шинные архитектуры просуществовали недолго. Стоимость периферийных устройств и компонентов ПК была высока. Продвижение на рынок персональных компьютеров технологии PS/2 было остановлено и в настоящее время доминирующей технологией является IBM PC.

С 1992 г. фирма Intel приступила к массовому внедрению в ПК архитектуры новой шины PCI (Peripheral Component Interconnect - шина расширения интерфейса периферийных компонентов). Она обладает хорошими показателями производительности, не антагонистична по отношению к другим шинам, с которыми может сосуществовать в пределах одной компьютерной системы. Шина расширяема, т.е. допускает использование плат устройств различного уровня интеллекта и производительности, разрядности, напряжений питания. Архитектура шины позволяет внедрять в нее очень существенные доработки, например графическую шину AGP (Accelerated Graphics Port - быстродействующий графический порт), что повышает общее быстродействие системы.

Шина ISA, в отличии от шин МСА и EISA, благополучно существует и ныне, обслуживая медленные периферийные устройства, доставшиеся пользователям от компьютеров AT. Её можно обнаружить в вычислительных системах по соседству с наборами микросхем и процессорами Pentium, превышающими пропускную способность шины ISA в несколько сот раз. Несмотря на еще достаточно живой интерес пользователей к этой шине, она, по рекомендациям спецификации PC-99, будет постепенно изыматься из системы ПК.

Что собой представляют спецификации РС-Х? Для координации усилий разработчиков аппаратного и программного обеспечения в среде Windows ежегодно проводится конференция WinHEC (Windows Hardware Engineering Conference), на которой производители программного обеспечения представляют свои рекомендации разработчикам аппаратных средств. Основными действующими лицами на конференции являются корпорации Intel и Microsoft. В результате дискуссий и обмена мнениями обнародуется очередная, на год-два, версия спецификации РС-Х, имеющая рекомендательный характер. Например, в соответствии с предыдущей спецификацией PC-98 в качестве базовой была принята система с процессором Pentium 200 МГц и 32 Мбайт ОЗУ. В качестве оптимальной была рекомендована видеосистема с разрешающей способностью 1024х768 пикселей. Было положено начало внедрению в ПК последовательной шины IEEE-1394 (FireWire). Ранее, в PC-97, были рассмотрены меры для повсеместной поддержки системной шины PCI и последовательной универсальной шины USB. Спецификация PC-99 расширила возможности ПК за счет применения процессоров с частотой 300 МГц и оперативной памяти 64 Мбайт. Планируется перевод синхронизации системной шины персональных компьютеров на тактовую частоту 100 МГц, внедрение графической шины AGP, новых типов процессоров с 32-разрядной архитектурой.

Компоненты персонального компьютера

Несмотря на разнообразие типов, форм и архитектур персональных компьютеров (рис. 1.1), в составе большинства ПК можно выделить следующие основные устройства: системный блок, монитор, клавиатура, мышь и периферийные устройства. В зависимости от потребностей и возможностей пользователей состав периферии может быть расширен аудиосистемой, модемом, принтером, сканером и т.д.

Системный блок включает все основные составляющие персонального компьютера. Важнейшим его компонентом является материнская (системная) плата (motherboard). Расположенные на ней электронные модули, или чипсеты, а также центральный процессор (ЦП), оперативная память (ОЗУ) составляют базовый комплект электроники компьютера. В системном блоке также размещаются источник питания, устройства внешней памяти - накопители на жестких магнитных дисках (НЖМД), и накопители на гибких магнитных дисках (НГМД). К этой группе устройств в последние годы присоединились дисководы компакт-дисков (CD-ROM-drive), и накопители на сменных магнитных дисках.

К системному блоку подсоединены все внешние устройства: монитор, клавиатура, мышь, принтер, модем, звуковые колонки, сканер и т.д.

Монитор - отображаете на экране текстовую и графическую информацию, вводимые с клавиатуры или выводимые из компьютера данные, сообщения компьютерной системы, копии всевозможных документов и прочую важную для пользователя информацию.

Клавиатура - предназначена для ввода в компьютер команд и данных.

Мышь позволяет указывать на элементы экрана с помощью указателя и путем щелчка на кнопках выполнять определенные операции.

Принтер - выводит в качестве твердой копии текстовую и графическую черно-белую или цветную информацию. Вывод информации осуществляется на бумагу или на пленку.

Модем - предназначен для подключения компьютера к телефонной линии.

Сканер - обеспечивает ввод в ПК текстовой или графической, черно-белой или цветной информации для ее дальнейшей обработки.

Звуковая система - состоит из звуковой карты и звуковых колонок, вынесенных или встроенных в дисплей. Колонки имеют свои усилители и органы регулировки уровня звука.

Архитектура ПК

Архитектура ПК - это структурная схема внутренней организации и взаимодействия основных функциональных модулей компьютера (ЦП, чипсеты, системы памяти персонального компьютера, контроллеры периферийных устройств и сами периферийные устройства). В понятие архитектуры ПК входят также принципы организации вычислительного процесса и переработки информации.

Детальный анализ архитектуры компьютера содержит описание представления программ и данных - систему счисления, информационные форматы и организацию вычислительного и обменного процессов. Он затрагивает также рассмотрение структуры памяти, методики выполнение машинных операций, системы размещения информации в памяти, системы диагностирования и контроля, а также управления вычислительным процессом.

Как видно, архитектура ПК - ёмкое понятие, рассматривающее тонкости взаимодействия всех его компонентов, объясняющее тончайшие нюансы вычислительных и обменных процессов.

На рис. 1.3 изображена упрошенная блок-схема, которая дает общее представление об архитектуре и основных компонентах компьютера.

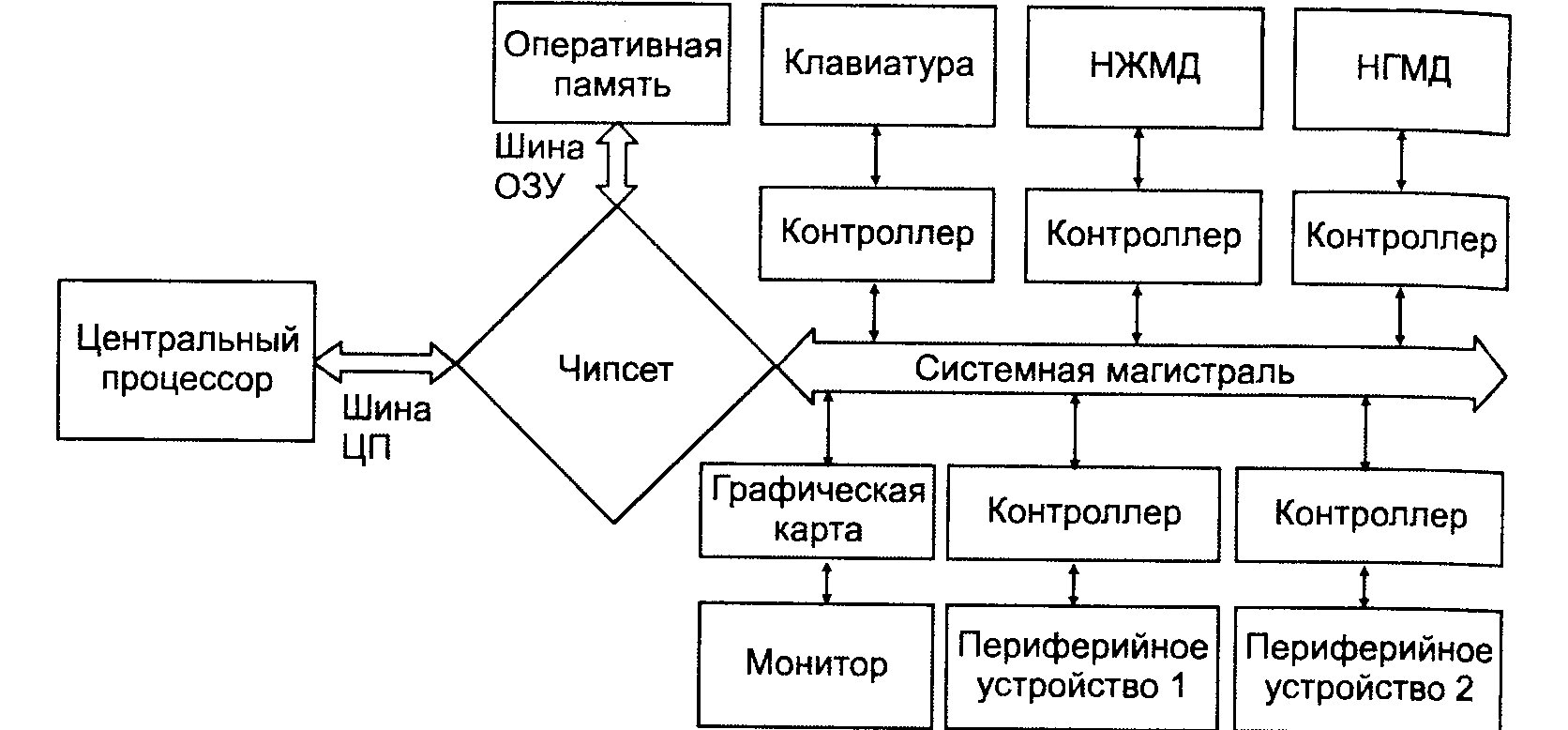


Рис. 1.3. Упрощенная блок-схема компьютера

Компоненты, представленные на блок-схеме, имеют следующее назначение.

Центральный процессор - это электронный модуль, выполняющий в компьютерной системе основную вычислительную работу. Он управляет взаимодействием между всеми блоками и системами компьютера. Именно к ЦП стягиваются все магистрали компьютерной системы. ЦП находится в функциональном центре компьютерной системы, окруженном аппаратными функциональными блоками и подсистемами.

Вообще говоря, процессор - это отдельный модуль компьютерной системы, реализующий определенные вычислительные и обменные процессы. В компьютере может быть несколько процессоров. Одни из них управляют вводом-выводом данных и называются процессорами ввода-вывода. Другие процессоры выполняют вычисления с вещественными числами и называются математическими сопроцессорами. Третьи генерируют изображения на экран монитора и называются видеопроцессорами. Но в любом персональном компьютере есть ЦП, который управляет всей компьютерной системой.

Чипсет - так называется комплект микросхем, предназначенный для поддержки в компьютерной системе функциональных возможностей, предоставляемых процессором, оперативной памятью, кэш-памятью, дисковой и видеопамятью и прочими компонентами системы. Чипсет объединяет различные составные части компьютерной системы. Микросхемы чипсета генерируют большинство сигналов для системных и периферийных компонентов, преобразуют сигналы между шинами, позволяют процессору и оперативной памяти работать с постоянной производительностью.

В состав нескольких микросхем, из которых состоят чипсеты, входят узлы, называемые "обрамлением центрального процессора". Это - таймеры, контроллеры прерываний и прямого доступа к памяти, контроллеры графической шины AGP, последовательного и параллельного портов и прочие устройства, поддерживающие системные процессы в ПК.

Контроллеры - предназначены для управления доступом из системы к какому-либо из устройств, а также для выполнения соответствующих операций информационного обмена. Каждое внешнее устройство имеет свой контроллер. После получения соответствующих команд от центрального процессора контроллер выполняет операции по обслуживанию внешнего устройства.

Как видно из схемы на рис. 1.3, все электронные элементы компьютера обмениваются информацией друг с другом и взаимосвязаны с помощью шин - совокупности линий и микросхем, осуществляющих передачу электрических сигналов определенного функционального назначения между различными компонентами ПК.

Совокупность всех шин информационно-вычислительной системы называется системной магистралью.

По шинам передаются сигналы трех групп: адресные, управляющие и данные. Соответственно различают следующие шины.

Шина данных - предназначена для передачи данных между электронными модулями ПК.

Шина адреса - обеспечивает пересылку кодов адресной информации к ОЗУ или электронным модулям ПК для доступа к ячейкам памяти или к устройствам ввода-вывода.

Шина управления - включает линии, по которым передаются сигналы управления, обмена, запросы на прерывания, передачи управления, синхронизации и т.д.

Шины характеризуются разрядностью, т.е. количеством линий, составляющих шину. Другое определение разрядности - количество одновременно передаваемых по линиям шины битов информации. В архитектуре персональных компьютеров чаще всего встречаются 8-, 16-, 32- и 64-разрядные шины.

Бывают шины последовательные и параллельные. Последовательная шина состоит из одной линии данных, которые передаются последовательно, бит за битом, и нескольких линий управления и адреса. Параллельная шина состоит из нескольких линий данных, адреса и управления.

Пропускной способностью шины называется количество информации, которое может быть передано по каналу за единицу времени.

Для пересылки данных между устройствами существует несколько стандартных процедур.

Процедура прерывания - это механизм, позволяющий ЦП обработать требования устройства на обмен данными с другим устройством или с памятью. Выполнение процедуры прерывания начинается с генерирования каким-либо из устройств, например клавиатурой или накопителем на гибких дисках, сигнала запроса на прерывание IRQ (Interrupt Request). Запрос может поступить также и в том случае, если системой выявлена аварийная ситуация.

Каждое из устройств, работающих по прерываниям, имеет собственную линию IRQ запроса на прерывание.

Линии IRQ - это физические линии на материнской плате, позволяющие передавать запросы на прерывания от контроллера устройства контроллеру прерываний, которым эти прерывания обрабатываются.

Контроллер прерываний при поступлении сигнала IRQ прерывает работу ЦП, формирует уникальный для каждого устройства код, посылает его в ЦП. В соответствии с этим кодом процессор формирует адрес обращения в оперативную память, который называется вектором прерывания. По адресам векторов прерываний расположены адреса обращений к программам-обработчикам прерываний. Программа-обработчик прерываний реализует ту функцию, потребность в которой вызвало данное прерывание.

Если устройство-источник прерываний расположено за пределами материнской платы, то прерывания от него называются внешними аппаратными. Если устройство, генерирующее прерывания, находится в пределах материнской платы, то такие прерывания называются внутренними аппаратными. Если источником прерываний является одна из команд выполняемой программы, то такое прерывание обрабатывается непосредственно в ЦП и называется программным прерыванием. Если прерывание является реакцией на ошибки процессора, например деление на нуль, то такие прерывания называются внутренними.

Другой способ, позволяющий устройствам обмениваться данными с оперативной памятью, называется прямым доступом к памяти (Direct Memory Access или DMA). В этом процессе участвует контроллер DMA, расположенный на материнской плате, к которому сходятся линии запросов DMA (Direct Request - DRQ) от периферийных устройств. Контроллер DMA блокирует линии управления от ЦП, и сам генерирует сигналы, необходимые для обмена данными между оперативной памятью и устройством. Таким образом, между оперативной памятью и устройством происходит быстрое "перекачивание" данных без участия ЦП.

Системные компоненты ПК

Персональный компьютер содержит множество электронных элементов, которые объединяются в более крупные компоненты, - модули, узлы, цепи, схемы, блоки и т.д. Если из всего этого разнообразия электронных компонентов изъять хотя бы один, то вся информационно-вычислительная компьютерная система перестанет работать. Вместе с тем, важность выполняемой различными электронными узлами работы для ПК неравнозначна. Одни устройства, например ЦП или ОЗУ, принимают участие практически во всех без исключения действиях, выполняемых ПК. Другие устройства, например контроллеры периферийных устройств, менее активны.

Рассмотрим подробнее, как работают такие важные компоненты материнской платы, как ЦП, память, порты, и как они взаимосвязаны с помощью шин.

Современный центральный процессор

Как вам уже известно, ЦП - это основной электронный модуль на материнской плате, который выполняет вычислительную работу, управляет обменом данными с оперативной памятью и устройствами ввода-вывода. ЦП, являясь аппаратным центром информационно-вычислительной системы, отвечает за характеристики производительности ПК.

ЦП работает циклически и упрощенно его работу можно описать следующим образом. В начале очередного цикла ЦП считывает из ОЗУ команду, расшифровывает ее и реализует указанные в ней действия. Например, считывает из памяти два числа, расположенные в памяти по определенным адресам, указанным в выполняемой команде, суммирует их и результат записывает в заданную ячейку ОЗУ. После этого цикл повторяется: считывается очередная команда (или команда, адрес которой указан в предыдущей команде), выполняются указанные в ней действия и т.д.

Центральный процессор оперирует целочисленными данными. Если необходимо выполнить вычисления с более высокой степенью точности, в работу включается встроенный в ЦП узел математического сопроцессора. Математический сопроцессор реализует высокоскоростные вычисления с вещественными числами, вычисления тригонометрических функций и т.д.

Важными характеристиками ЦП являются тактовая частота, скорость выполнения команд и разрядность шин данных и адреса.

Все современные процессоры располагают 64-разрядной шиной данных. По разрядности адресной шины процессора можно судить об адресном пространстве памяти. Например, если ширина шины адреса 32 разряда, то процессору доступно пространство памяти 4 Гбайт, а при 36-разрядной шине - 64 Гбайт.

Адресное пространство памяти состоит из всех ячеек ОЗУ, видеопамяти и памяти BIOS (Basic Input Output System - базовая система ввода-вывода), к которым может обращаться процессор.

Процессор обращается не к отдельной ячейке памяти и даже не к байту, а сразу к нескольким байтам, которые в информатике называются "слово". Все "слова" должны располагаться по четным адресам обращения к памяти. Процессор одновременно может обратиться, например, сразу к восьми байтам, если располагает 64-разрядной шиной данных.

Вся доступная память разбита на модули, предназначенные для хранения отдельных частей программного обеспечения и данных (рис. 1.4). Так, например, по самым младшим адресам (в пределах 640 Кбайт) располагаются векторы прерываний, данные BIOS и программные модули операционной системы. За пределами 640 Кбайт 128 Кбайт занимает видеопамять, а за ней 256 Кбайт отводятся для хранения различных компонентов BIOS. За пределами 1 Мбайт памяти хранятся коды команд и данных защищенного режима работы системы.

|  |
| --- |
| РАСШИРЕННАЯ  ПАМЯТЬ |
| BIOS |
| ВЕРХНЯЯ ПАМЯТЬ  UPPER MEMORY AREA |
| ОСНОВНАЯ ПАМЯТЬ  BASE MEMORY |

Рис. 1.4. Карта распределения доступной памяти ПК IBM PC.

Все процессы, связанные с вычислениями, обработкой и пересылками данных между электронными модулями компьютера, должны быть синхронизированы во времени.

Синхронизация ЦП и всех узлов компьютера осуществляется с помощью специального электронного узла - тактового генератора, расположенного на материнской плате. Тактовый генератор формирует периодические последовательности тактовых импульсов, которые направляются и в ЦП, и в систему памяти, и во все остальные подсистемы компьютера. Однако необходимо отличать частоту тактового генератора и тактовую частоту процессора. В ЦП поступают тактовые сигналы с материнской платы (от тактового генератора), а необходимая для работы внутрипроцессорная частота получается при умножении частоты материнской платы в специальном встроенном блоке умножения. Она-то и синхронизирует вычислительные процессы непосредственно в самом процессоре. Таким образом, если материнская плата может работать на фиксированной частоте, например 100 МГц, то процессор может синхронизироваться, например, частотой 450 МГц.

Частота синхронизации электронных компонентов на материнской плате (типичные значения 66, 75, 83, 100 МГц) синхронизирует обменные процессы вне процессора, например между оперативной памятью и ЦП, устройствами ввода-вывода и т.д. С ростом частоты синхронизации повышается скорость работы ЦП и компьютера в целом.

Для ускорения доступа к ОЗУ и увеличения производительности в ЦП встроен модуль кэш-памяти (кэш). Это промежуточная память между ЦП и ОЗУ. Кэш малоёмкое, но более быстрое, чем ОЗУ, хранилище данных.

Кэш работает следующим образом. Все свои запросы ЦП адресует одновременно ОЗУ и кэш. Если адреса обращения идентичны, то констатируется попадание в кэш. В этом случае данные или команды считываются из более быстрой памяти, что повышает общую производительность ПК. Если кэш не располагает информацией по искомому адресу, процессор обращается за ней к ОЗУ. В этом случае скорость обмена данными замедляется.

Таким образом, современные ЦП - это высокотехнологичные электронные изделия, выполняющие сотни миллионов операций в секунду и позволяющие ПК решать очень сложные задачи за короткие промежутки времени.

В изготовлении процессоров задействованы технологические процессы, практически полностью протекающие под управлением робототехники, выполняемые при этом операции отличаются высокой точностью и гарантируют отсутствие в кристалле микросхемы примесей, не предусмотренных в документации технологического процесса.

Одним из основных параметров, определяющих главные характеристики процессора, является показатель конструктивных технологических норм (измеряется в мкм), определяющий минимально допустимые размеры элементов топологии микросхем. Например, норма 0,25 мкм свидетельствует о том, что минимальное расстояние между двумя соседними транзисторами составляет 0,25 мкм. Чем меньше норма, тем выше скорость информационного обмена, меньше потребление тока и нагрев элементов, и тем плотнее "набивка" кристалла интегральной микросхемы.

Процессоры Pentium производства 1993 г. изготавливались по технологическим нормам 0,8 мкм, современные процессоры имеют конструктивные нормы на уровне 0,25-0,18 мкм. Снижение технологических норм позволяет переходить на более высокие внутрипроцессорные частоты. Если процессоры с технологическими нормами 0,35 мкм работают на частотах 133—233 МГц, то для кристаллов с нормами 0,25 мкм характерны скорости от 300 МГц и выше. Процессоры, изготовленные в соответствии с 0,25 мкм нормами, имеют напряжение питания 1,8 В.

Процессоры Intel и их клоны - базовая основа IBM-совместимых компьютеров (клонами принято называть компьютеры и их компоненты, разработанные сторонними фирмами в соответствии с характеристиками и параметрами фирм-стандартизаторов).

Корпорация Intel является разработчиком шести поколений процессоров, составляющих аппаратное ядро ПК семейства IBM PC. В таблице 1.1 представлены краткие характеритики производимых компанией Intel процессоров.

Таблица 2.2. Краткие характеристики процессоров Intel

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Микропроцессор | Количество транзисторов на кристалл (млн.) | Начало производства (год) | Шина данных (разрядность) | Тактовая частота (МГц) |
| 8086 | 0,029 | 1978 | 16 | 4,77-10 |
| 8088 | 0,029 | 1979 | 8 | 4,77-10 |
| 80286 | 0,134 | 1982 | 16 | 6-25 |
| 80386DX | 0,275 | 1985 | 32 | 16-40 |
| 80386SX | 0,275 | 1988 | 16 | 16-33 |
| 80486DX | 1,2 | 1989 | 32 | 20-50 |
| 80486SX | 1,185 | 1991 | 32 | 16-33 |
| 80486DX2 | 1.3 | 1992 | 32 | 40-66 |
| 80486DX4 | 1,35 | 1992 | 32 | 100-132 |
| Pentium | 3,1 | 1992 | 64 | 60-100 |
| Pentium MMX | 3,8 | 1996 | 64 | 120-300 |
| PentiumPro | 36,5 | 1996 | 64 | 150-300 |
| Pentium II | 7,8 | 1997 | 64 | 233-450 |
| Pentium III | 9,5 | 1999 | 64 | 450-500 |

Процессоры Pentium MMX (Multimedia Extantive - расширение мультимедиа) начали продвижение на компьютерном рынке в 1996 г. Они отличаются от классического Pentium тем, что располагают средствами для обработки дополнительной группы мультимедиа-команд, используемых для специальных целочисленных вычислений. Для их обработки в процессор встроены дополнительные аппаратные модули. В этих процессорах также используется метод обработки нескольких блоков данных одной командой, что позволяет использовать технологию MMX для решения весьма широкого спектра задач мультимедийного и коммуникационного плана. Массовое внедрение новой технологии Pentium MMX позволило корпорации Intel уже в 1997 г. прекратить выпуск классических процессоров Pentium.

Процессоры различных поколений имеют преемственность на аппаратном и программном уровнях. В настоящее время ПК оснащаются процессорами Intel пятого-шестого поколений. К пятому поколению относятся всевозможные модификации процессоров Pentium, включая Pentium MMX.

Первым представителем шестого поколения процессоров Intel является PentiumPro, поступивший на компьютерный рынок в 1996 г. Он имеет стандартную для процессоров Pentium архитектуру, но с существенными доработками. К шестому поколению относится Pentium II.

Для удовлетворения массового спроса пользователей корпорация Intel разработала ряд интересных процессоров для "недорогих систем" с ограниченными возможностями расширения. В этих недорогих компьютерах можно разместить меньше периферийных устройств, их быстродействие ниже, чем у систем с "полноценными" Pentium II. В 1998 г. в производство компьютеров низкоценового уровня было внедрено целое созвездие процессоров, объединенных общим названием Celeron.

В 1998 г. для применения в мощных серверах и рабочих станциях был разработан процессор Pentium II Хеоn (рис. 2.3). Этот процессор создан в ходе реализации программы Intel Inside, которая опирается на все наиболее важные достижения архитектуры Р6 (обозначение к процессорам шестого поколения Intel). Процессор Pentium II Хеоn располагает всеми средствами температурного контроля и стабилизации напряжения, присущими Pentium II.

Процессоры шестого поколения Pentium III и Pentium III Xeon — это усовершенствованные варианты процессоров Pentium II и Pentium II Xeon. Pentium III имеет следующие основные характеристики:

Изготавливается по 0,25 мкм технологии.

Работает на тактовой частоте 450 и 500 М Гц.

Содержит дополнительные компоненты для обработки 70 новых команд, улучшающих работу с графическими программами.

Располагает разделенной кэш-памятью первого (32 Кбайт) и второго (512 Кбайт) уровней.

Поддерживает работу с двумя процессорами.

Pentium III Xeon имеет дополнительные, по сравнению с Pentium III, возможности:

работает на тактовых частотах 500 и 550 МГц;

поддерживает совместную работу от 2 до 8 процессоров;

выпускается с кэш-памятью второго уровня 512 Кбайт, 1 Мбайт, 2 Мбайт (500 МГц) или 512 Кбайт (550 МГц).

В настоящее время компания Intel ведет разработки полностью 64-разрядного процессора седьмого поколения с кодовым именем Merced, который, предположительно, будет работать с тактовой частотой 1000 МГц.

Шины ПК

Рассмотрим более подробно блок-схему современного ПК (рис. 1.6).

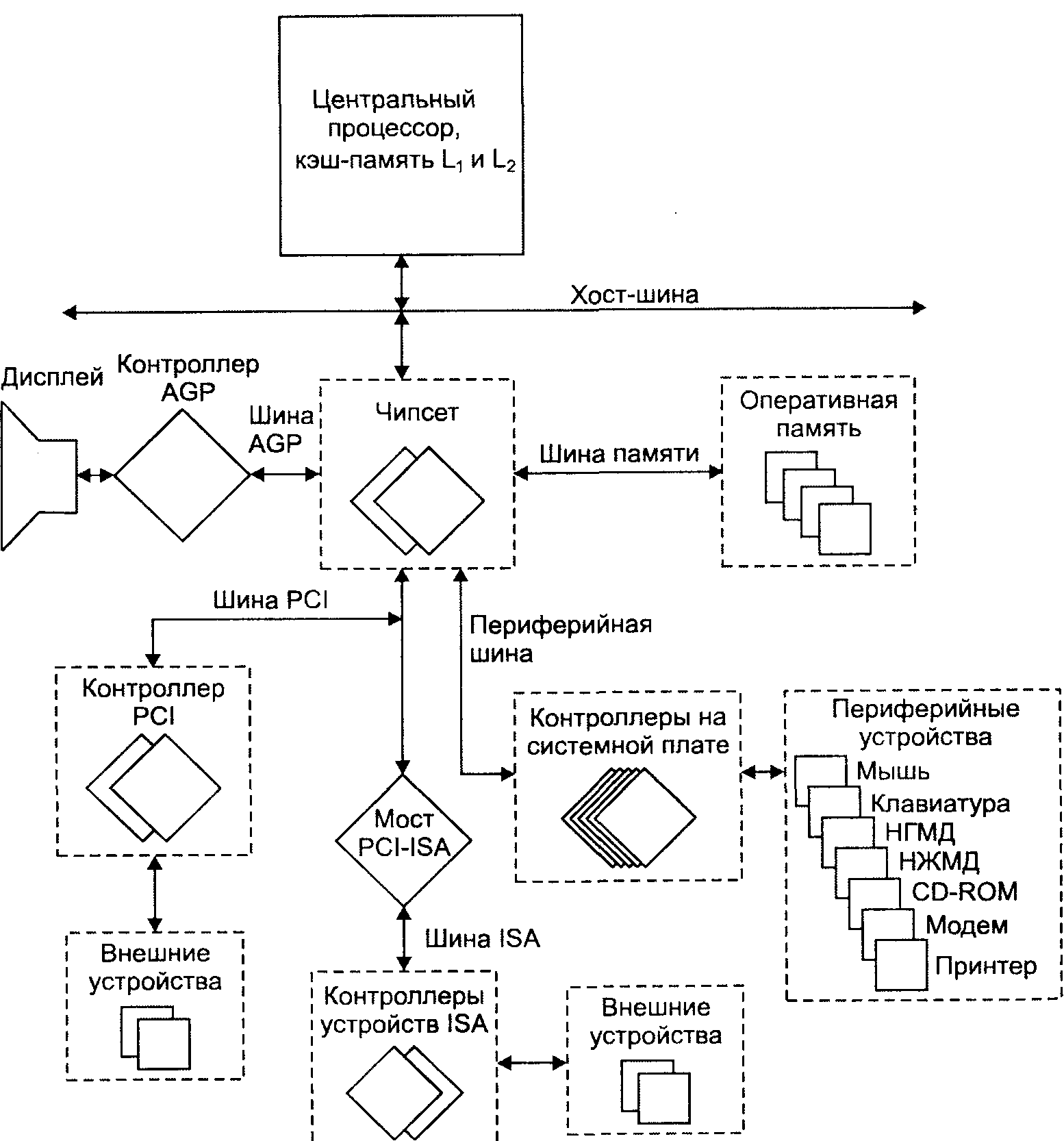


Рис. 1.6. Блок схема ПК.

Электронные модули компьютера объединяются шинами нескольких типов. Существуют: шина процессора (хост-шина), шина ОЗУ, системные шины, периферийные шины.

Хост-шина - самая быстрая из шин ПК; она соединяет ЦП со всеми остальными компонентами. Шина ОЗУ также очень быстрая, но в отличие от хост-шины имеет меньшую пропускную способность.

В IBM-совместимых компьютерах используются следующие стандарты шин:

ISA - стандартная индустриальная шина (Industrial Standard Architecture);

EISA - усовершенствованная шина ISA (Enhanced ISA);

МСА - шина микроканальной архитектуры (Micro Channel Architecture);

VL-bus, или VESA-bus - стандарт локальной шины, разработанный Video Electronic Standards Association (VESA Local bus);

PCI - локальная шина для подключения периферийных устройств (Peripheral Component Interconnect).

В таблице 1.2 представлены характеристики наиболее распространенных системных шин, используемых в ПК.

Таблица 1.2. Характеристики системных шин

|  |  |  |  |
| --- | --- | --- | --- |
| Системная шина | Частота на материнской плате (МГц) | Разрядность шины (бит) | Скорость передачи данных (Мбайт/с) |
| ISA | 8 | 16 | 8 |
| EISA | 8,33 | 32 | 33,32 |
| МСА | 10 | 32 | 21,6 |
| VESA-bus 1.0 | 33 | 32 | 130 |
| PCI 2.0 | 33 | 32 | 132 |
| PCI2.1 | 66 | 64 | 528 |

В одном ПК могут встречаться комбинации шин, например шины PCI и ISA. Стандартные шины имеют разъемы расширения, к которым подсоединяются платы контроллеров периферийных устройств. Для согласования электрических характеристик между шиной PCI и шинами других типов устанавливаются электронные мосты, благодаря которым на системной шине PCI удается разместить до десяти плат расширения.

Графическая шина AGP (Accelerated Graphics Port) помогает системной шине PCI разгрузиться от потока видеоданных. Для работы этой шины в систему добавлены некоторые аппаратно-программные компоненты. Плата графического акселератора должна поддерживать функции и режимы работы протокола AGP. Плата графического контроллера AGP устанавливается в специальный разъем на материнской плате. Графический контроллер AGP может быть встроен также на материнскую плату. В этом случае в дальнейшем видеоподсистему нельзя будет модернизировать.

Для аппаратной поддержки шины AGP используются системные чипсеты (такие как Triton 82440BX, VIA VP3 (ETEQ6628), SIS5591, Ali Aladdin V), способные за один такт системы выполнить две операции - передачу данных между процессором и шиной PCI, а также чтение команд и видеоданных из оперативной памяти. Причем на шине AGP организовано два разделенных канала - для команд и видеоданных.

Шина AGP имеет несколько версий. Первая версия AGP 1х обеспечивает скорость передачи до 264 Мбайт/с при тактовой частоте 66 МГц. Версия AGP 2х поддерживает скорость передачи 528 Мбайт/с на частоте 66 МГц, а AGP 4х - 1 Гбайт на частоте 133 МГц. Последняя версия шины несовместима с ранними версиями.

Система памяти

Система памяти ПК включает следующие компоненты: оперативную память, сверхоперативную память, кэш-память, память базовой системы ввода-вывода и память системного конфигуратора и часов реального времени.

Оперативная память (ОЗУ) - это основная память компьютера, предназначенная для хранения текущих данных и выполняемых программ, а также копий отдельных модулей операционной системы. Большинство программ в процессе выполнения резервируют часть ОЗУ для хранения своих данных. К данным, хранящимся в ОЗУ, ЦП может обращаться непосредственно, используя хост-шину. После отключения питания компьютера все содержимое ОЗУ стирается.

Каждая ячейка ОЗУ может хранить данные объемом в один байт и имеет свой уникальный адрес. Для удобства управления оперативная память разбивается на банки (memory banks). Ёмкость и разрядность используемых в банках микросхем зависит от конструкции материнской платы.

На первых ЭВМ объем оперативной памяти составлял всего 1 Мбайт. Причем под программы пользователя отводилось только 640 Кбайт, так называемая основная память. Остальная часть памяти резервировалась для служебных целей: для передачи изображения на экран; для обеспечения взаимодействия с дополнительными устройствами компьютера и др. Эта часть памяти носит название верхней памяти.

Отдельной строкой стоит BIOS - базовая система ввода-вывода (Base Input-Output System). Ее основные функции:

Инициализация и начальное тестирование аппаратных средств.

Настройка и конфигурирование аппаратных средств и системных ресурсов.

Выполнение основных низкоуровневых операций ввода-вывода.

Загрузка операционной системы.

Что это означает? В настоящий момент разработано очень много различных типов дополнительных устройств (например, много различных видов жестких дисков). Если машине каждый раз придется определять, какие устройства, с какими параметрами присутствуют в схеме, то подготовка ЭВМ к работе заняла бы столько времени, что работать было бы уже некогда. Поэтому некоторые характерные особенности устройств (в частности дисководов и жестких дисков) хранятся в BIOS.

Вся оперативная память кроме BIOS является энергозависимой, т.е. при выключении компьютера ее содержимое очищается. Только BIOS питается от специальных батареек или аккумуляторов, что гарантирует сохранение настроек.

640 Кбайт памяти очень быстро перестали устраивать пользователей. Поэтому вскоре были разработаны расширенная (extended) и дополнительная (expanded) памяти.

Дополнительная память представляет собой специальную плату расширения памяти. Доступ к этой памяти осуществляется через специальную программу. Другая возможность увеличения размера доступной памяти появилась с созданием процессора Intel-80286. Он и последующие процессоры позволяли работать с памятью свыше 1 Мбайта. Такая возможность достигается использованием наряду с реальным режимом (real mode - когда доступны только 640 Кбайт) защищенного режима (protected mode). Этот режим и позволял использовать память, расположенную после 1 Мбайта, которая получила название расширенной памяти.

Микросхемы памяти имеют четыре основные характеристики:

Тип - обозначает статическую или динамическую память;

Объем - показывает емкость микросхемы;

Структура - это количество ячеек памяти и разрядность каждой из них;

Время доступа - характеризует скорость работы микросхемы памяти.

Сверхоперативное запоминающее устройство (СОЗУ) расположено внутри самого ЦП. Эта память очень маленького объема - всего несколько десятков байт, без которой ПК работать не будет. Дело в том, что после включения компьютера операционная система загружает в СОЗУ всю информацию, необходимую для работы процессора. Можно сказать, что операционная система программирует процессор. В процессе работы ЦП сохраняет в СОЗУ результаты своей вычислительной деятельности.

Кэш-память (кэш) выполняет промежуточное хранение данных при обмене между ЦП и ОЗУ и служит как бы "карманом" между ними. Она позволяет повысить скорость информационного обмена в целом.

В ПК имеется кэш нескольких уровней. Самая быстрая кэш, кэш первого уровня, располагается на кристалле процессора (как правило 32 Кбайт). Кэш второго уровня чаще всего выносится за пределы ЦП, однако, имеются процессоры и со встроенной кэш второго уровня (128 Кбайт-1 Мбайт).

Флэш-память - предназначена для хранения программного обеспечения BIOS - базовой системы ввода-вывода операционной системы. BIOS - это набор программ, хранящийся во флэш-памяти. Эти программы выполняют операции запуска и управляют стандартными периферийными устройствами ПК.

Современная флэш-память - это разновидность ППЗУ с электронным стиранием, гарантирующая свыше 100000 перезаписей данных.

Микросхемы этой памяти имеют емкость 4-512 Мбит, что вполне достаточно для хранения следующих программ:

обслуживания устройств ввода-вывода;

проверки оборудования ПК;

инициирования загрузки операционной системы;

программы настройки конфигурации ПК (setup).

Память часов реального времени и конфигурационных установок системы - предназначена для организации часов реального времени и хранения сведений о системе. Содержимое этой памяти не должно уничтожаться после выключения компьютера, поэтому она имеет постоянное питание от автономного источника - аккумулятора 3,6 В. Данные в этой памяти пользователь может прочитать или изменить с помощью программы Setup. Часы реального времени используют для хранения информации 14 байтах. Информация о конфигурации системы хранится в 114 байтах.

Магнитные накопители.

Накопители в зависимости от способа записи на носитель делятся на:

накопители на магнитной ленте

накопители на дисках.

В свою очередь ленточные накопители делятся на:

на катушечной магнитной ленте

на кассетах с магнитной лентой

Дисковые накопители делятся

ГМД (FDD, дискеты)

ЖМД (HDD, винчестер)

На оптических дисках (CD-ROM)

На перезаписываемых оптических дисках (WORM-CD)

На магнитооптических дисках.

Ленточные накопители используются достаточно редко. В основном для хранения больших архивов данных, чтобы в случае выхода компьютера из строя можно было после ремонта восстановить исходные программы и данные.

Гибкие магнитные диски (ГМД) служат для переноса информации с одного компьютера на другой. Сейчас наиболее распространены дискеты размером 3,5 дюйма (89 мм). Емкость этих дискет обычно составляет 1,44 Мбайта. На некоторых компьютерах еще можно встретить и 5,25-дюймовые дисководы (133 мм), но из-за плохой защиты от физических повреждений они гораздо чаще выходят из строя и поэтому используются все реже и реже.

Жесткий магнитный диск (ЖМД) предназначен для постоянного хранения информации, используемой при работе с компьютером. Часто его называют винчестером. Третья главная характеристика мощности компьютера - объем жесткого диска. Измеряется он в мегабайтах.

Вторая существенная для пользователя характеристика винчестера - время доступа к информации. От него зависит, насколько быстро будет производиться работа с этим диском.

В связи с повышением возможностей ЭВМ растет и объем используемой информации. Если раньше самые большие программы занимали несколько сот килобайт, то теперь программа меньше 10 мегабайт считается маленькой. Дискеты уже не справляются с такими объемами информации. Поэтому компьютеры все больше используют для передачи информации компакт-диски. При этом есть CD-ROM - диски, с которых можно только считывать информацию и компакт-диски, на которые данные можно записывать. Такие диски стоят гораздо дороже.

Видеосистема.

Видеоадаптер (видеокарта, видеопамять) служит для хранения изображения передаваемого на монитор. Он нужен для того, чтобы процессору не было необходимости постоянно формировать изображение, и он мог бы заниматься другими задачами, отвлекаясь на работу с видеоизображением только по мере надобности.

Монитор компьютера предназначен для вывода на экран текстовой и графической информации. Они бывают цветными и монохромные, т.е. использовать для создания изображения на черном фоне точки различной яркости. Мониторы могут работать в текстовом и графическом режиме. Текстовый режим отличается очень экономным расходованием системных ресурсов. Экран монитора условно разбивается на отдельные участки, чаще всего 25 строк на 80 символов. В каждом участке может быть выведен один из заранее заданных символов.

Графический режим монитора предназначен для вывода на экран рисунков, графиков, анимации и т.д. Экран монитора состоит из точек (пикселов). Каждая точка может быть темной, светлой, либо иметь один из цветов. ( На цветных мониторах для создания точки нужного цвета используют три стоящих рядом точки красного, синего и зеленого цветов. Меняя яркость этих точек можно добиться любого необходимого цвета ). Количество точек на экране называют разрежающей способностью или разрешением монитора. Например: 640х480, 800х600. Последний означает, что у монитора 800 точек по горизонтали и 600 по вертикали. Еще одной важной характеристикой монитора является его размер. Измеряется в дюймах. Самыми распространенными являются 14-тидюймовые (35,5 см. по диагонали).

На четкость изображения на экране существенно влияет размер точки или пиксела. Чем меньше эта характеристика, тем более четким получится изображение. Эта характеристика называется зерном экрана. Пример: на мониторе размером 14 дюймов при разрешении 640х480 необходимо зерно не больше 0,31 мм, а для режима 1024х768 - 0,28 или 0,25 мм.