Глава 1 Контрольные вопросы

1. Понятие концептуального класса и элемента концептуального класса.

Сущность - все, с чем человек имеет дело в реальном или математическом мире. Сущности, одинаковые с некоторой точки зрения, объединяются в концептуальные классы. Например: ЧЕЛОВЕК, ДЕТАЛЬ,ГЕОМЕТРИЧЕСКАЯ ФИГУРА. Сущность, входящую в состав некоторого концептуального класса, будем называть элементом этого концептуального класса.

2. Основная цель обработки информации.

Информация – мера осведомленности о некотором элементе концептуального класса, представленная в виде данных.

Цель обработки информации - увеличение меры осведомленности относительно элемента заданного концептуального класса.

Данные – совокупности символов (конструктивных), используемые для представления информации и организованные в соответствии с формальными правилами и принятыми нормами.

Символ (конструктивный) можно записывать, читать и хранить, а
также передавать по линиям связи. По сути дела, конструктивный символ
является состоянием некоторого носителя символов.

Говоря другими словами, данные представляют собой обозначения объектов и процессов, а также характеристик этих объектов и процессов

3. Атрибуты и коды значений атрибутов.

Информация (осведомленность) о совокупности групповых признаков отождествляется с дефиницией концептуального класса. Дефиниция концептуального класса представляется в виде совокупности атрибутов – свойств, присущих элементам концептуального класса. Каждый атрибут представляет одно какое-либо неотъемлемое
свойство элемента и идентифицируется уникальным именем. Таким образом, дефиниция концептуального класса представляется в виде:

$N \rightarrow S(A1,A2,\ldots,Ai,\ldots, Ak)$; (I.1)

где N –имя концептуального класса, Ai – имя атрибута, S – совокупность атрибутов.

Каждое допустимое значение A_i^{ξ}
обозначается совокупностью символов d_i , которая образует код значения атрибута - единицу данных

4. Понятие единицы данных и набора данных.

Каждое допустимое значение A_i^{ξ}обозначается совокупностью символов d_i , которая образует код значения атрибута - единицу данных.

Совокупность индивидуальных признаков объекта (значений всех его атрибутов) образует совокупность единиц данных – набором данных.

Пусть совокупность групповых признаков концептуального класса N представляется в виде дефиниции: $N \rightarrow S(A1,A2,\ldots,Ai,\ldots,Ak)$. Пусть атрибут Ai в качестве допустимых кодов значений имеет множество единиц данных Di. Тогда набор данных, обозначающий элемент χ этого концептуального класса, представляется как: $D(d\chi 1,\ldots,d\chi l,\ldots,d\chi k)$.

5. Внешнее и внутреннее представление данных.

Человек, общающийся с компьютером, имеет дело с тремя мирами и двумя уровнями представления данных.

- Реальный мир, который существует вне зависимости от владельца компьютера, и информацию о котором человек желает хранить и обрабатывать.
- Модельный мир – внешние наборы данных как представление информации об объектах реального мира в виде моделей. Под моделью (математической, кибернетической, информационной) подразумевается представление информации об объектах реального мира посредством удобных для человека формализмов, отражающих необходимые характеристики этих объектов;
- Компьютерный мир – машинные наборы данных как представление информации о моделях в виде, эффективном с точки зрения ее обработки компьютером.

6. Проблемы использования информации в человеческой деятельности.

Проблема 1. Передача информации на расстояние. Для конкретного объекта выделяется совокупность атрибутов и обозначения характеристик этих атрибутов (данные) передаются от передатчика к приемнику по линиям связи. При этом обращается внимание на скорость и безошибочность передачи данных.

Проблема 2. Хранение информации. Для конкретного объекта выделяется совокупность атрибутов и коды характеристик этих атрибутов (данные) представляются одним из возможных состояний носителя информации (бумага, память компьютера). При этом обращается внимание на надежность и стоимость хранения информации.

Проблема 3. Обработка информации. Для конкретного объекта выделяется совокупность атрибутов, среди которой выделяются первичные атрибуты (величины которых известны) и вторичные атрибуты (величины которых необходимо определить в процессе обработки информации). Разрабатывается алгоритм, преобразующий значения первичных атрибутов (данных) в значения вторичных атрибутов (данных).

Проблема 4. Кодирование информации: конструирование формальных языков и формальных кодов, обозначающих характеристики объектов реального мира; создание наборов данных для обозначения конкретных объектов реального мира с целью обработки информации об этих объектах.

Глава 2 Контрольные вопросы

1. Как определяется позиционная система счисления (кодирования) целых неотрицательных чисел?

Целое число без знака определяется как характеристика (мера) количества элементов в множестве. Существует еще одно определение целого без знака – характеристика порядка расположенных в ряд элементов.

Определена алгебра целых чисел без знака (целых неотрицательных чисел):

Ацбз = <N+; ⊕, ⊗>

Для того чтобы иметь возможность пересылать числа по линиям связи и выполнять над числами арифметические операции, каждое число должно быть обозначено совокупностью конструктивных символов. Такое обозначение называется записью или кодом числа. Рассмотрим проблему создания системы обозначений целого числа без знака. При создании такой системы необходимо выполнить два условия:

• множество целых чисел без знака бесконечно и, следовательно, необходимо придумать также бесконечное число уникальных символьных записей (обозначений, кодов), позволяющих отличать числа друг от друга;

• как говорилось ранее, записи (обозначения, коды) должны быть не только уникальными, но также допускать достаточно простые алгоритмы выполнения арифметических операций как действий по преобразованию символьных обозначений.

Каждый, наверное, занимался усовершенствованием системы примитивных записей целых без знака. Для этого, последовательность единиц группировалась в десятки, десятки группировались в сотни и т.д Проделав такое позиционирование, мы можем представить число в следующем виде: ...((d3*10+d2)*10+d1)*10+d0

Здесь, d3,d2,d1,d0 - полное число тысяч, сотен, десятков и единиц соответственно.

Мы перешли к алфавитной позиционной системе счисления - алфавитной системе кодирования целого числа без знака, предложенной, как считается, арабами и основанной на следующих принципах:

• вводится конечное, относительно небольшое число знаков - алфавит цифр;

• каждая линейная последовательность цифр образует запись (обозначение) некоторого числа. Используя конечный алфавит цифр, можно получить бесконечное

количество различных записей, каждая из которых обозначает одно и только одно целое число без знака. Однако не следует забывать, что эти цепочки должны быть самоинтерпретирующимися, т.е. нести в себе информацию о величине числа.

Этому условию удовлетворяет алфавитная позиционная система обозначения (кодирования) целых чисел без знака. Ее также называют позиционной системой счисления. Прежде всего, выделим два целых числа без знака, которые играют особую роль: число, обозначаемое как 0, характеризует отсутствие элементов в множестве; число, обозначаемое как 1, характеризует минимально - возможное увеличение или уменьшение числа элементов в множестве.

Концепция позиционной B-ичной системы кодирования (счисления) целых чисел без знака заключается в четырех пунктах.

1. Выбирается произвольное, большее единицы, число, которое называется основанием системы счисления и обозначается как B. Числа меньшие B образуют базу системы счисления.

2. Для чисел базы выбираются уникальные обозначения, которые называются цифрами.

3. Любое целое число без знака единственным образом представляется с помощью чисел базы системы счисления и операций сложения - умножения в виде канонического арифметического выражения:

ЗНАЧЕНИЕв(цбз)=(dn-1*Bn-1+dn-2*Bn-2+...+d1*B1+d0*B0)

Последовательность коэффициентов канонического арифметического выражения также единственным образом характеризует величину числа и называется записью числа в B-ичной системе кодирования (счисления):

ЗАПИСЬв(цбз) = (dn-1dn-2...d1d0)

При этом, о коэффициенте di принято говорить как об i-м разряде числа, а об n - как о количестве разрядов числа. Диапазон чисел 0÷n-1 называется разрядной сеткой числа

2. Алгоритм перевода числа из системы счисления с основанием 10 в систему счисления с основанием B.

Перевод десятичной записи числа в B-ичную запись числа. Используются операции целочисленного деления: X mod Y - результатом является остаток от деления X на Y;
X div Y - результатом является целая часть от деления X на Y.
Вычисления выполняются в десятичной системе. В результате получается запись числа в системе с основанием B, записанная в обратном порядке.

Пример. Перевод (запись числа)$_{10}$ → (запись числа)$_2$

125	2						
-124	62	2					
1	-62	31	2				
	0	-30	15	2			
		1	-14	7	2		
			1	-6	3	2	
				1	-2	1	
					1		

В результате получаем: $[125]_{10} = [1111101]_2$

Пример. Перевод (запись числа)$_{10}$ → (запись числа)$_{16}$

125	16	
-112	7	
13		

В результате получаем: $[125]_{10} = [7D]_{16}$

3. Алгоритм перевода числа из системы счисления с основанием B в систему счисления с основанием 10.

1. Построить в системе с основанием B соответствующее каноническое выражение [e] в, представляя основание как 10;
2. Представляя элементы этого канонического выражения [e] в в десятичной системе, перевести его в неканоническое выражение (e) 10 десятичной системы;
3. Вычислить неканоническое выражение (e) 10и получить десятичную запись числа (d)10.
Пример.
Перевод (запись числа)8→(запись числа)10;
(d)8: запись числа в 8-й системе: $(125)_8$
[e]8: каноническое выражение числа: $[1*10^2+2*10^1+5*10^0]_8$
(e)10: выражение числа в 10-й системе: $(1*8^2+2*8^1+5*8^0)_{10}$
(d)10: запись числа в 10-й системе: $(85)_{10}$

4. Перевод чисел из одной системы счисления в другую, если основания систем кратны двум.

Перевод (запись числа)2→ (запись числа)8.
Двоичная запись числа разбивается справа - налево на триады (каждая по три разряда). Если в самой левой триаде меньше трех цифр – слева добавляются нули. Трехразрядное двоичное число каждой триады заменяется восьмеричной цифрой в соответствии с Таблицей
Перевод (запись числа)8→ (запись числа)2
Каждая цифра восьмеричной записи числа заменяется соответствующим двоичным трехразрядным числом.
Пример. (2753)8↔ (010.111.101.011)2
Алгоритм I.5. Перевод (запись числа)2→ (запись числа)16
Двоичная запись числа разбивается справа - налево на тетрады (каждая по четыре разряда). Если в самой левой тетраде меньше четырех цифр – слева добавляются нули. Четырехразрядное двоичное число каждой триады заменяется восьмеричной цифрой в соответствии с Таблицей
Алгоритм I.6. Перевод (запись числа)16 → (запись числа)2
Каждая цифра шестнадцатеричной записи числа заменяется соответствующим двоичным четырехразрядным числом.
Пример.
(5EB)16↔ (0101.1110.1011)2

3. Оперативная память

1. Понятие двоичного символа, двоичного слова и двоичного текста.

Двоичный символ - минимальная единица данных (бит, двоичная цифра, логическая константа), которая может храниться устройствами компьютера и передаваться между этими устройствами. Все остальные единицы данных образуются как совокупности двоичных символов. Двоичный символ можно рассматривать как двоичную переменную, которая имеет два допустимых значения: d∈{0,1}.

Двоичное n - разрядное слово определяется как последовательность двоичных символов длины n: w = dn-1,dn-2,...,di,...,d1,d0. (I.10) Каждый элемент di такой последовательности являет собой i-й разряд слова. Разряды нумеруются справа - налево (от младшего к стар-шему). Длина слова определяется как число составляющих его разрядов.

Двоичный Q словный текст мы определяем как последовательность длины Q , состоящую из n-разрядных двоичных слов: text = w0,w1,...,wj,...,wQ-2,wQ-1

2. Технические средства запоминания одного двоичного символа.

Носитель двоичного символа (триггер) - электронная схема, которая в каждый момент времени может находиться в одном из двух возможных состояний. Будем обозначать носитель двоичного символа - триггер как b. С каждым состоянием связано одно из двух допустимых значений {0,1}. То состояние, в котором находится триггер в рассматриваемый момент времени t, будем называть текущим состоянием и обозначать bt.

Будем также говорить, что если триггер находится в состоянии bt, то он хранит (помнит) двоичную цифру D(bt) = dt, связанную с этим состоянием.

Двоичным запоминающим элементом (ДЗЭ) мы называем триггер в совокупности с электронными схемами, реализующими операции чтения - записи двоичного символа. В каждый момент времени ДЗЭ находится в одном из двух возможных состояний. Операция записи бита в ДЗЭ - установка ДЗЭ в состояние, связанное с записываемым двоичным символом:

ДЗЭ := Линия записи

Не следует думать, что любой поступающий на линию записи двоичный символ запоминается на ДЗЭ. Для того чтобы его запомнить необходимо подать на управляющий вход (пр) единичный сигнал разрешения приема с линии записи. По сути дела, этот сигнал инициирует выполнение операции записи в ДЗЭ. Точно так же хранящейся в ДЗЭ двоичный символ выдается на линию чтения только при поступлении на управляющий вход (вд) единичного сигнала разрешения выдачи на линию чтения. По сути дела, этот сигнал инициирует выполнение операции чтения вДЗЭ

3. Технические средства запоминания одного двоичного слова.

Двоичный носитель слова (ДНС) мы определяем как последовательность, состоящую из n двоичных запоминающих элементов

Элемент bi этой последовательности называется i- м разрядом носителя слова.

Каждый разряд носителя слова в текущий момент времени находится в состоянии bti

и хранит двоичный символ D(bti), связанный с этимсостоянием. Последовательность:

btn-1,btn-2,...,bti,...,bt1,bto текущих состояний разрядов носителя слова мы называем текущим состоянием носителя слова.

Последовательность двоичных символов:

wt= D(btn-1),D(btn-2),...,D(bti),...,D(bt1),D(bto) ;

соответствующую текущему состоянию носителя слова, мы называем словом, хранящимся в носителе слова в текущий момент времени. Всего имеется 2^n

состояний n - разрядного носителя слова, с каждым из которых связано одно и только одно n - разрядное двоичное слово. Следовательно, n - разрядный носитель слова способен хранить 2^n различных n - разрядных двоичных слов.

Носитель слова в совокупности с реализованными для него операциями чтения - записи мы называем двоичный запоминающий регистр.

Носитель слова превращается в двоичный запоминающий регистр

r следующим образом :

• линии чтения объединяются в шину чтения ширины n;
• линии записи объединяются в шину записи ширины n;
• управляющие линии (вд) объединяются в общую управляющую
линию ВД;
• управляющие линии (пр) объединяются в общую управляющую
линию ПР.

Операция записи слова в запоминающий регистр r – установка носителя слова в состояние, представленное словом на шине записи:

Запись(r, Линия_записи)

Операция чтения слова из запоминающего регистра r – выдача на шину чтения слова, представленного текущим состоянием носителя слова:

Чтение(r, линия_чтения)

4. Технические средства запоминания множества двоичных слов.

Двоичный носитель текста (ДНТ) мы определяем как последовательность, состоящую из Q регистров. Каждый регистр носителя текста rj в текущий момент времени находится в состоянии rtj и хранит двоичное слово W(rtj), связанное сэтим состоянием. Последовательность: rt0,rt1,...,rtj,...,rtQ-2,rtQ-1, текущих состояний регистров носителя текста называется текущим состоянием носителя текста.

Последовательность двоичных слов:

Textt= W(rt0),W(rt1),,,,.W(rtj),...,W(rtQ-2),W(rtQ-1)

соответствующих текущему состоянию носителя текста, мы называем текстом, хранящимся в носителе текста в текущий момент времени. Носитель текста в совокупности с реализованными для него операциями чтения - записи мы называем двоичным запоминающим устройством или двоичной памятью (в дальнейшем, просто памятью).

По аналогии с предыдущим, следовало бы определить для носителя текста операции чтения-записи, обновляющие или читающие текст целиком. Однако технически сложно и дорого реализовать операцию записи, которая одновременно изменяет содержимое всей памяти, т.е. всю хранящуюся в ней последовательность слов. Точно так же можно сказать об операции чтения всего содержимого памяти.

Поэтому определяются операции чтения - записи в регистр памяти с указанным адресом. Причем, в качестве адреса регистра используют его порядковый номер в носителе текста. Последовательность целых чисел: 0,1,...,j,...,Q-2,Q-1 - называется адресным пространством памяти.

Операция записи слова в регистр памяти с заданным адресом: Запись(адрес, новое слово)

Операция чтения из регистра памяти с заданным адресом: Чтение(адрес)

Используя операции чтения - записи можно определить операцию обмена словами между регистрами памяти: Запись(адрес2,Чтение(адрес1))

Более привычно представление операции обмена данными в виде оператора присваивания:

адрес2 := адрес1 (I.19)

Регистр памяти, для которого реализуется запись - чтение называется доступным регистром, а его адрес – адресом доступа.

5. Понятие разрядной сетки и адресного пространства.

В качестве адреса регистра используют его порядковый номер в носителе текста. Последовательность целых чисел: 0,1,...,j,...,Q-2,Q-1 - называется адресным пространством памяти.

6. Понятие прямого доступа и равнодоступности.

Говорят, что память устроена как память прямого доступа в случае, когда в операциях явно указывается адрес регистра памяти. Память прямого доступа называется равнодоступной (памятью со случайным доступом) в случае, когда время доступа к регистру памяти не зависит от того, какой регистр был доступен перед этим. Равнодоступная память (память со случайным доступом), подключенная непосредственно к устройству выполнения операций обработки данных (центральному процессору), называется оперативной памятью (оперативным запоминающим устройством - ОЗУ). Ее англоязычное название - Random Access Memory (RAM).

По принципу действия и технической реализации RAM подразделяется на статическую SRAM, основой которой является триггер, и на динамическую память DRAM, основой которой служит конденсатор.

7. Оптимальность двоичной памяти.

С теоретической точки зрения, запоминающий регистр может строиться из n носителей символа, каждый из которых имеет B состояний равновесия и способен хранить любую из B цифр. В этом случае запоминающий регистр в целом может хранить n-разрядное B-ичное число. Однако память современных компьютеров строится на основе носителей двоичного символа. И для этого есть свои причины. Прежде всего, очевидно, что чем меньше число состояний у носителя символа, - тем надежнее он работает. Минимальное число состояний у носителя символа равно двум.

К счастью, критерий надежности не приходит в противоречие с критерием минимальной стоимости изготовления регистра. Существует разумное предположение, что стоимость изготовления регистра пропорциональна числу знаков регистра, которое определяется как произведение: z = B×n, где B- число состояний носителя знака, n-число разрядов регистра.

Пусть z = 20. Рассмотрим два крайних варианта: двухразрядный регистр из носителей знака с десятью состояниями равновесия и десяти разрядный регистр из носителей знака с двумя состояниями равновесия. В первом случае, регистр может хранить N =102 = 100 различных слов. Во втором случае: N = 210 = 1024 различных слова.

Оптимальным, с точки зрения стоимости изготовления, является нецелое основание e = 2.71... Среди целых оснований оптимум достигается при B=3.

8. Свойства оперативной памяти.

1. Основой оперативной памяти служит носитель текста (запоминающий массив) из 2^p запоминающих регистров - байтов памяти;

2. В каждый момент времени каждый байт памяти хранит байт данных (8 - разрядное двоичное слово);

3. В каждый момент времени память в целом хранит двоичный текст размером 2^p байт;

4. Оперативная память принимает p - разрядный двоичный код адреса с шины адреса и обеспечивает прямой доступ к каждому из 2^p регистров массива запоминающих регистров по заданному адресу;

5. Оперативная память является равнодоступной (время прямого доступа к байту памяти не зависит от предыстории);

6. Оперативная память обеспечивает выполнение операций чтения/записи для доступного регистра.

4. Типы данных

1. Необходимость введения понятия типа данных.

Минимальной адресуемой единицей памяти является байт. Однако, в общем случае, восьми двоичных разрядов недостаточно для размещения любой единицы данных. Поэтому мы будем говорить, что единица данных хранится в ячейке памяти, которая состоит из нескольких байт оперативной памяти. Адресом ячейки служит адрес первого байта, ее образующего.

Здесь мы рассматриваем только простые единицы данных, каждая из которых хранится и обрабатывается как единое целое. К числу простых единиц данных относятся числа и символы. В некоторых случаях в качестве простой единицы данных выступает строка символов.

Понятие типа данных является важнейшим понятием программирования. Тип данных - совокупность правил, определяющих некоторую категорию данных как:

- **множество допустимых для нее значений;**
- **множество допустимых операций над допустимыми значениями;**
- **способ хранения значения (формат ячейки хранения).**

Типизация данных необходима с нескольких точек зрения. При вводе единицы данных имеется возможность автоматически определить формат ячейки, необходимой для ее хранения.

Указание типа позволяет проконтролировать корректность применения операции к операндам. Например, сложение вещественного числа и символа не имеет смысла. Появляется также возможность использовать один символ для обозначения различных операций (перегрузка операций), например, операция сложения целых чисел и операция сложения вещественных чисел выполняются по различным алгоритмам, но обозначаются одним знаком «+». Исходя из описания типов операндов, можно автоматически распознать, какой алгоритм исполнения операции в данном конкретном случае применять. Более того, во многих случаях возможно автоматическое преобразование значений разнотипных операндов к одному формату.

2. Типизация целых чисел.

Очевидным образом вводится понятие алгебры целых чисел:

A<цч> = <Z;+,-,*,⊕,Ø,div,mod> (I.20)

Здесь, Z - основное множество алгебры; +,-,* - бинарные операции сложения, вычитания и умножения целых чисел. Существуют также две унарные операции изменения знака числа:

⊕ - унарный плюс и Ø - унарный минус

Позиционная система счисления (кодирования) целых чисел строится по аналогии с позиционной системой счисления целых чисел без знака. Каноническое арифметическое выражение имеет вид: ЗНАЧЕНИЕ в(целое) = ±(dn-1*Bn-1+dn-2*Bn-1+...+d1*B1+d0*B0)

Запись целого числа образуется очевидным образом: ±< dn-1, dn-2,…d1,d0 >

Для хранения двоичного кода целого числа используется ячейка памяти, в которой старший разряд представляет знак числа: 0 – знак «плюс». 1 - знак «минус». Ячейка памяти (размером один байт, два байта, четыре байта) имеет конечное число состояний. Отсюда следует, что имеется максимально представимое целое число (положительное) MAX и минимально представимое целое число (отрицательное) MIN.

Короткое целое, ячейка хранения размером байт: диапазон пред-

ставимости чисел Ω = -127÷+127, (один разряд для представления знака, 2^7=128 состояний для представления значения).

Целое, ячейка хранения размером два байта: диапазон представимости чисел Ω = -32768÷+32767(один разряд для представления знака, 2^{15} состояний для представления значения).

При использовании целых чисел без знака, переполнение разрядной сетки возможно при сложении, вычитании и умножении.

3. Специфика вычислений с данными целого типа.

Существует еще одно ограничение на использование целых чисел, связанное также с конечностью числа разрядов ячейки памяти. Дело в том, что каждое слагаемое может "убираться" в ячейку памяти, а их сумма уже не "убирается" в ячейку памяти. В этом случае, компьютер выдает сигнал "переполнение разрядной сетки" и прекращает дальнейшие вычисления.

4. Типизация символов и символьных строк.

Обозначение (код) символа на модельном уровне являет собой комбинацию графических линий. Число символов, используемых в модельном мире, конечно и они должны быть реализованы на клавиатуре компьютера в виде клавиш. На заре эры компьютеризации было принято волевое решение - число различных символов должно быть не более двухсот пятидесяти шести. Для представления любого из 2^8= 256 чисел необходима ячейка размером в 8 двоичных разрядов.Таким образом, появилась единица измерения данных и памяти – байт, т.е. слово длиной в восемь двоичных разрядов. Достаточно быстро выяснилось, что число различных символов, используемых для общения с компьютером, значительно больше двухсот пятидесяти шести. С целью увеличения числа комбинаций двоичных кодов, представляющих символы, были введены два дополнительных бита. Для реализации этих бит на клавиатуре компьютера были введены специальные клавиши. Сначала Ctrl, затем Alt - число комбинаций двоичных кодов символов возросло до 210. Но и этого оказалось недостаточно. В настоящее время для хранения символов может использоваться ячейка размером в несколько байт.

Соответствие символов и кодирующих их целых чисел без знака представляется в таблице кодирования. При создании такой таблицы учитываются соображения эффективности передачи кода по каналам связи и помехозащищенности его при такой передаче. С целью удовлетворения различных требований существуют различные таблицы кодирования. Такие таблицы разрабатывались, прежде всего, в расчете на латинский алфавит. Поэтому в любой из них соблюдается упорядоченность кодирования цифр и букв. Так, буква “A” кодируется меньшим числом, чем буква “B” и т.д. Это позволяет использовать арифметические операции для сравнения букв (цифр). Например, для определения того, является ли значение переменной Symbol буквой, достаточно вычислить значение отношения: “A”<= Symbol<=“Z’.

Назовем наиболее распространенные таблицы кодирования символов: ASCII /альтернативная/, CP 866 (упорядоченность русских букв, используется в DOS)

Таблица кодирования CP 125 также соблюдает упорядоченность кодирования русских букв, используется в операционной среде Windows.

Таблица кодирования КОИ-8 не соблюдает упорядоченность кодирования русских букв, используется в операционной среде UNICS. Таблица кодирования ISO соблюдает упорядоченность кодирования русских букв и является Госстандартом для оформления документации в Российские фонды алгоритмов и программ.

5. Типизация вещественных чисел.

Вещественное (действительное) число образуется, когда мы производим измерения какой либо числовой характеристики d объекта, используя для этого эталонную числовую величину d0.

В общем случае эталонная величина укладывается в измеряемой величине нецелое число раз. В результате такого измерения получается вещественное число как результат операции деления d\d0.

Очевидным образом вводится понятие алгебры вещественных чисел:

A<вч> = <R;+,-,*,/ >.

Позиционная система счисления (кодирования) вещественных чисел строится по аналогии с позиционной системой кодирования

целых чисел. Каноническое арифметическое выражение имеет вид:

ЗНАЧЕНИЕв(вещ) = ±(dp-1*Bp-1+dp-2*Bp-2+...+d1*B1+d0*B0+ d-1*B-1+d-2*B-2 +...)в.

Последовательность коэффициентов этого канонического выражения также однозначно обозначает вещественное число и называется записью вещественного числа.

ЗАПИСЬВ(вещ) = (dp-1dp-2...d1d0.d-1d-2...)B

Последовательность цифр до точки называется целой частью числа, последовательность за точкой называется дробной частью числа.

Вспомним, что множество целых чисел Z дискретно (образует счетное множество). Тогда как множество вещественных чисел R не дискретно (образует всюду плотное множество). И, в отличие от целых чисел, в любом, сколь угодно малом интервале оси R содержится бесконечное множество вещественных чисел.

Ячейка памяти, какой бы размер ее не выбрали, имеет конечное число разрядов и конечное число состояний. Но это означает, что не существует даже малого отрезка оси вещественных

чисел R, которому можно поставить в соответствие конечное множество состояний ячейки памяти.

Единственный выход из такой тупиковой ситуации – представлять состояниями ячейки памяти не числа из R, а отрезки числовой оси R Более подробно, каждый такой отрезок представляется «средним» n - разрядным рациональным числом. Множество таких чисел конечно и появляется возможность представить каждое из них состоянием n - разрядной ячейки памяти. Очевидно, что существует максимально представимое число и минимально представимое число в ячейке памяти с заданным числом разрядов. Аналогично предыдущему, определяется диапазон представимости вещественных чисел. Существует два различных формата представления вещественного числа в n - разрядной ячейке памяти.В формате с фиксированной точкой на представление целой и дробной части числа отводится соответственно p и q разрядов (Рис. I.20).

Формат с плавающей точкой (полулогарифмический формат) в той же самой разрядной сетке позволяет реализовать значительно больший диапазон представимости вещественных чисел.

Этот формат основан на следующем свойстве человеческого сознания: в записях очень больших числах (масса солнца 2·1030г.) и в записях очень маленьких числах (масса электрона 9·10-28г.) человеческий мозг запоминает лишь небольшое количество значащих (отличных от нуля) цифр. Величина же числа определяется количеством нулей. Это выражается в так называемой полулогарифмической записи десятичного числа:

<мантисса>10<порядок>

Здесь, мантисса - десятичное рациональное число с фиксированной запятой (значащие цифры), порядок- число нулей (величина числа). Строго говоря, представление числа в таком формате неоднозначно. Например: 200·101 = 20000·10-2 = 2·102= 0.2·103. Однако, если использовать нормализованную мантиссу, то это представление становится однозначным. Нормализованная мантисса сразу после точки содержит отличную от нуля цифру. Компьютерный формат вещественного числа с плавающей точкойстроится по тому же принципу: <нормализованная мантисса>16<порядок>.

Здесь, мантисса - двоичная запись вещественного числа в формате с плавающей точкой, порядок - двоичная запись количества нулей

В конкретном языке программирования обычно определяется несколько вещественных типов. Обычно, для хранения числа с плавающей точкой отводится ячейка размером в четыре байта: 8 бит для представления порядка и знака; 24 бита - для мантиссы.

6. Специфика вычислений с данными вещественного типа.

Основное отличие обработки чисел на компьютере по сравнению с «ручными» вычислениями - представление значений в ячейке с конечным числом разрядов. Относительно максимального представимого числа, минимального представимого числа и диапазона представимости целых чисел в n - разрядной ячейке памяти мы говорили выше. Эта же специфика сохраняется и при использовании вещественных чисел. Следствием способа представления всюду плотного множества вещественных чисел в ячейке с ограниченным числом разрядов является принципиальная приближенность вычислений. При этом возникают следующие побочные эффекты. Содержимое ячейки памяти представляет не одно значение, а диапазон значений вещественных чисел. Возникает проблема, когда не являющееся нулем число находится в окрестность нуля. В этом случае компьютер воспринимает его как "машинный нуль" со всеми вытекающими отсюда последствиями.В некоторых случаях происходит потеря точности вычислений. Приведем пример для четырехразрядной ячейки, хранящей десятичное число: (10.00/03.00)*03.00 = 09.99. В некоторых случаях результат вычислений зависит от порядка вычислений.

Пусть x=10, y=12,z=5. В абстрактной алгебре: (x*y)/z = x*(y\z)=24.

Используем по-прежнему четырехразрядную десятичную ячейку. При одном порядке вычислений x*y = 10.00*12/00 = 00.00 – переполнение разрядной сетки, дальнейшие вычисления невозможны.

При другом порядке вычислений получаем правильный результат.

y\z = 12.00\05.00 = 02.40; x*(y\z) = 10.00*02.40 = 24.00

7. Интерпретация хранящегося в памяти двоичного кода.

Данные различного типа в памяти вычислительной машины представляются двоичными кодами. Причем, если для единицы данных определенного типа формат хранения ее значения определяется однозначно, то этого нельзя сказать об интерпретации хранящегося в памяти двоичного кода.

Таким образом, важной проблемой является интерпретация (истолкование) двоичного кода, хранящегося в регистрах памяти. Наиболее очевидный способ - использовать для указания типа специальный байт (тэг), который является первым в ячейке памяти. Это позволяет при обращении по адресу к ячейке памяти проанализировать первый байт и определить тип хранящейся в этой ячейке единицы данных.

В качестве платы за простоту имеем расточительное расходование оперативной памяти (дополнительный байт на каждую ячейку). На первом этапе становления компьютеризации оперативная память была дорогая, поэтому конструкторы компьютеров и программного обеспечения пошли другим путем. Тип содержимого ячейки определяется по ее месторасположению в памяти. Так, если операция целочисленного сложения использует ячейки с адресами α, β и γ, то в этих ячейках должны храниться данные целого типа. Таким образом, возникает задача распределения оперативной памяти, т.е. «нарезка» ее на области, в которых хранятся данные одинаковых типов.

При программировании на уровне машинных кодов или языка ассемблера распределение памяти осуществляется «вручную». При этом программист определяет для себя: какие участки памяти образуют какие ячейки; тип значения в каждой ячейке. Записывая затем программу, программист следит за корректностью использования типов данных в операциях их обработки.

При таком программировании средств автоматического контроля не существует, что приводит к многочисленным ошибкам в программе, которые вызываются несоответствием типов единиц данных.

Языки программирования высокого уровня подразумевают автоматическое распределение памяти под переменные и константы, на основании спецификации (описания) их типов. Компилятор при первом проходе составляет таблицу распределения памяти, и при втором проходе создает для каждого оператора программы соответствующий машинный код.

5. Сложение и вычитание целых неотрицательных чисел

1. Алгоритм сложения целых неотрицательных чисел (вычисления в десятичной системе).

Сложение B-ичных целых беззнака (вычисления производятся в десятичной системе)

Рассмотрим сложение двух чисел, представленных в B-ичной системе счисления. Пусть Z=X+Y, где первое слагаемое X, второе слагаемое Y и сумма Z являются целыми неотрицательными числами без знака.

Представим i-й разряд при сложении в следующем виде:

wi+1←xi← wi

yi

zi

Здесь, wi перенос из предыдущего разряда в разряд i, wi+1 - перенос из i-го разряда в следующий разряд. Очевидно, что w0=0.

Преобразуем оба слагаемых в B-ично десятичную систему (цифры B-ичного числа представляются десятичными числами). Для i =0...n-1 производим следующие действия в десятичной системе.

1. Вычисляем вспомогательную сумму ci: ci := (xi + yi) + wi

2. Вычисляем цифру разряда и перенос в следующий разряд.

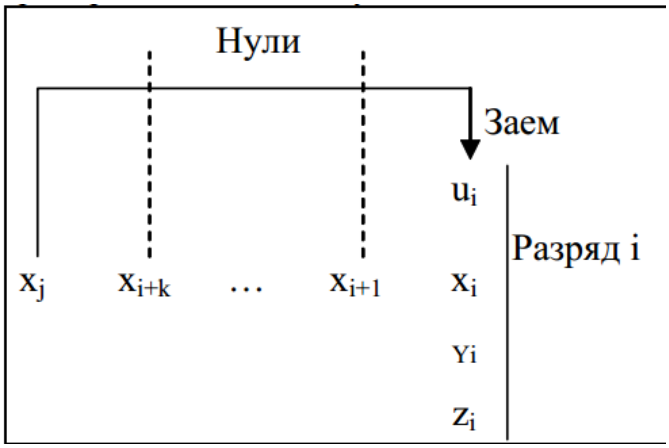
ЕСЛИ (ci < B) ТО zi:= ci; wi+1 := 0 ИНАЧЕ zi := ci - B; wi+1 := 1

3. Результат вычислений преобразуем в систему с основанием B. (Де-

сятичное число заменяем на соответствующую B-ичную цифру).

2. Алгоритм вычитания целых неотрицательных чисел (вычисления в десятичной системе).

Рассмотрим вычитание двух чисел, представленных в B-ичной системе счисления. Пусть Z=X-Y, где уменьшаемое X, вычитаемое Y и разность Z являются целыми неотрицательными числами беззнака. Представим i-й разряд при сложении в следующем виде:



Здесь, xi, yi, zi разряд уменьшаемого, вычитаемого и разности; xi+1... xi+k - разряды с нулевыми значениями; xj - первый, после нулевых, разряд с ненулевым значением; ui - значение займа.

Преобразуем уменьшаемое и вычитаемое в B-ично десятичную систему. Устанавливаем нулевые значения займов для всех разрядов: ui = 0.

Для i = 0...n-1 производим следующие действия в десятичной системе. Вычисляем вспомогательную разность ci: ci = xi - yi

Вычисляем заем (если он есть) из предыдущего разряда.

ЕСЛИ(ci<0) ТО xj := xj-1; xi+k := B-1; ...; xi+1 := B-1; ui:= B.

Вычисляем значение разности для разряда: zi:= ui + xi - yi.

Результат вычислений преобразуем в систему с основанием B.

3. Почему эти алгоритмы не являются машинными алгоритмами?

Приведенный выше алгоритм не является машинным алгоритмом, т.к. не может исполняться без вмешательства человека. Дело в том, что онпредполагает знание и выполнения “в уме” алгоритма разрядного сложения и, более того, алгоритма вычитания B-ичных чисел. Имеется возможность автоматизировать эти вычисления за счет использования таблицы сложения и заменить вычисления "в уме" формальной процедурой поиска в таблице сложения (вычитания).

4. Алгоритм сложения целых неотрицательных чисел, использующий таблицы сложения.

При вычислении разрядной суммы Ci используется B-ичная таблица сложения: реализуется поиск ячейки таблицы по заданным xi и yi и, в случае наличия единицы переноса из младшего разряда – спуск на одну ячейку вниз. Такой алгоритм является машинным, т.к. может исполняться без вмешательства человека.

2	0	1
0	00	01
1	01	10
10	10	11

6. Автоматическое исполнение команды обработки данных

1. Каким образом алгоритмы преобразования символьных строк обеспечивают исполнение арифметических операций?

Рассмотрим для примера операцию сложения двух целых чисел.
Здесь, a и b - абстрактные числа.
Дано:
• запись (последовательность цифр) первого слагаемого: Запись(a);
• запись (последовательность цифр) второго слагаемого: Запись(b).
• Выполнить преобразование записи двух слагаемых в запись результата сложения, обеспечивающее равенство:
Запись(a)+ Запись(b) = Запись (a "сложить" b).
Здесь "сложить" - абстрактная операция сложения целых чисел, + - операция преобразования записей, реализующая сложение целых чисел. Хорошо известен алгоритм сложения «столбиком» чисел в десятичной системе счисления. На его примере мы видим, что сложение сводится к преобразованию двух символьных строк в результирующую символьную строку. Аналогично для других систем счисления. Более подробно. Первое слагаемое (абстрактное число) представляется в позиционной системе счисления каноническим выражением:
E(a) = an-1Bn-1+an-2Bn-2+ ... +a1B1+a0B0
Точно также, второе слагаемое (абстрактное число) представляется каноническим выражением:
E(b) = bm-1Bm-1+bm-2Bm-2 ... +b1B1+b0B0
Алгоритм сложения должен получить каноническое выражение
суммы абстрактных чисел:
E(a ⊕ b) = cP-1BP-1+cP-2BP-2+ ... +c1B1+c0B0
Аналогично, для остальных арифметических операций:
• Запись(a) - Запись(b) = Запись (a "минус" b).
• Запись(a) * Запись(b) = Запись (a "умножить" b).
• Запись(a) / Запись(b) = Запись (a "разделить" b).
Здесь, кавычками обозначены абстрактные операции над числами.

2. Команды обработки данных.

Подводя итог предыдущему обсуждению, скажем, что арифметико-логическое устройство (АЛУ) реализует исполнение команд обработки (преобразования) данных. Они подразделяются на четыре класса.
1. Арифметические и логические команды формата: КОП A1,A2,A3 -выполняют операции сложения и вычитания по следующему алгоритму:
• выбрать из оперативной памяти содержимое ячейки A1- значение первого аргумента;
• выбрать из оперативной памяти содержимое ячейки A2 – значение второго аргумента;
• произвести определяемую КОП (код операции) операцию над аргументами;
• записать результат в оперативную память по адресу A3.
2. Команда пересылки данных формата: КОП A1,0,A3.
3. Команды сдвига кода в регистре и команда обращения кода в регистре.
4. Команда ввода формата: ВВОД 0,0,A3. Код нажатой на клавиатуре клавиши вводится в байт оперативной памяти с адресом A3.
5. Команда вывода формата: ВЫВОД A1,0,0. Символ, код которого хранится в байте оперативной памяти, выводится на терминал.

3. Исполнение команды сложения.

Мы знаем, что оперативная память являет собой последовательность байт. Но каждый байт - это последовательность восьми бит. Так что в конечном итоге, содержимое оперативной памяти представляется в виде последовательности бит (двоичной последовательности). Будем называть такую последовательность двоичным вектором памяти.
Предположим, что записи - операнды сложения, хранятся в ячейках a, b оперативной памяти, запись - результат сложения хранится в ячейке c.

Не вводя дополнительных предположений, мы можем лишь утверждать, что каждый i-й разряд результата сложения определяется всеми разрядами слагаемых. Таким образом, операцию сложения можно представить в виде множества функций:

c0 = F+0(an-1,...,ai,...,a0,bn-1,...bi,...,b0)

…

ci = F+i (an-1,...,ai,...,a0,bn-1,...bi,...,b0)

...

cn-1 = F+n-1 (an-1,...,ai,...,a0,bn-1,...bi,...,b0)

Таким образом, для получения i - го разряда результата использу-ется функция:

ci= F+i (an-1,...,ai,...,a0,bn-1,...bi,...,b0)

Аргументы функции - суть логические переменные. Значение функции - также логическое. Таким образом, значение разряда результата является значением логической (булевской) функции.

Заметим, что i-й разряд результата определяется не всеми разрядами слагаемых, а лишь младшими разрядами, от нулевого до i-го:

Для учета значений разрядов от 0-го до i -1-го вводится специфическая переменная wi- перенос в i-й разряд. Таким образом, сложение в одном разряде описывается логической функцией:

ci = Fi (wi,ai,bi).

Аналогично, перенос в следующий разряд описывается логической функцией:

wi+1 = Wi+1(wi,ai,bi)

Теперь можно конструировать электронную схему, вычисляющую значение i-го разряда суммы.

Запишем таблицу значений (график) функции сложения Fi и таблицу значений (график) функции переноса Wi+1:

w_i	a_i	b_i	c_i
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

w_i	a_i	b_i	w_{i+1}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

В математической логике доказана теорема: любая логическая функция может быть записана в виде композиции трех элементарных логических функций: отрицания, дизъюнкции и конъюнкции.

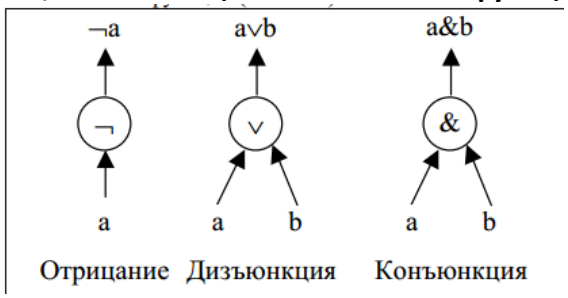
Если имеется табличное задание логической функции, можно получить ее представление в виде композиции отрицания, дизъюнкции и конъюнкции.

ci =(¬wi&¬ai&bi_)∨ (¬wi&ai&¬bi)∨ (wi&¬ai&¬bi)∨(wi&ai&bi)

Проделав аналогичные действия для таблицы переносов, получаем ДНФ функции переноса:

wi+1 =(¬pi&ai&bi)∨ (pi&¬ai&bi)∨(pi&ai&¬bi)∨(pi&ai&bi)

Названные выше три элементарных логических функции замечательны еще тем, что существуют простые электронные элементы, вычисляющие значения этих функций.



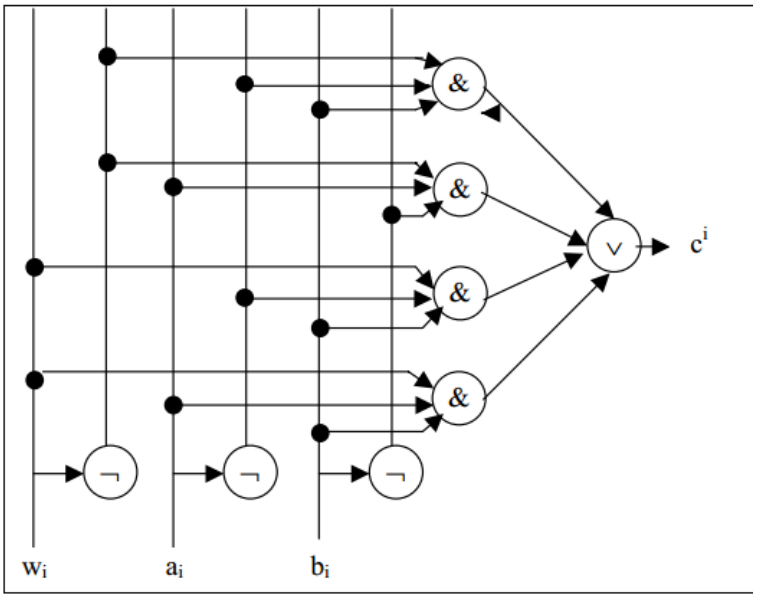


Рис. I.30. Функциональная схема сложения

4. Арифметико-логическое устройство и его функционирование.

Мы будем говорить, что электронная схема, реализующая вычисление логической функции, является физической моделью логической функции. При этом собственно электронная схема задает функцию (физический аналог таблицы функции), а функционирование схемы - реализует процесс вычисления функции.

Заметим, что в операции сложения используется работа с битами. Но минимальной адресуемой единицей памяти является регистр памяти (ячейка памяти). Следовательно, возможности адресовать каждый бит памяти у нас нет.

Для решения этой проблемы используется.

Для реализации автоматического исполнения операций конструируется арифметико-логическое устройство(АЛУ), в состав которого входят три регистра: R1 - хранение первого операнда, R2 - хранение второго операнда, R3 - хранение результата.И прежде чем автоматически исполнять операцию, ее операнды выбираются из оперативной памяти и пересылаются на регистры R1,

R2. После выполнения операции ее результат с регистра R3 пересылается в оперативную память.

При этом между АЛУ и оперативной памятью передаются сразу все разряды регистра памяти, для чего требуется лишь адресация ячейки оперативной памяти. А адресация каждого бита необходима лишь в АЛУ.

Таким образом, физические модели функции сложения и функции переноса в качестве своих входов и выходов используют не значения битов оперативной памяти, а значения разрядов регистров АЛУ.

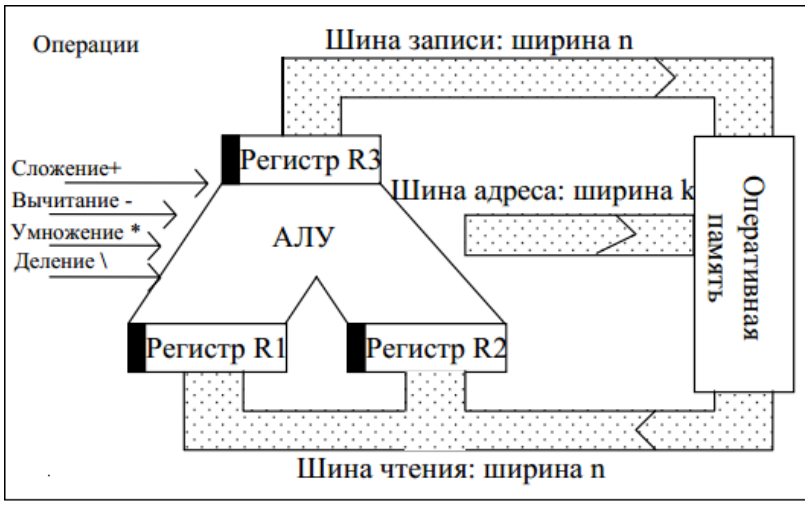


Рис. I.31. Подключение арифметико-логического устройства к оперативной памяти

Объединение модели функции сложения и модели функции переноса образует конструктивный элемент АЛУ - функциональную схему одноразрядного сумматора .

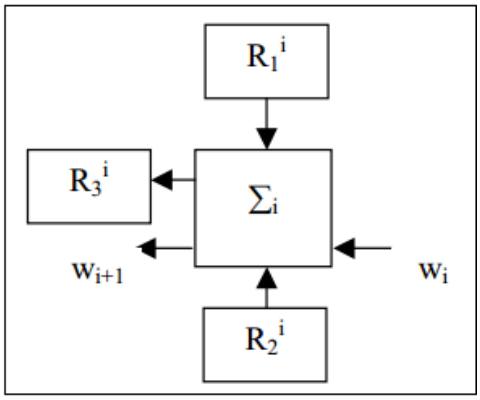


Рис. I.32. Одноразрядный сумматор Σ_i

Последовательность из n одноразрядных сумматоров, образует полный сумматор .

Специфика связи АЛУ - оперативная память приводит к необходимости введения понятия команды обработки данных, в полях которой указываются: операция, адреса операндов, адрес результата. В данном случае такой командой является команда сложения:

КОП+: А1 ,А2 ,А3. (I.37)

Здесь, КОП+ - код операции сложения, А1 - адрес в оперативной памяти первого операнда , А2 - адрес в оперативной памяти второго операнда, А3 - адрес в оперативной памяти результата.

5. Сложение и вычитание целых чисел.

При сложении чисел разных знаков приходится использовать операцию вычитания. Следуя рассмотренной выше методике, можно построить электронную схему, осуществляющую вычитание двух целых положительных чисел. Но это приведет к существенному увеличению сложности АЛУ. Если использовать специальные коды для представления отрицательного числа, этого усложнения можно избежать. Более того, использование этих кодов позволяет не только реализовать сложение чисел произвольных знаков, но также просто реализовать операцию вычитания чисел произвольных знаков.

Прямой, обратный и дополнительный коды числа определяются следующим образом. Здесь и далее:

S - код знака числа, N - код величины числа.

Положительное число.

(дополнительный и обратный код совпадают с прямым кодом).

S(прямой код) = S(обратный код) = S(дополнительный код) = 0;

N(прямой код) = N(обратный код) = N(дополнительный код) = двоичный код абсолютной величины числа.

Отрицательное число.

S(прямой код) = S(обратный код) = S(дополнительный код) = 1;

N(прямой код) = двоичный код абсолютной величины числа;

N(обратный код) = Инверсия от N(прямой код);

N(дополнительный код) = N(обратный код) + 1.

Таблица I.19. Коды положительного и отрицательного числа

Положит. число		Отрицат. число		
S	N	S	N	
+	5	-	5	Число
0	101	1	101	Прямой код
0	101	1	010	Обратный код
0	101	1	011	Дополнительный код

Таблица I.20. Сложение - чисел в дополнительном коде

R1	R2	
+	+	R3:=R1+R2;
-	+	C:=Доп(R1)+R2; Если Знак(C)=1 то R3:=Пр(C) иначе R3:=C
+	-	C:=R1+Доп(R2); Если Знак(C)=1 то R3:=Пр(C) иначе R3:=C
-	-	C:=Доп(R1)+Доп(R2); R3:=Пр (C)

Примечание. В оперативной памяти числа хранятся в прямом коде.

6. Арифметическое умножение и деление.

Арифметические команды обеспечивают автоматическое выполнение арифметических операций над данными, хранящимися в оперативной памяти. Схемная реализация операций сложения и вычитания рассмотрена выше.Операции умножения и деления целых чисел выполняются по обычным алгоритмам - "столбиком". Для их реализации АЛУ должно обеспечивать, наряду с операциями сложения - вычитания, также операции сдвига двоичного кода влево и вправо. Также можно представить процесс выполнения операций арифметики над вещественными числами (формат с плавающей точкой). При этом выполняется выравнивание порядков, затем производятся операции над мантиссами и, в заключение, определяется значение порядка результата. Арифметические команды позволяют вычислять значения элементарных вещественных функций многих вещественных аргументов(формализация физических задач).

7. Логические команды.

Однако не меньшее значение при вычислениях на компьютере имеют логические команды, позволяющие решать проблемы принятия решений.

Задачи принятия решений, явно или неявно, сводятся к вычислению логического выражения. Логическое выражение представляет собой комбинацию трех элементарных логических функций (отрицание, дизъюнкция, конъюнкция) и предикатов – отношений (равно, не равно, больше, меньше, больше или равно, меньше или равно). Предикат – отношение определяется следующим образом: <арифметическое выражение> <знак отношения> <арифметическое выражение>. Он имеет два возможных значения: истина, ложь – и связывает "мир"формальной логики с "миром" вещественных функций. Логические выражения используются в качестве условий для операторов условного перехода и операторов цикла.

Для программирования логических вычислений используются три команды: инверсия, логическое сложение и логическое умножение.

Общепринято, что в качестве операнда таких команд выступает не отдельный бит памяти, а слово памяти. Команды отношения для двух слов (сравнение, больше, меньше и т.д.) определяются очевидным образом. Тогда как логические команды определяются специфически. Команда инверсии (поразрядное отрицание) во всех битах слова заменяет значение на противоположное.

Команда логического сложения (поразрядная дизъюнкция) выполняется по следующему алгоритму: Ci = aiVbi

Команда логического умножения (поразрядной конъюнкции) выполняется по следующему алгоритму: Ci= ai&bi

8. Автоматическое исполнение команды обработки данных.

Для хранения текущей исполняемой команды используется регистр команд (РК). Учитывая формат команды, он подразделяется на следующие поля: код операции КОП; А1, А2 - адреса ячеек оперативной памяти, в которых хранятся операнды операции; А3 - адрес ячейки оперативной памяти, куда следует записать результат исполнения операции. Как сказано выше, АЛУ соединена с оперативной памятью шинами передачи данных (шина записи и шина чтения). Если n - ширина шины передачи данных, то по ней одновременно передаются n бит данных. Адресные поля регистра команд соединены с оперативной памятью шиной адреса. Если ширина шины адреса p, то размер адресного пространства 2^p.

Т.е. имеется возможность адресации 2^p байт оперативной памяти. Реализация автоматического исполнения находящейся на РК команды осуществляется Устройством Управления Автоматическим Выполнением Команды (УАВК). Это устройство для каждой команды вырабатывает специфическую последовательность электронных импульсов (сигналов), которые принято называть микрооперациями. Каждая микрооперация заставляет работать вполне определенный электронный блок компьютера.

Функционально микрооперации объединяются в микрокоманды.

Таким образом, каждая команда обработки данных представляется как последовательность микрокоманд, а каждая микрокоманда - как последовательность микроопераций.

Существует изображен дешифратор кода операции (ДШКОП). Код операции команды, находящейся на регистре команд, вызывает появление единичного сигнала на одном и только одном его выходе. И сигнал на этом выходе предписывает УУАВК выполнить вполне определенную последовательность микроопераций, т.е. вполне определенную команду обработки данных.

7. Автоматическое исполнение программы

1. Команды обработки данных и команды управления.

При конструировании алгоритма предполагается, что существует исполнитель, способный выполнять конечное множество элементарных действий по преобразованию данных. Исполнителем может быть как человек, так и техническое устройство. Обозначение элементарного действия по преобразованию данных называется командой преобразования данных исполнителя. Спецификой алгоритма является представление сложного преобразования любого допустимого исходного набора данных d в результирующий набор данных r в виде последовательности элементарных действий – команд исполнителя. Такую последовательность σ(d) будем называть траекторией вычислительного процесса решения прикладной задачи.

Вспомним, чтопрограмма - последовательность команд. Каждая команда имеет формат:

КОП A1,A2,A3.

Множество команд (обработки данных и управления) образует систему команд компьютера.

Система команд существенным образом определяет тип (архитектуру) вычислительной машины. Однако, вне зависимости от конкретной архитектуры, команды вычислительной машины подразделяются на два класса: команды обработки данных и команды управления выполнением вычислительным процессом. К командам обработки данных относятся:

- **команды преобразования данных (пересылка, арифметические и логические команды).**
- **команды сдвига кода;**
- **команды ввода данных в оперативную память;**
- **команды вывода данных из оперативной памяти;**

Команды управления реализуют безусловную и условную передачу управления, а также обращение к подпрограммам и возврат из подпрограмм.

Команды обработки данных можно классифицировать по числу адресов; наиболее часто используются одноадресные и двухадресные команды. Команды управления, как правило, одноадресные.

2. Понятие траектории вычислительного процесса.

Фундаментальным понятием информатики является понятие алгоритма. Алгоритм решения прикладной задачи однозначно определяет способ преобразования исходных данных в результирующие данные. По сути дела, решение прикладной задачи и заключается в таком преобразовании данных.

При конструировании алгоритма предполагается, что существует исполнитель, способный выполнять конечное множество элементарных действий по преобразованию данных. Исполнителем может быть как человек, так и техническое устройство. Обозначение элементарного действия по преобразованию данных называется командой преобразования данных исполнителя. Спецификой алгоритма является представление сложного преобразования любого допустимого исходного набора данных d в результирующий набор данных r в виде последовательности элементарных действий – команд исполнителя. Такую последовательность σ(d) будем называть траекторией вычислительного процесса решения прикладной задачи.

3. Понятие линейной программы и программы.

Описание траектории вычислительного процесса на формальном языке программирования и называется линейной программой. Очевидно, что линейная программа состоит только из команд обработки данных(арифметические, логические, ввод, вывод) и заканчивается специальной командой останова.

Программа – одна из возможных моделей вычислительного процесса. Она представляет собой линейный текст (последовательность команд), записанный в соответствии с правилами формального языка программирования.

4. Принцип программного управления.

Следует сказать, что понятие программы не является следствием изобретения компьютеров. Человечество программировало вычисления задолго до появления подобных вычислительных машин. Исполнителем таких программ являлся человек, оснащенный, в лучшем случае, средством автоматического исполнения арифметических операций (арифмометр, счеты, логарифмическая линейка). Появившийся задолго до изобретения компьютеров арифмометр также способен автоматизировать только исполнение арифметических операций.

Настоящая революция в области автоматизации вычислений произошла после того, как американский ученый фон Нейман сформулировал свои принципы построения автоматической вычислительной машины.

Принцип программного управления фон-Неймана формулируется в виде следующих пунктов.

1. Программа преобразования данных, представленная в двоичном виде, размещается в той же оперативной памяти, что и данные;
2. Электронная схема устройства управления реализует алгоритм главного цикла компьютера, который обеспечивает последовательную автоматическую выборку из оперативной памяти команд программы и их исполнение.

Вспомним, что программа - последовательность команд. Каждая команда имеет формат:

КОП A1,A2,A3.

Здесь, КОП - код операции, A1, A2, A3 - адреса ячеек оперативной памяти (p- разрядные двоичные последовательности). Каждый код операции также можно представить r- разрядной двоичной последовательностью. Таким образом, для представления команды используется r+3×p двоичных разрядов. Как правило, команда "не убирается" в один байт и размещается в ячейке размером в несколько байт. Адрес команды определяется как адрес первого байта хранящей эту команду ячейки. Длина команды - число занимаемых командой байт. Таким образом, в оперативной памяти образуются три области:

1. Область хранения программы;
2. Область хранения данных;
3. Свободная область.

Команды линейной программы должны выполняются последовательно, начиная с первой команды. Необходимо физически реализовать отношение следования команд линейной программы. Т.е. необходимо решить проблему указания того, какая команда должна исполняться после выполнения текущей команды. Здесь возможны два способа такого указания. Во - первых, можно добавить еще одно (четвертое) поле адреса Аслк, в котором указывается следующая исполняемая команда:

КОП A1,A2,A3,Аслк.

В этом случае мы имеем дело с четырехадресной командой, а линейная программа образует связный список. Образно говоря, программа становится похожей на нитку бус, разбросанных по памяти, где в качестве бусин выступают команды программы.

Преимуществом такого способа хранения программы является возможность "разбрасывания" команд по памяти произвольным образом.

А платой за такую произвольность является большой расход памяти, необходимой для хранения адресов следующих команд.

Если принять достаточно естественное условие, что следующая команда линейной программы хранится в следующей ячейке оперативной памяти, то можно обойтись трехадресной командой. При этом достаточно иметь лишь один регистр - счетчик команд (СЧК) - на котором хранится адрес следующей исполняемой команды. Очевидно, что для перехода к следующей команде счетчик команд необходимо увеличить на длину команды в байтах (Δ).

5. Команда, микрокоманда и микрооперация.

При исполнении команды обработки данных, устройство управления автоматическим выполнением команд (УУАВК) генерирует последовательность микроопераций, которые объединяются в микрокоманды

6. Главный цикл программно - управляемой вычислительной машины.

Алгоритм главного цикла вычислительной машины реализуется посредством генерации повторяющейся последовательности микроко-

манд. Как и прежде, микрокоманда реализуется в виде последовательности микроопераций.

Таблица I.26. Реализация алгоритма главного цикла компьютера

	Микрокоманда	Микрооперация	Комментарий
1	СЧК := адрес первой команды.	ВДА2	Выдача адреса первой команды на шину адреса
		ПРСЧК	Прием адреса первой команды на счетчик команд
2	РК := ПАМ(СЧК)	ВДСЧК	Выдача адреса следующей команды на шину адреса
		ЧТ	Инициация чтения из оперативной памяти
		ПРРК	Прием считанной из оперативной памяти команды на РК
3	СЧК := СЧК+Δ	+ΔСЧК	Подготовка к чтению следующей команды
4	ВДКОП	ВДКОП	Расшифровка кода операции
5	ЕСЛИ КОП(РК) = "ОСТАНОВ"	останов	Окончание исполнения программы
6	запуск УУАВК	запуск команды	Автоматическое исполнение команды
7	повторение 2	пуск	Повторение главного цикла

7. Устройство автоматического исполнения программы.

Вычислительная машина с трехадресной системой команд исполняет линейную программу (состоящую только из команд обработки данных), последовательно выбирая команды из памяти и передавая их на регистр команд для автоматического исполнения. Таким образом, после исполнения очередной команды начинает исполняться следующая команда.

Архитектура такой трехадресной, программно - управляемой вычислительной машины приведена в Приложении 2. На рисунке показано,
что шина чтения подходит к регистру команд, а счетчик команд соединен
с шиной адреса.

При исполнении команды обработки данных, устройство управления автоматическим выполнением команд (УУАВК) генерирует последовательность микроопераций, которые объединяются в микрокоманды. Исполнение линейной программы обеспечивается Устройством Управления Автоматическим Выполнением Программы (УУАВП), которое реализует алгоритм главного цикла вычислительной машины. Алгоритм главного цикла вычислительной машины реализуется посредством генерации повторяющейся последовательности микрокоманд. Как и прежде, микрокоманда реализуется в виде последовательности микроопераций.

Особого рассмотрения заслуживает вопрос присвоения перед на-
чалом исполнения счетчику команд адреса первой исполняемой команды
программы. Мы предполагаем, что этот адрес заносится непосредственно
на регистр команд (РК) с пульта - управления вычислительной машиной.
И после этого нажимается кнопка "Пуск", вызывающая запуск УУАВП

8. Принципы фон – Неймана.

Принципы фон - Неймана в сжатом виде формулируются следующим образом:

1. Информация кодируется в двоичной форме и представляется в виде наборов данных. Набор данных разделяется на единицы данных, называемые словами.
2. Разнотипные слова различаются по способу использования, но не по способу кодирования.
3. Слова размещаются в ячейках памяти машины и идентифицируются номерами ячеек, называемыми адресами ячеек (адресами слов).
4. Алгоритм представляется в форме последовательности управляющих слов, которые определяют наименование операции и слова, участвующие в операции. Каждое такое управляющее слово называется командой. Алгоритм, представленный в виде последовательности машинных команд, называется программой.
5. Выполнение преобразования данных, предписанных алгоритмом, сводится к последовательному выполнению команд в порядке, однозначно определяемом программой и конкретным набором исходных данных.

Совершенствование архитектуры вычислительной машины – компьютера

1.Организация управления компьютером

1.Команда, микрокоманда и микрооперация.

Рассматривая проблемы реализации автоматического исполнения команды и автоматического исполнения программы, мы остановились на следующей терминологии:

- **программа →последовательность команд;**
- **команда → последовательность микрокоманд;**
- **микрокоманда → последовательность микроопераций.**
- **микрооперация → двоичный электронный сигнал, включающий**

или выключающий вполне определенную подсхему в электронной схеме компьютера.

Таким образом, для того, чтобы выполнить программу, необходимо сгенерировать последовательность микроопераций, обеспечивающую реализацию алгоритма главного цикла

Аналогично, для того, чтобы автоматически выполнить команду, необходимо сгенерировать последовательность микрокоманд, реализующую алгоритм ее исполнения.

2. Электронная реализация управления.

Каждый алгоритм (в том числе алгоритм главного цикла и алгоритм выполнения команды) может иметь или схемотехническую или программную реализацию. В первом случае конструируется электронная схема, функционирование которой обеспечивает исполнение алгоритма.

Во втором случае алгоритм реализуется в виде программы для некоторой

программно - управляемой вычислительной машины.

При схемотехнической реализации для каждого конкретного алгоритма необходимо конструировать уникальную схему. Это дорого, но обеспечивает минимальное время исполнения алгоритма. При программной реализации аппаратура вычислительной машины создается один раз, а для реализации конкретного алгоритма конструируется соответствующая программа. Но время исполнения алгоритма становится больше за счет многошагового исполнения алгоритма.

Имеется генератор тактовых импульсов, который задает повторяющуюся дискретную последовательность: t1,t2,...tk,t1,t2...tk…

Аппаратура, реализующая алгоритм главного цикла, после нажатия кнопки "Пуск" генерирует последовательность микроопераций:мо1, мо2, мо3, мо4, мо5, мо6, мо7, мо8, мо9 мо4, мо5, мо6, мо7, мо8, мо9

Микрооперация мо7 устанавливает счетчик команд на следующую команду и одновременно инициирует выдачу кода операции с регистра команд РК на дешифратор команды. Если на регистре команд находится команда останова, то на одном из выходов дешифратора команд

появляется единичный сигнал, играющий роль микрооперации мо8 (прекращение выполнения главного цикла, т.е. останов компьютера).

В противном случае, микрооперация мо9 запускает устройство автоматического исполнения команды. Это устройство генерирует ту или иную последовательность микроопераций, обеспечивающую исполнение команды, находящейся на регистре команд.

И так продолжается до тех пор, пока в программе не встретится команда останова, которая породит микрооперацию мо8 (то же, что и кнопка "Останов").

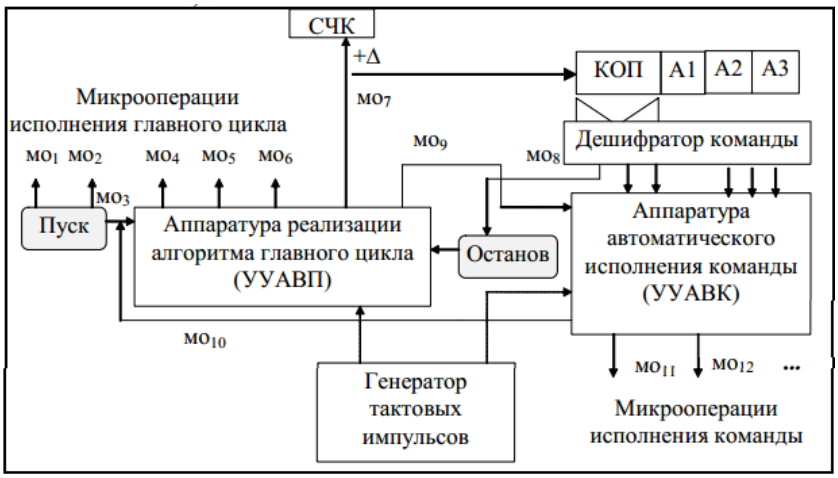


Рис. II.1. Схемотехническая реализация устройства управления компьютером

3. Понятие микропрограммы.

Микропрограммирование основано на представлении команды в виде последовательности микрокоманд, каждая из которых при своем выполнении генерирует последовательность микроопераций.

Каждой команде компьютера поставлена в соответствие микропрограмма, состоящая из последовательности микрокоманд, способных сгенерировать необходимую для исполнения команды последовательность микроопераций.

4. Микропрограммное управление.

Устройство микропрограммного управления (МПУ) представляет собой встроенный в основной компьютер специализированный управляющий компьютер, в качестве оперативной памяти использующий постоянное запоминающее устройство (ПЗУ). В ПЗУ хранятся микропрограммы, соответствующие командам основного компьютера.По сути дела, исполнение команды основного компьютера реализуется посредством исполнения соответствующей ей микропрограммы устройством микропрограммного управления. При выполнении микропрограммы генерируется последовательность микроопераций, необходимая для исполнения команды основного компьютера. Алгоритм главного цикла вычислительной машины также представляется в виде микропрограммы.

Все микрокоманды имеют одинаковый формат:

- **Адрес следующей микрокоманды**
- **Упраляющие биты микроопераций**

Число управляющих бит в микрокоманде равно числу всех микроопераций, необходимых для реализации управления. Напомним, что микрооперация - это двоичный сигнал, осуществляющий включение - выключение вполне определенного элемента электронной схемы. Если некоторый управляющий бит имеет значение "1", то это означает генерацию соответствующей микрооперации при выполнении микрокоманды.

Таким образом, микрокоманды отличаются друг от друга количеством управляющих бит, в которых находятся единицы.

Особенностью микрокоманды является наличие в ней поля адреса следующей микрокоманды. Таким образом, микропрограмма представляет собой связный список микрокоманд.

Последовательность микрокоманд назовем микропрограммой главного цикла.

Пусть находящаяся на регистре микрокоманд (РМК) микрокоманда имеет единицы в s управляющих битах. Тогда эта микрокоманда выполняется за (s+1) такт. Первые s тактов порождают микрооперации, соответствующие единичным управляющим битам, а последний такт обеспечивает прием на регистр микрокоманд РМК следующей микрокоманды (с адресом АМК).

Мы по-прежнему считаем, что команда компьютера является трехадресной. Только вместо поля кода операции (КОП) используется поле адреса начала соответствующей микропрограммы в ПЗУ

Естественно, что программу составляет программист, а микропрограмма создается конструктором на стадии проектирования компьютера.

Множество микропрограмм (представляющих систему команд компьютера) "зашивается" в ПЗУ при изготовлении компьютера, и это множество одинаково для всех машин одного типа.

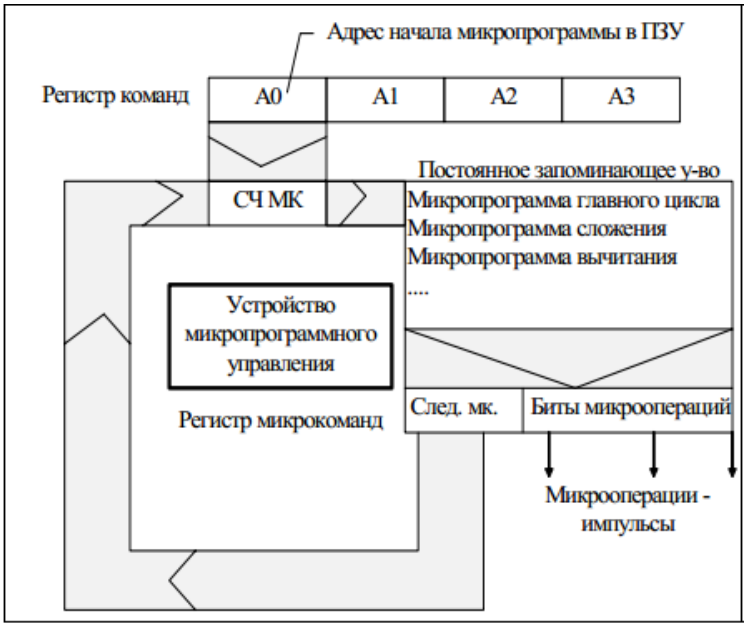


Рис. II.3. Структура микропрограммного управле

5. Микропрограммная реализация алгоритма главного цикла компьютера

Если микропрограммное управление являет собой программно - управляемый компьютер, то существует алгоритм главного цикла микро-программного управления.

Таблица II.3. Микропрограмма главного цикла

1: СЧМК := α
2: РМК := ПЗУ(СЧМК)
3: Исполнение микрокоманды - генерация микроопераций
4: РМК := АМК.РМК
5: Повторение 2

Алгоритм главного цикла устройства микропрограммного управления реализуется электронной схемой, которая гораздо проще электронной схемы аппаратного управления компьютером (Рис. II.1). Функционирование устройства микропрограммного управления обеспечивает исполнение любой программы обработки данных, введенной в оперативную память компьютера.

Мы уже говорили, что микропрограммная реализация управления компьютером более простая и дешевая по сравнению со схемотехнической реализацией. Имеются еще два преимущества микропрограммного управления. Вспомним, что система команд компьютера определяется набором микропрограмм, которые хранятся, как правило, в постоянном запоминающем устройстве (ПЗУ).

Ничто не мешает использовать при работе на одном и том же компьютере различные ПЗУ.

Это позволяет:

- **на современном компьютере - эмулировать команды старого компьютера, обеспечивая преемственность программного обеспечения;**
- **на современном - компьютере - эмулировать команды вновь разрабатываемого компьютера, обеспечивая одновременную разработку технического и программного обеспечения вновь создаваемого компьютера.**

2. Иерархия памяти компьютера

1. Зачем нужны внешние запоминающие устройства?

Практически с момента начала эксплуатации первых серийных вычислительных машин возникла проблема дефицита объема оперативной памяти.

Расширение объема оперативной памяти ограничивалось как техническими сложностями, так и высокой стоимостью изготовления одного регистра.

Для реализации запоминающих устройств большой емкости и с гораздо меньшей стоимостью изготовления одного регистра был использован принцип записи сигнала на движущийся магнитный носитель, который использовался в аудимагнитофонах. Сначала это были магнитные ленты и магнитные барабаны, затем появились магнитные диски. В настоящее время к ним добавились лазерные диски и флэш-память. Такой класс устройств получил название внешние запоминающие устройства

Более того, полезность внешних запоминающих устройств заключается в том, что они являются долговременными запоминающими устройствами, в отличие от оперативной памяти, в которой при отключении электрического питания данные исчезают.

Второе же свойство внешних запоминающих устройств, как уже отмечалось выше – большая емкость и дешевизна хранения данных. За все надо платить, и платой за эти достоинства является большое время доступа, т.е. малое быстродействие внешних запоминающих устройств.

Дело в том, что внешние запоминающие устройства (за исключением флеш – памяти) используют движущиеся носители данных и, в силу этого, не являются равнодоступными. Время доступа к байту на магнитном диске или лазерном диске варьируется в очень больших пределах, но в среднем является неизмеримо большим по сравнению с временем доступа к байту оперативной памяти. Поэтому прямой "побайтовый" обмен центрального процессора с внешним запоминающим устройством нерационален.

2. Специфика доступа в запоминающих устройствах на движущихся магнитных носителях.

Магнитные диски (дискеты, винчестеры) хранят двоичные коды в виде намагниченных (единица) и не намагниченных (ноль) участков магнитного носителя, которые образуются когда, участки магнитного диска проходят под записывающей головкой. Считывание основано на возникновении ЭДС, когда намагниченный участок проходит под считывающей головкой. Характеризуются малым быстродействием, но большой емкостью (десятки гигабайт). Записанные данные могут искажаться с течением времени под воздействием магнитных полей.

Двоичные коды на поверхности лазерного диска (компакт-диск, CD –диск, DVD –диск) представляются в виде углублений (единицы) и площадок (нули), выжигаемых лучом лазера во время записи. При считывании "ловится" отраженный от углублений и площадок лазерный луч, который преобразуется в электрические импульсы. Преимущество лазерных дисков – реализация сменных носителей данных без опасности искажения информации с течением времени. Очевидно, что при продвижении по иерархической структуре памяти вниз увеличивается объем памяти и время доступа, но уменьшается стоимость одного байта

3. Файлы и способы доступа к файлам.

Массив данных, которыми за один сеанс обмениваются внешнее запоминающее устройство и оперативная память, называется записью. Запись может состоять из нескольких полей. Группы записей, размещенные на внешних устройствах, называются файлами. Каждый файл идентифицируется уникальным именем.

Групповой обмен данными определяет специфический способ обработки больших массивов данных, хранящихся на внешней памяти. Хранящиеся на внешних устройствах данные разбиваются на записи, которые поочередно вызываются в оперативную память для обработки. После обработки текущей записи в оперативной памяти она может быть возвращена на место во внешнюю память. Различают несколько типов файлов.

Последовательный файл состоит из множества линейно упорядоченных компонент - записей. Специфика работы с последовательным файлом – добавление в конец новой записи, и последовательное чтение записей файла, начиная с первой.

Текстовый файл – частный случай последовательного файла, где в качестве записей используются символы.

Файл прямого доступа представляет собой множество записей, каждая из которых снабжена уникальным именем. Таким образом, возможен прямой доступ по имени к компонентам файла.
Индексно-последовательный файл представляет собой комбинацию прямого и последовательного доступа: реализовав прямой доступ к какой-либо записи, следующие за ней записи можно обрабатывать в режиме последовательного доступа.
Двоичный файл – состоит из одной записи и содержит либо двоичный код программы, либо двоичный код данных. Обращение к такому файлу реализуется по имени.
Названные выше типы файлов реализуются в системах программирования совместно средствами аппаратуры и операционной системы.
На аппаратном уровне файл представляет последовательность байт определенной длины. С каждым открытым файлом связан указатель на текущий байт, который надо считать или записать. Команды чтения – записи операционной системы считывают или записывают заданное число n байт, начиная с позиции определяемой указателем. Возможна предварительная установка указателя на заданный байт. После этого указатель перемещается на n байт. Таким образом, на машинном уровне имеется некоторое подобие индексно последовательного файла, записями которого служат байты.

4. Целесообразность иерархии запоминающих устройств

Регистры выполнены на транзисторных двоичных запоминающих элементах, расположены непосредственно в центральном процессоре и, в силу этого, являются сверхбыстродействующими.
Кэш-память, назначение которой подробнее рассматривается ниже, также выполняется на транзисторных двоичных запоминающих элементах, располагается близко к процессору и является быстродействующей.
Плата за быстродействие регистров и кэш-памяти – высокая стоимость байта.
Оперативная память строится на основе конденсаторных динамических запоминающих элементов, связана с центральным процессором шиной передачи данных и имеет среднее быстродействие.

3. Операционные системы и системы программирования

1. Средства автоматизации программирования и средства автоматизации управления вычислительным процессом

Несмотря на большую сложность аппаратуры компьютера, он неспособен обеспечить условия для эффективной работы пользователя -программиста.

Первая проблема - технология программирования в двоичных кодах, язык которых только и понимает аппаратура вычислительной машины, чрезвычайно трудоемка и потенциально содержит большие возможности для совершения ошибок в записи кода.

Вторая проблема - технология подготовки программы и пропуска ее через компьютер состоит из многих этапов, которые могут комбинироваться в различных вариантах.

Решение первой проблемы состоит в разработке средств автоматизации программирования, ориентированных на эффективность и надежность программирования. Вторая проблема решается посредством разработки операционных систем, управляющих вычислительным процессом.

Средства автоматизации программирования состоят из нескольких компонент.

- **Языки программирования высокого уровня, позволяют записывать алгоритмы в достаточно понятном виде не только для системного программиста, но и для проблемного специалиста (инженера, физика и т.д.). Кроме того, эти языки освобождают программиста от ручного распределения памяти за счет механизмов ее автоматического распределения, а также содержат средства конструирования сложной программы из более простых компонент (подпрограммы, сопрограммы, модули и т.д.).**
- **Трансляторы (компиляторы, интерпретаторы) представляют собой программы – переводчики с языка высокого уровня на язык двоичных кодов. Кроме того, трансляторы производят синтаксический контроль программы и реализуют сборку сложной программы из более простых компонент.**
- **Библиотеки стандартных программ содержат специальным образом оформленные программные реализации общезначимых алгоритмов, которыми можно пользоваться, не вдаваясь в детали их функционирования. Библиотеки специфичны для каждого конкретного языка программирования и постоянно расширяются фирмами, выпускающими на рынок трансляторы. Операционные системы расширяют сервисные и функциональные возможности аппаратуры компьютера за счет разработки системных программ.**

2. Основные функции операционной системы

Операционная система представляет собой большую и сложную системную программу, состоящую из двух частей. Относительно небольшая, резидентная часть (ядро) операционной системы постоянно размещена в оперативной памяти компьютера и реализует функции, время выполнения которых должно быть минимальным (распределение времени центрального процессора, анализ прерываний и т.д.). Гораздо большая по объему нерезидентная часть операционной системы расположена на внешних устройствах.

Выделяются три основных функции операционной системы:

1. Распределение ресурсов компьютера между множеством выполняемых (программ). К числу таких ресурсов относятся: оперативная память, время центрального процессора, внешние устройства (диски, принтеры и т.д.), системные программы общего назначения (программы ввода, программы вывода и т.д.). Эти функции выполняет управляющая программа планировщик и управляющая программа супервизор

2. Управление данными (каталогизация, хранение, поиск, защита данных). Эти функции реализует файловая подсистема.

3. Реализация "дружественного" интерфейса (связи) человека с аппаратурой компьютера. Долгое время на вопросы "дружественного" интерфейса не обращалось должного внимания. В силу этого с операционными системами могли общаться только системные программисты.

3. Понятие виртуальной вычислительной машины.

Принято говорить, что пользователь имеет дело с виртуальной машиной (совокупность аппаратных и программных средств), обеспечивающих достаточно "комфортную" работу человека с компьютером.Машина называется виртуальной (изменчивой) в силу того, что

ее сервисные и функциональные возможности зависят от того системного программного обеспечения, которое установлено на ней в настоящее время.

Общение с виртуальной вычислительной машиной реализуется посредством двух языков. Язык программирования (Паскаль, С++ и т.д.) является средством записи алгоритмов в виде программы. Язык управления заданиями (ЯУЗ) является средством формирования сценария пропуска программ через вычислительную машину.

4. Программа и процесс (задача)

Работа - действия по преобразованию данных, приводящие к решению прикладной задачи. Процесс (задача) – единица работы, существующая в динамике. Процесс функционирует в соответствии с некоторой программой, являющейся его статическим описанием. Говоря другими словами, процесс возникает, когда программа загружается в оперативную память и начинает претендовать на ресурс времени центрального процессора. Процесс прекращается, когда программа освобождает оперативную память и центральный процессор от своего присутствия.

Существенно, что несколько процессов, порожденных одной программой, могут существовать независимо, выполняясь последовательно или параллельно. В последнем случае программа, порождающая процесс, называется реентерабельной. Процесс является единицей работы, претендующей на ресурсы.

5. Пакетный и диалоговый режим работы операционной системы.

Стремление эффективно использовать центральный процессор привела к созданию пакетного режима обработки программ. Все, что надо было сделать с программой, какую информацию выдать и когда ее выдать, программировал пользователь на языке управления заданиями (ЯУЗ) операционной системы. Предложения ЯУЗ, программы и исходные данные переносились на перфокарты, колоды которых (задания) сдавались диспетчеру, после чего программист терял контроль над своей программой. Диспетчер формировал пакет заданий многих пользователей, а операционная система, исходя из своих приоритетов, создавала из заданий пакета задачи, выполняла их и выдавала результаты. Для технической поддержки пакетной технологии обработки программ был изобретен многозадачный (мультипрограммный) режим работы вычислительной системы. Он допускал одновременное существование нескольких задач. И если по какой-либо причине работающая задача временно не могла продолжаться, запускалась задача готовая к исполнению. Тем самым простои центрального процессора практически исключались. Таким образом, основное требование пакетного режима – процесс обработки программы со всеми ее деталями должен быть предусмотрен с начала и до конца. Неконструктивность такого подхода сказывалась особенно во время отладки. Но также нередки задачи, ход решения которых либо сложно, либо невозможно предсказать заранее. В силу сложности такого предсказания отладка и решение задачи проводились небольшими порциями, что вызывало неоправданно большие затраты времени на получение результата. Очевидно, что пакетная технология обработки программ невозможна без мощной управляющей программ – операционной системы.

Альтернативой пакетной технологии обработки программ явился диалоговый (интерактивный) режим функционирования операционной системы, который не требует знания всех особенностей вычислительного процесса с начала и до конца. Вычислительный процесс прерывается в точках возможного разветвления и запрашивает у пользователя указание для дальнейшего продолжения.

Для технической поддержки диалогового режима был изобретен режим разделения времени центрального процессора. В этом случае многие пользователи одновременно работают за своими абонентскими пультами (терминалами), возможно удаленными от вычислительной машины на значительное расстояние.

Время центрального процессора делится между абонентскими пультами путем выделения каждому небольшого кванта времени. При этом создается эффект квазиодновременной работы многих пользователей с одной вычислительной машиной.

4. Совершенствование адресации оперативной памяти

1. Расширение размеров адресного пространства за счет уменьшения числа адресов команды.

Здесь мы рассмотрим тенденцию уменьшения числа адресов команды, что позволяет избежать "пустых адресов" и, в некоторой степени, решает проблему расширения размеров адресного пространства оперативной памяти.

Возьмем для начала трехадресную команду, в полях адреса которой записывается физический адрес оперативной памяти. Пусть, например, имеем 48-ми битовую трехадресную команду (Рис. II.10), поле адреса которой имеет размер в 12 бит. В этом случае имеем размер адресуемого пространства оперативной памяти: Q = 2^{12} = 1024 = 4 Кб.

Уменьшив число адресов в этой команде до двух, получаем двухадресную команду Адреса такой команды принято называть Аист – адрес источника данных и Апр – адрес приемника данных. Получаем расширение адресного пространства до Q = 2^{18} = 256 кб. Двухадресная команда пересылки не содержит "пустых адресов" (MOV Аист Апр). Двухадресная команда передачи управления содержит один "пустой адрес" (вместо двух в трехадресной команде).

Оказывается, можно использовать и одноадресные команды. В этом случае однопараметрическое действие передачи управления реализуется одноадресной командой: (например, JMP addr), которая не содержит "пустых адресов". Но для обеспечения возможности исполнения двух и трех параметрических действий обработки данных необходимо изменить архитектуру арифметического устройства (Рис. II.10). Вместо трех регистров (два входных регистра и один регистр результата) используется единственный накопительный регистр R или регистр – аккумулятор. Команда использует два операнда: первый хранится в оперативной памяти, а второй являет собой результат выполнения предыдущей операции и хранится на аккумуляторном регистре. Результат исполнения команды остается на аккумуляторном регистре. В качестве платы за одноадресность, вводятся две специфические одноадресные команды обмена данными между оперативной памятью и регистром-аккумулятором:

LOAD addr - пересылка на аккумулятор R содержимого ячейки оперативной памяти с адресом addr;

STORY addr – пересылка в ячейку оперативной памяти с адресом addr содержимого аккумулятора R.

2. Схема выполнения трехпараметрических действий двухадресными и одноадресными командами.

А как обеспечить исполнение трехпараметрического действия обработки данных, например, действия сложения? Определим формат двухадресной команды сложения:

ADD Аист,Апр; (II.3)

Тогда, для реализации трехпараметрического действия необходимо выполнить последовательность из трех двухадресных команд:

MOV A1,temp; ADD A2,temp; MOV temp,A3.

Здесь используется вспомогательная ячейка оперативной памяти с адресом temp, а также команда MOV - пересылки между ячейками оперативной памяти (из источника в приемник).

LOAD addr - пересылка на аккумулятор R содержимого ячейки оперативной памяти с адресом addr;

STORY addr – пересылка в ячейку оперативной памяти с адресом addr содержимого аккумулятора R.

Выполнение двухпараметрического действия пересылки реализуется двумя одноадресными командами:

LOAD Аист; STORY Апр. (II.5)

Выполнение трехпараметрического действия сложения осуществляется следующим образом:

LOAD A1; ADD A2; STORY A3.

3. Однокомпонентные способы адресации.

Как уже говорилось неоднократно, самым простым и очевидным способом указания местоположения операнда является расположение в поле адреса команды полного (физического) адреса операнда в оперативной памяти, т.е. использование прямой адресации оперативной памяти. Синоним прямой адресации – абсолютная адресация.

Регистры общего назначения и регистровая адресация

Еще раз подчеркнем, что до сих пор мы имели дело только с прямой адресацией оперативной памяти, т.е. все адресные поля команд содержали физические адреса оперативной памяти. Но также использовались два вида памяти, которые адресовались по умолчанию: регистр аккумулятор R и стековая память.

Кроме всего прочего, регистр-аккумулятор, выполненный как сверхбыстродействующая ячейка памяти, при определенной технологии вычислений обеспечивал большее быстродействие за счет возможности не пересылать промежуточные результаты в оперативную память, а хранить их на регистре-аккумуляторе. В дальнейшем оказалось целесообразным ввести в состав центрального процессора массив сверхбыстродействующих ячеек, которые получили название регистров общего назначения (РОН).

На регистрах общего назначения можно хранить различные промежуточные результаты, текущую исполняемую команду, адреса стека и массивов данных и т.д.

По сути дела, в архитектуре компьютера, кроме оперативной памяти, появляется еще один класс памяти – сверхбыстродействующая регистровая. Такая память имеет небольшой объем (8 или 16 регистров) и имеет свою систему адресации. Адрес регистра общего назначения приня то называть номером регистра.

Регистровая адресация является ничем иным как прямой адресацией РОН (Рис. II.19b). В силу того, что для хранения номера регистра требуется незначительное число разрядов, поле адреса команды и вся команда в целом имеет небольшую длину. Как говорилось ранее, команда может включать в себя несколько адресных полей. Вполне допустимо, что каждое поле может иметь свою специфическую адресацию. Например, первое поле – регистровая адресация, второе поле – прямая адресация. Теперь имеется два класса памяти (оперативная и регистровая) и возникает немаловажный вопрос распознавания способа адресации, который используется в каждом поле адреса. Здесь используются два основных подхода.

Первый подход. Способ адресации задается кодом операции команды. Например, команда пересылки байта из регистра РОН в оперативную память записывается на языке ассемблера в виде:

MOV mem_byte, R1; (II.8)

и имеет код операции (88)16. Тогда как команда пересылки байта из оперативной памяти на регистр РОН записывается на языке ассемблера в виде:

MOV BL, mem_byte; (II.9)

и имеет код операции (8A)16.Второй подход. Признак регистровой адресации можно также указывать в поле адреса, которое в этом случае состоит из двух частей:

<префикс способа адресации> <номер регистра>.

Способы адресации современного компьютера (особенно PC) чрезвычайно разнообразны. Подробное их перечисление выходит за рамки настоящего издания. Скажем только, что в системе команд современного компьютера используются обе возможности: что-то определяется кодом операции, что-то префиксом способа адресации в поле адреса.

Косвенная регистровая адресация появилась впервые в шестнадцатиразрядных машинах как средство, позволяющее небольшим числом разрядов поля адреса адресовать большое количество регистров оперативной памяти.

В случае косвенной регистровой адресации, поле адреса команды содержит адрес регистра общего назначения, который является исполнительным адресом для доступа к ячейке оперативной памяти

Более строго, исполнительный адрес - функция, аргументами которой служат значение полей команды и/или содержимое регистров. В случае косвенной регистровой адресации имеем: Аисп = (RI). Здесь - RI номер регистра общего назначения, (RI) -содержимое этого регистра.

Автоинкрементная и автодекрементная адресация

Косвенная регистровая адресация позволяет эффективно реализовать последовательный доступ к ячейкам массива за счет последовательного прибавления константы (единицы) к содержимому регистра, содержащего исполнительный (косвенный) адрес. Для более эффективного использования этого свойства введены две разновидности косвенной регистровой адресации.

Автоинкрементная адресация обеспечивает вычисление исполнительного адреса, как и при косвенном регистровом способе, а затем автоматически увеличивает исполнительный адрес, хранящийся в РОН, на длину операнда в байтах

Автодекрементная адресация аналогична автоинкрементной адресации и обеспечивает вычитание размера операнда в байтах из исполнительного адреса.

4.Присоединенная адресация.

Уменьшение числа адресов команды позволило до некоторой степени расширить адресное пространство оперативной памяти. Но не столь радикально, как того требовали программисты. Необходимость радикального расширения адресного пространства без замедления выполнения команд, а также специфика мультипрограммного режима функционирования вычислительной машины привели к разработке многокомпонентных способов адресации, прежде всего, присоединенной адресации и относительной адресации по базе.

Страничная организация памяти и присоединенная адресация

Прообраз страничной организации памяти появился в трехадресных машинах типа М-20. Как сказано выше, 12 разрядов поле адреса допускали адресацию блока памяти (страницу) размером 212 = 4096 регистров. Можно поставить еще один блок (страницу) памяти, но как адресовать эту расширенную память? Чтобы разрешить эту коллизию, было предложено разбивать оперативную память на блоки и ввести в состав архитектуры вычислительной машины регистр приращения. В каждый момент времени центральный процессор работал с блоком оперативной памяти, номер которого хранился в двухбитовом регистре приращений. В систему команд вычислительной машины были введены специальные команды установки и запоминания значения регистра приращений. Таким образом, адресное пространство расширялось до четырех блоков (страниц) оперативной памяти. Глава II. Совершенствование адресации оперативной памяти

В общем случае страничная организация памяти предполагает разбиение оперативной памяти на равные по размеру страницы (блоки).

Пусть имеется N страниц, каждая из которых имеет длину L. На этой основе реализуется присоединенная адресация (Рис. II.22), которая использует исполнительный адрес Аисп, состоящий из двух компонент:

Аисп = <номер страницы>⊕<номер строки>.

При этом в адресном поле команды записывается только номер строки. Тогда как в качестве номера страницы может использоваться номер базовой страницы, номер текущей страницы или номер, хранящийся на специальном регистре страниц.

Очевидно, что размер адресуемого пространства оперативной памяти не зависит от длины поля адреса команды, который устанавливает только длину страницы, но определяется числом страниц, на которое «нарезана» оперативная память. В силу того, что исполнительный адрес вычисляется посредством функции конкатенации (присоединения), такая адресация называется присоединенной адресацией.

Сегментная организация памяти и адресация с индексированием

Страничная организация оперативной памяти и связанная с ней присоединенная адресация может использоваться при мультипрограммном режиме работы вычислительной системы. В этом случае несколько задач (программ вместе с данными) располагаются в оперативной памяти одновременно, и каждой задаче выделяется целое число страниц. Очевидно, что оперативная память расходуется нерационально: задаче, занимающей всего несколько строк в странице, отводится страница целиком. Этого недостатка лишен способ адресации, который использует сегментную организацию оперативной памяти. По сути дела, это модификация страничной организации памяти, при которой страницы могут иметь различную длину.

При адресации с индексированием два компонента адреса объединяются путем сложения. Этот способ является удобным средством для организации доступа к массивам и таблицам. Как показано, составной частью команды является базовый адрес addr, который всегда является физическим адресом оперативной памяти. Индекс (I) хранится в специальном индексном регистре I, который либо существует отдельно, либо является одним из регистров РОН. При вычислении исполнительного адреса индекс прибавляется к базовому адресу:

Аисп = addr+(I). (II.10)

5.Адресация по базе.

Способ адресации по базе также использует сегментную организацию памяти и похож на адресацию с индексированием, по этой причине их часто путают.

При способе адресации с индексированием команда содержит базовый адрес, а в индексном регистре находится смещение. Базовый адрес всегда является физическим адресом оперативной памяти. Смещение может быть как положительным, так и отрицательным и указывает положение операнда относительно базового адреса.

Еще раз подчеркнем, что базовым адресом является физический адрес оперативной памяти, что делает команду "длинной". Поэтому, индексная адресация не преследует цель эффективной адресации памяти большого объема, но позволяет эффективно программировать процедуры Работы с массивами и списками.

При адресации по базе ситуация обратная – команда содержит смещение, а в специальном базовом регистре B размещен базовый адрес (B)

Аисп = (B)+D.

6.Стековая адресация.

При автоматическом выполнении программы, записанной на языке высокого уровня, возникают две проблемы, которые для своего решения требуют использования стековой памяти.

Стековая память представляет собой динамический массив, дисциплина обслуживания для которого формулируется следующим образом: последний вошел – первый вышел.

Носитель текста, на основании которого строится стековая память, так же как и для случая оперативной памяти, представляет собой последовательность строк. Но число строк в этой последовательности изменяется при выполнении операций включения/ исключения.

При выполнении операций со стеком используется переменная связи X. Операция включения добавляет строку в вершину стека и записывает в нее значение переменной связи X. Операция исключения присваивает переменной связи X значение строки, находящейся в вершине стека, и исключает эту строку из стека. Необходима также логическая тестовая функция "Пустой стек".

7.Виртуальная память.

Привлекательна идея организации абстрактного адресного пространства памяти большого объема, относительно которого записывается программа обработки данных. При этом подразумевается, что существует система управления абстрактной памятью, которая берет на себя все трудности организации обработки в двухуровневой памяти. Такая абстрактная память получила название виртуальной памяти. Рассмотрим принципы ее организации.

В оперативной памяти выделяется какое-то число страниц фиксированной длины. Внешняя память (или часть ее) нарезается на страницы такой же длины . Предполагается, что число страниц внешней памяти значительно больше числа страниц оперативной памяти.

Вводится понятие виртуального (абстрактного) адреса, который состоит из двух компонент, как в присоединенной адресации:

Виртуальный адрес = V⊕p,

где V - номер страницы внешней памяти, p - номер строки в странице.

При работе с виртуальной памятью используются следующие функции.

Function Страница_ОП(V; var Nоп): Boolean;

выдает значение True, если страница внешней памяти V переписана в страницу Nоп оперативной памяти, и выдает значение False в противном случае.

Function Переполнение: Boolean; выдает значение True, если нет свободных страниц воперативной памяти.

Также используются вспомогательные процедуры.
Procedure Освобождение(Nоп);
освобождение страницы оперативной памяти в соответствии с каким-либо правилом.
Простейшее правило - освобождать страницу, к которой дольше всего не было обращения. При этом различаются два варианта освобождения страницы. Если в освобождаемую страницу не было записи, то номер страницы заносится в список пустых страниц, иначе страница из оперативной памяти переписывается на свое место во внешней памяти и только после этого ее номер заносится в список свободных страниц.
Procedure Подкачка(V;Nоп);
перепись содержимого страницы внешней памяти V в страницу оперативной памяти Nоп.
Таким образом, система управления виртуальной памятью обеспечивает работу непосредственно со страницами оперативной памяти, подкачивая в нее необходимые страницы из внешней памяти в случае необходимости.
Как уже говорилось, это обеспечивает эффективность программирования с использованием адресов большого адресного пространства виртуальной памяти. Кроме того, использование виртуальной памяти позволяет конструировать информационные системы обработки данных любого объема (в пределах размера виртуальной памяти). Причем, небольшой объем данных, который может разместиться в оперативной памяти, обрабатывается быстро, а данные большего объема обрабатываются медленнее, но всеже обрабатываются. Быстродействие обработки данных с использованием виртуальной памяти зависит от частоты смены страниц в оперативной памяти (малая скорость обмена между внешней памятью и оперативной памятью). В большинстве случаев вероятность того, что программа, затребовав доступ к странице Nоп, следующий доступ затребует к той же странице, достаточно велика (гипотеза компактного доступа к данным). Поэтому виртуальная память обеспечивает не только эффективность программирования в виртуальных адресах, но также быстродействие обмена данными в системе внешняя память – оперативная память.

8.Кэш-память как разновидность виртуальной памяти

Своеобразным вариантом виртуальной памяти служит распространенная в современных компьютерах система обмена данными между центральным процессором и оперативной памятью с использованием кэш памяти.
Как говорилось ранее, основное время выполнения команды составляет время обмена данными между оперативной памятью и центральным процессором. Кэш-память служит быстродействующим посредником между ними. Являясь относительно дорогостоящим устройством, кэш-память имеет небольшой объем по сравнению с оперативной памятью и хранит данные, к которым центральный процессор обращается достаточно часто.
Самой простой организацией обладает кэш-память с прямым отображением страниц оперативной памяти на строки кэш памяти. В этом случае в адресе оперативной памяти выделяется три поля: номер тома(тег), номер страницы в томе и смещение в странице.
Пусть, например, имеется оперативная память размером 2^{10} = 4096 байт. Исполнительный адрес в этом случае содержит 10 разрядов Пусть также кэш-память состоит из восьми строк, каждая из которых может хранить восемь страниц оперативной памяти.
В этом случае исполнительный адрес подразделяется на три поля: тег (том), страница в теге, смещение в странице. Всего имеется 16 тэгов (томов), в каждом томе содержится 8 страниц длиной в 8 строк каждая.
Таким образом, всего 2^{10} = 1024 байт. Для любого тега страница I отображается в строку I кэш-памяти. Таким образом, в строку I кэш-памяти отображается 16 страниц оперативной памяти (по одной из каждого тэга). Для того, чтобы определить, страница какого тома хранится в строке I кэш-памяти, используется дополнительное поле строки кэш памяти, хранящее номер тома.
Кроме прямого отображения, используются другие способы организации кэш-памяти, которые имеют как свои недостатки, так и свои преимущества. Но назначение кэш памяти не зависит от способа ее организации.

В силу того, что центральный процессор работает не непосредственно с оперативной памятью, а использует посредника в виде кэш памяти, возникает проблема поддержания соответствия одних и тех же единиц данных, хранящихся в посреднике и в оперативной памяти (проблема целостности данных).

Эта проблема возникает при замещении страницы оперативной памяти, хранящейся в строке I кэш-памяти, новой страницей с тем же номером, но из другого тега. Смена содержимого строки I кэш-памяти реализуется следующим образом:

- **если в строку кэш-памяти I записи не производилось, в нее можно переписывать новую страницу оперативной памяти I из тэга T2;**
- **в противном случае, перед переписью новой страницы из тега T2 в строку I кэш-памяти, страница оперативной памяти I из тега T1 заменяется содержимым строки I кэш-памяти.**

Выше мы обсуждали способ повышения быстродействия компьютера за счет использования кэш-памяти для данных. Что касается команд, в программах часто встречаются участки, многократно выполняемые в цикле. При этом центральный процессор многократно считывает из оперативной памяти одни и те же команды. Если использовать кэш-память для промежуточного запоминания команд и если эта память достаточного объема, чтобы хранить все команды одного цикла, вычисления будут выполняться с большей скоростью. Более того, для каждого типа компьютеров можно разрабатывать оптимизирующие компиляторы, которые анализируют заданные программистом циклы и разбивают их на циклы меньшего размера, размещающиеся в кэш памяти команд целиком. Более того, используются кэш-памяти для дисковой памяти, что позволяет существенным образом снять проблему замедления вычислений при работе с внешней памятью.

5. Структурная организация вычислительной машины

1. Прямое управление вводом-выводом.

При использовании прямого управления в состав устройства управления компьютера, наряду с устройством автоматического исполнения программы (УУАВП), входит также устройство управления вводом –выводом УУВВ (Рис. II.33).

Прямое управление подразумевает наличие специальных команд для программирования ввода и вывода. Существенно, что такие команды выполняются непосредственно центральным процессором. Это не вызывает вопросов с точки зрения логики программирования, но приводит к чрезвычайно неэффективной работе вычислительной системы.

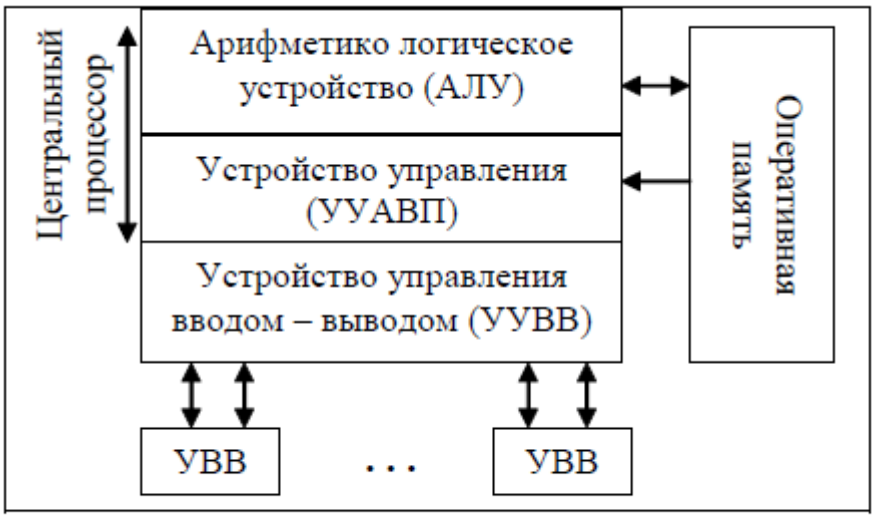


Рис. II.33. Прямое управление устройствами ввода – вывода компьютеров первого поколения

2. Целесообразность параллельной работы центрального процессора и устройств ввода-вывода.

В случае программ, содержащих значительное число команд ввода-вывода, центральный процессор в основном простаивает.Для решения этой проблемы конструкторы предложили выполнять устройство управления вводом – выводом в виде отдельного специализированного процессора (или несколько специализированных процессоров), реализующего только команды ввода и вывода. Главная идея этого нововведения состоит в следующем: выполняя команду ввода–вывода центральный процессор не занимается передачей данных между оперативной памятью и внешними устройствами, а лишь запускает УУВВ и продолжает выполнять следующие команды программы. Устройство ввода - вывода выполняет работу по пересылке и преобразованию данных, функционируя параллельно с центральным процессором.

3. Использование канала ввода-вывода.

С целью еще большей степени параллелизма функционирования центрального процессора и УВВ был разработан так называемый канал данных. Канал данных представляет собой специализированный процессор,созданный для выполнения только операций ввода–вывода

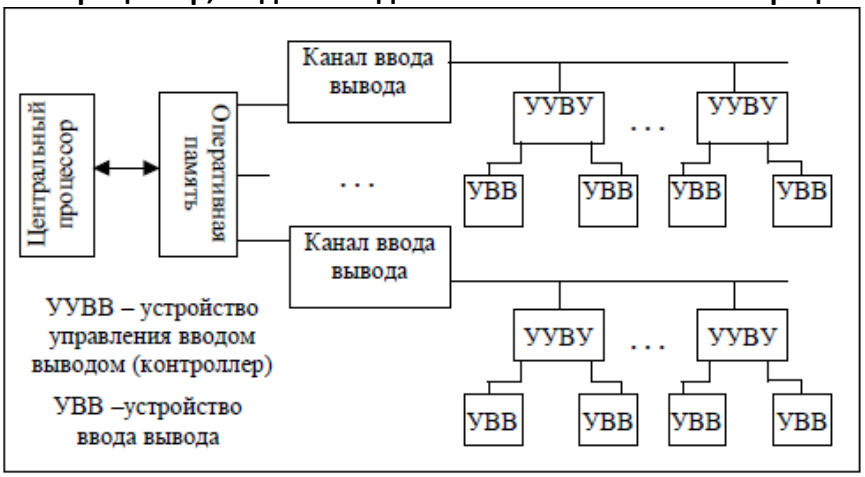


Рис. II.34. Структурная организация с каналами ввода – вывода

4. Понятие порта ввода-вывода.

Порт ввода – вывода (специфический регистр ввода – вывода) является частью интерфейса (контроллера) устройства ввода – вывода и представляет собой группу разрядов, значение которых доступно центральному процессору во время исполнения операций ввода – вывода.

5. Архитектура с раздельными шинами данных.

В этом случае оперативная память подключается к центральному процессору посредством шины памяти, а подсистема ввода – вывода подключается к центральному процессору посредством шины ввода – вывода

Как шина памяти, так и шина ввода-вывода включают в себя:линии адреса, линии данных, линии управления

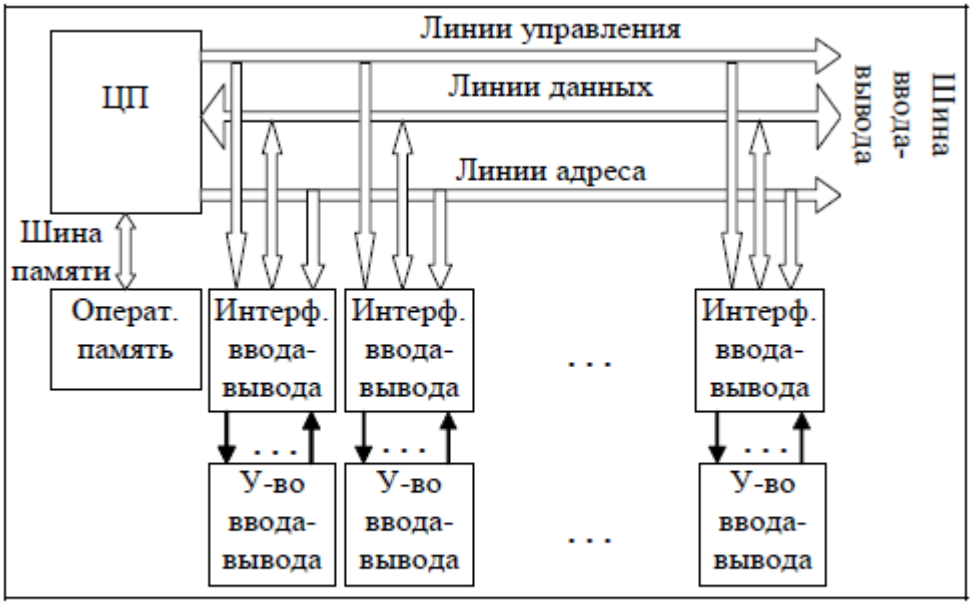


Рис. II.36. Структурная организация с шиной памяти и шиной ввода – вывода

Так как основная оперативная память и порты ввода-вывода (в составе интерфейсов ввода – вывода) подключены к разным шинам, адресные пространства для команд ввода – вывода и для остальных команд различны.

6. Архитектура с общей шиной данных.

В этом случае, как устройства ввода – вывода, так и оперативная память подключены к единой шине данных (Рис. II.37). Говоря другими словами, это ввод – вывод, организованный по аналогии обращения к памяти.

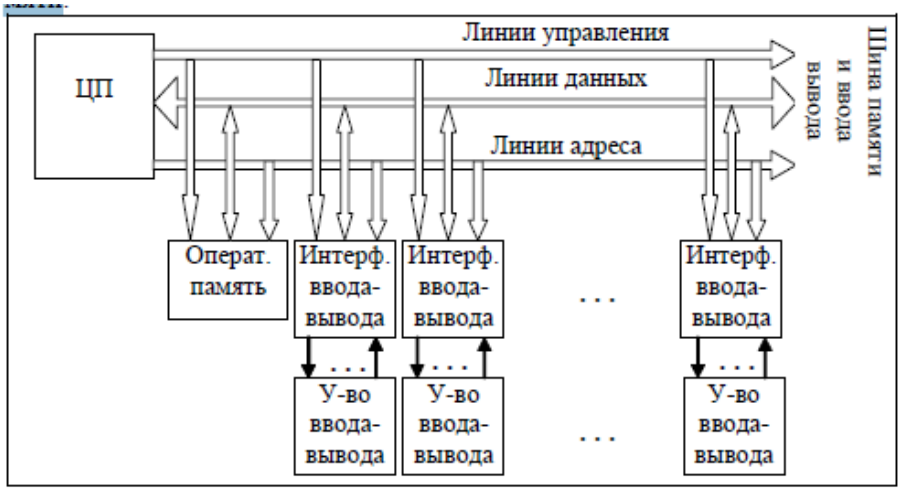


Рис. II.37. Структурная организация с единой шиной

Каждый порт ввода–вывода имеет адрес в пространстве адресов оперативной памяти. Порт ввода допускает выполнение любой команды,которая осуществляет чтение по его адресу, порт вывода–любой команды,которая производит запись по его адресу.
MOV addr_mem, addr_mem - команда пересылки в оперативной памяти;
MOV addr_mem, rp – команда вывода;
MOV rp, addr_mem – команда ввода.
В зависимости от того, какое устройство подсоединено к порту с адресом rp, ввод или вывод будет осуществляться на это устройство.

6. Параллельная работа центрального процессора и устройств ввода-вывода

1. Необходимость параллельной работы центрального процессора и устройств ввода-вывода.

Выполняемая программа содержит цикл, включающий в себя команду вывода "печать строки". Выполняя эту команду,центральный процессор заполняет буфер вывода символами строки и запускает устройство вывода (ВУ). Пока ВУ осуществляет вывод буфера центральный процессор может продолжать выполнение основной программы и готовить для печати следующую строку. ***Центральный процессор и устройство вывода работают параллельно.*** **Но подготовка следующей строки для печати, как правило, выполняется быстрее, чем реализуется вывод предыдущей строки. Т.е. следующая команда "печать строки" не может быть выполнена до завершения вывода предыдущей строки. И центральный процессор простаивает, т.е. быстродействие центрального процессора все равно ограничивается медленным электромеханическим устройством.**

2. Прерывания как средство управления вводом-выводом.

Прерывание – реакция центрального процессора на некоторое событие, приводящая к прекращению выполнения текущей задачи с целью выполнения какой – либо другой, более важной, задачи.

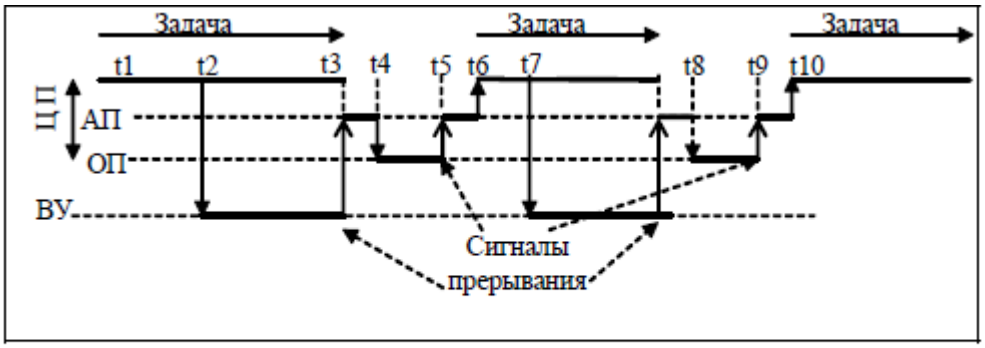


Рис. II.38. Прерывание задачи внешним устройством: нет простоя процессора (АП – анализ прерывания, ОП – обработка прерывания, ВУ – внешнее устройство)

Глава 7_2 вопросы

1. Стратегия распределения времени центрального процессора между несколькими задачами.

Отдав приказ на вывод (t2) и подготовив вывод следующий строки, Задача1 не может продолжать работу, она выдает сигнал прерывания, который запускает на выполнение Задачу2 (t3).

Теперь сигнал прерывания от внешнего устройства прерывает Задачу2 в момент времени t3' и запускает Задачу1на продолжение. Последняя выдает приказ на вывод в момент t4 и т.д.

2. Понятие прерывания и обработки прерывания.

Прерывание – реакция центрального процессора на некоторое событие, приводящая к прекращению выполнения текущей задачи с целью выполнения какой – либо другой, более важной, задачи. События,требующие прерывания, могут произойти как внутри компьютера (сбои в работе отдельных устройств, попытка деления на нуль, коллизии в устройствах ввода–вывода и т.д.), так и вне его (нажатие клавиши при вводе символа, запрос на совместную работу с другим компьютером и т.д.). Во всех случаях моменты возникновения событий заранее неизвестны и не могут быть учтены при программировании

Обычно выделяются пять типов прерываний:

1. Прерывания от схем аппаратного контроля вычислительной машины возникает в случае неисправности оборудования.

2. Внешние прерывания дают возможность центральному процессору реагировать на сигналы от таких источников, как таймер, кнопка прерывания на клавиатуре, внешние датчики и т.д.

3. Прерывания от ввода–вывода позволяют центральному процессору реагировать на события, возникающие в контроллерах и устройствах ввода–вывода.

4. Программные прерывания происходят в тех случаях, когда в исполняемой программе возникает какой – либо особый случай (переполнение разрядной сетки, деление на нуль и т.д.).

5. Прерывания при обращении к операционной системе возникают, когда в исполняемой программе встречается команда перехода в "защищенный режим".

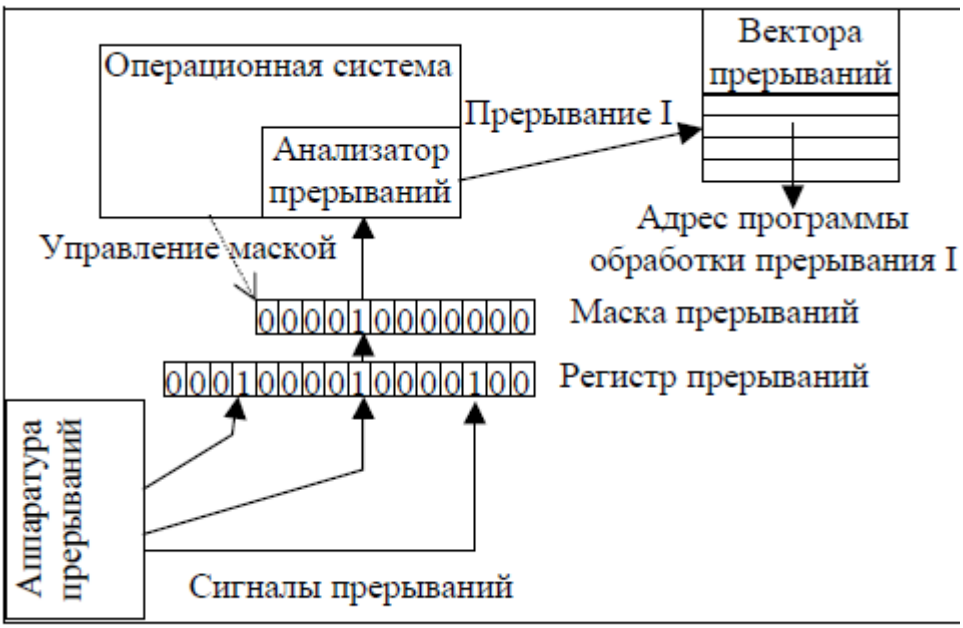


Рис. II.41. Маскирование и обработка прерываний.

3. Регистр прерываний и маска прерываний.

Регистр прерываний являет собой двоичный регистр, каждый разряд которого соответствует какому-либо прерыванию. Когда от аппаратуры прерываний приходит сигнал прерывания, соответствующий разряд регистра прерываний устанавливается в "1". После окончания обработки прерывания этот разряд устанавливается в "0".

Для сортировки поступивших прерываний используется двоичный регистр маски прерывания, каждый бит которого соответствует определенному биту регистра прерываний. Если бит регистра прерываний установлен в единицу и также в единицу установлен соответствующий бит маски, – сигнал прерывания проходит на анализатор, который запускает программу обработки этого прерывания. Установкой в "1" и "0" регистров маски управляет операционная система. Образно говоря, регистр маски представляет собой своеобразное решето с управляемыми ячейками.Обычно выделяются пять типов

4. Слово состояния задачи и цикл прерывания.

При прерывании программы PI, и запуска вместо нее программы PJ, для обеспечения возможности продолжения нормальной работы PI,необходимо запоминать данные, которые может испортить программа PJ.Все эти данные сводятся в одно слово состояния программы и в момент прерывания запоминаются в стековой памяти

Цикл прерывания

1. Сигнал прерывания принимается блоком управления прерываниями.

2. Используемая в данный момент времени команда доводится до завершения в установленном порядке.

3. Состояние счетчика команд и регистров центрального процессора запоминается в стеке.

4. Реализуется переход на программу обработки соответствующего прерывания.

5. При завершении программы обработки прерывания управление передается либо на прерывающую задачу (если она готова продолжаться), либо другой программе, готовой к продолжению. При этом восстанавливаются значение счетчика

8. Компьютерыпараллельного действия

1. Параллелизм как принципиальное средство увеличения быстродействия вычислительной системы.

Здесь можно назвать два основных подхода.

Один подход рассматривает компьютеры параллельного действия как набор микросхем, которые соединены друг с другом определенным образом. При другом подходе возникает вопрос, какие процессы выполняются параллельно. Здесь имеется два крайних варианта.

Для решения задач на уровне крупных структурных единиц используются вычислительные системы со слабой связью – множество процессоров, которые взаимодействуют по схемам с низкой скоростью передачи данных. Для решения задач на уровне мелких структурных единиц. используются системы с тесной связью – компоненты аппаратуры меньше,расположены ближе друг к другу и взаимодействуют через специальные коммуникационные сети с высокой пропускной способностью.

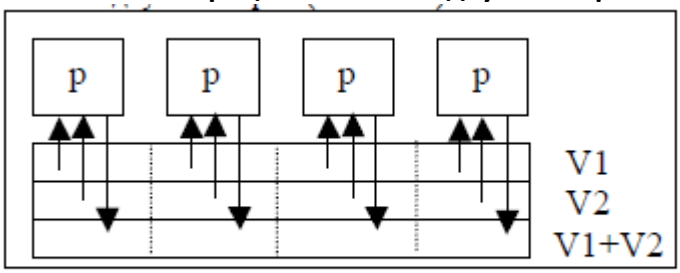
2. Конвейерная обработка команд.

	1	2	3	4	5	6	7	8	9
C1									
C2		1	2	3	4	5	6	7	8
C3			1	2	3	4	5	6	7
C4				1	2	3	4	5	6
C5					1	2	3	4	5

Рис. II.44. Конвейерная обработка команд

3. Параллелизм в случае векторных вычислений.

при научно-технических расчетах необходимо выполнять операции над векторами и матрицами очень большого размера.Практически всегда такие данные можно разбить на сегменты, число которых определяется числом процессоров, и одновременно выполнять обработку всех сегментов данных. Такие процессоры имеют общую память и называются мультипроцессорами. Характерным примером является операция сложения двух векторов



ис. II.45. Сложение векторов, реализованное на четырех параллельных процессорах.

Компьютеры, состоящие из нескольких процессоров и выполняющие одни и те же вычисления над разными сегментами одного набора данных в одно и то же время, называются векторными компьютерами

4. Понятие потока управления и его роль в организации параллельных вычислений.

Поток управления – последовательность, в которой команды выполняются динамически, т.е. во время выполнения задачи. Последовательный поток управления изменяется командой вызова процедуры. Прерывание – это изменение в потоке управления, вызванное не самой программой, а возникшем событием в системе.

При начале работы w по запросу одного пользователя запускается задача t(w), которая порождает свой клон в виде потока управления s(p(w)), d1)) по обработке данных d1. При поступлении запроса на выполнение той же работы w с другим набором данных d2 (другой пользователь) задача t(w) порождает еще один клон в виде потока управления s(p(w),d2), выполняющего задачу t(w) для второго пользователя по обработке данных d2 и т.д.

5. Нити и параллелизм внутри одного потока управления.

При выполнении потока управления на одном процессоре могут возникать ситуации, когда его дальнейшее продолжение невозможно до наступления вполне определенного события в вычислительной системе, например – до завершения операции ввода данных. В случае мультипрограммирования единственный процессор переключается на выполнение другой задачи. В случае многопроцессорной системы запуск новой задачи или другого потока управления далеко не всегда целесообразен, и для устранения простоя процессора в некоторых случаях можно использовать параллелизм на уровне одного потока управления.

Если программа выполнения работы такая, что в ней можно выделить несколько участков, которые можно выполнять независимо друг от друга (либо всегда, либо при некоторых условиях), то для каждого такого участка создается частичный поток управления – нить (Рис. II.46). Все нити выполняются на одном процессоре в режиме традиционного мультипрограммирования.

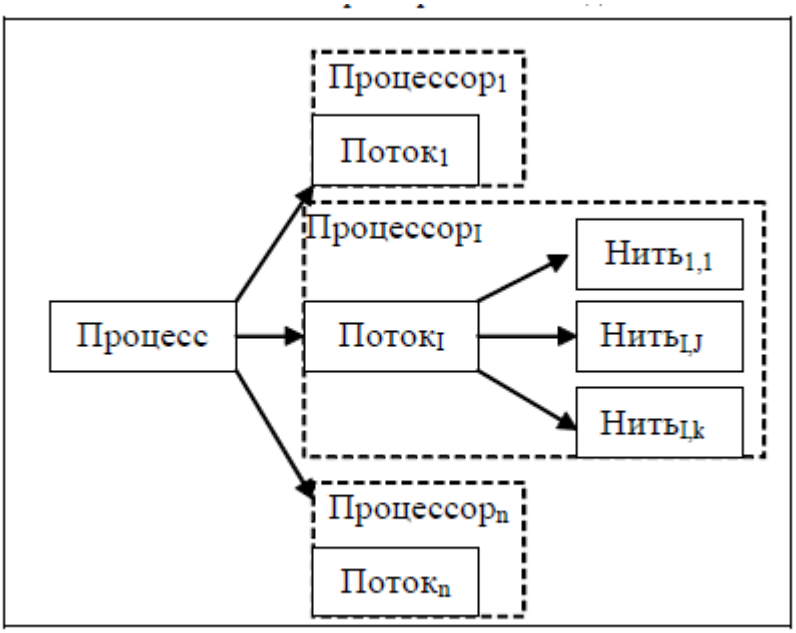


Рис. II.46. Процесс порождает потоки, каждый поток порождает нити

6. Параллелизм на уровне процессов.

Во многих случаях одну супер работу по обработке данных целесообразно представить в виде нескольких слабо связанных по данным работ, каждая из которых может выполняться на своем процессоре, обмениваясь данными с другими работами. Обмен данными между процессорами происходит по сети межсоединений, и вычислительная система превращается в мультикомпьютер.

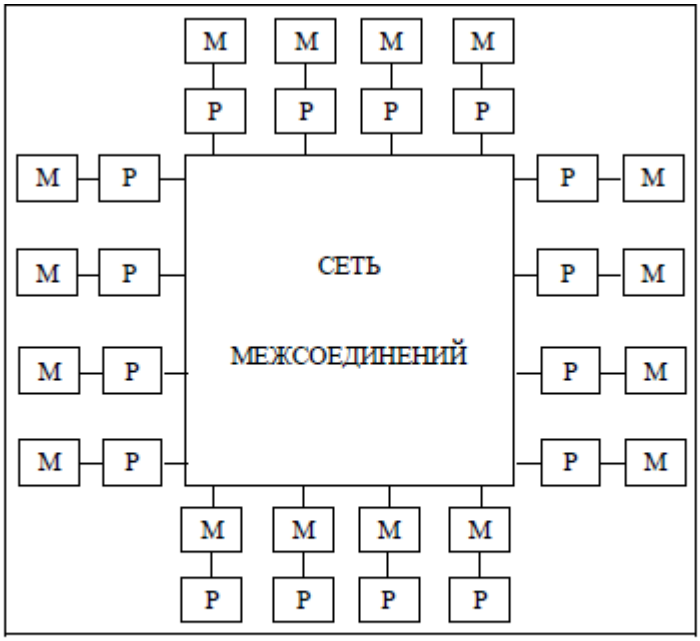


Рис. II.47. Мультикомпьютер, состоящий из 16 – ти процессоров, каждый из которых имеет собственную память

Ключевое отличие мультипроцессора от мультикомпьютера состоит в том, что каждый процессор в мультикомпьютере имеет собственную память, к которой никакой другой процессор не может обращаться непосредственно. Поскольку процессоры в мультикомпьютере не могут взаимодействовать друг с другом путем чтения из общей памяти, они посылают друг другу сообщения, используя сеть межсоединений. При этом основной проблемой становится правильное разделение данных и разумное их размещение, тогда как в мультипроцессоре размещение сегментов данных не влияет на правильность выполнения задачи.

9. Автоматизация программирования

1. Два уровня представления данных и программ в компьютерах.

Уже здесь появляется специфика двухуровневого представления данных и программ в системе человек - компьютер:

• на внешнем уровне необходимо представлять данные и программы в виде, максимально удобном для человеческого восприятия (графика, тексты, таблицы и т.д.);

• такое представление данных и программ, как правило, неэффективно с точки зрения ее обработки компьютером;

• на внутреннем уровне данные и программы представляются в виде эффективном для их обработки (двоичные коды, списковые структуры, структуры данных);

• как следствие, необходимы специфические программы – трансляторы , реализующие автоматическое преобразование представления данных и программ с человеческого уровня на машинный уровень и обратно

2. Языки символьного кодирования и ассемблирования программ.

Символьное кодирование

В результате предписание на выполнение действия в виде символьной строки. Например, команду сложения можно обозначить следующим образом: ADD 1001B,1010B,1011B; (II.15)

где B - признак двоичного числа. Каждая символьная строка, обозначающая машинную команду, называется мнемокомандой, а совокупность таких команд образует язык автокод (язык автоматического кодирования программы) Дальнейшим развитием языка автокод является введение в его описательные возможности средств управления полуавтоматическим распределением оперативной памяти. Суть такого распределения памяти состоит в следующем:

• ячейки и массивы ячеек можно обозначать символьными именами (адресами);

• в адресных полях команды можно использовать эти символьные имена (адреса);

• имеется псевдокоманда MEM, позволяющая установить соответствие символьного адреса и физического адреса;

• имеется псевдокоманда CONST, позволяющая присвоить ячейке памяти с заданным символьным адресом конкретное символьное значение.

Ассемблирование программ

Прежде всего, ассемблер реализует функции трансляции (перевода) с языка автокод на язык машинных команд. В качестве входного набора данных для ассемблера выступает программа, записанная на языке автокод, которая переводится в перемещаемую программу на машинном языке.

Подводя итог, можно сказать что ассемблером (англ. assembler, от assemble - собирать, монтировать) называется компонента системного программного обеспечения, переводящая программу, написанную на автокоде (символьный модуль) в машинную программу (загрузочный модуль). Программа на автокоде представляется в командах, отражающих архитектуру конкретного типа компьютера. Помимо трансляции (перевода) важнейшей функцией ассемблера является сборка или связывание многих объектных модулей в единый загрузочный модуль.

3. Программы загрузчики.

Назначение загрузчика определяется его именем. Он загружает программу в оперативную память машины. Различаются три типа загрузчиков.: двоичные, настраивающие и связывающие.

Двоичный загрузчик **загружает машинную программу, записанную в двоичном виде относительно фиксированных адресов, с какоголибо внешнего устройства.**

Настраивающий загрузчик **получает программу в перемещаемой двоичной форме (записанную в относительных адресах) вместе с информацией о настройке ее по месту и размещает ее в памяти, предварительно настроив относительные адреса.**

Связывающий загрузчик **получает набор объектных модулей, записанных в двоично-символьной форме, и загружает их в оперативную память как единую программу, заполнив перекрестные ссылки между ними и настроив их по месту.**

4. Понятие подпрограммы.

теоретики и практики программирования изобрели механизм подпрограммы. Вместо статической подстановки, размножающей текст макроопределения, было предложено использовать динамическую подстановку. Суть ее состоит в том, что подпрограмма существует в единственном экземпляре и при каждом к ней обращении в тексте основной программы происходит динамическая подстановка. При этом управление передается на начало подпрограммы и обеспечивается возврат в основную программу после окончания работы подпрограммы

5. Макросы как средство статической подстановки подпрограмм.

Для того, чтобы при составлении программы многократно не переписывать один и тот же (или, почти один и тот же) текст, был изобретен аппарат макросов – макрокоманд, определяемых пользователем.

Макроопределение служит для конструирования пользователем своей собственной макрокоманды, более крупной, по сравнению с машинной. Макроопределение состоит из заголовка и тела.

Макро - это псевдокоманда ассемблера, сообщающая ему, что данное предложение является заголовком макроопределения. Имя макро используется для последующих ссылок на это макроопределение, а список формальных параметров определяет каналы связи макроопределения с основной программой.

Предполагается, что перед запуском ассемблера будет использоваться макрогенератор, который в процессе просмотра макропрограммы каждый раз, когда встретит макровызов, будет осуществлять макроподстановку. в процессе макрогенерации каждый макровызов будет заменен телом соответствующего макроопределения. Причем, имена формальных параметров будут заменены именами фактических параметров.

6. Переменная в императивных языках программирования высокого уровня.

Прежде всего, было введено понятие переменной и понятие типа переменной. С одной стороны, переменная - это символьное обозначение области оперативной памяти, как в автокоде. Но "привязывание" к каждой переменной определенного типа означает определение правила интерпретации текущего значения и области допустимых значений этой перемен-
ной. Это позволяет решить, по крайней мере, две проблемы автоматизации программирования:

- **полностью автоматическое распределение памяти для хранения значений переменных;**
- **автоматический контроль корректности использования операндов в операциях.**

7. Оператор присваивания как императив вычисления функции.

Оператор присваивания можно назвать основным оператором языка программирования высокого уровня и утверждать, что остальные операторы только "обслуживают" его. На самом деле, только оператор присваивания позволяет устанавливать значение переменной левой части в зависимости от значений переменных правой части. По сути дела, *оператор присваивания является императивом (предписанием) для вычисления функции F, заданной арифметическим выражением F'*: y = F'(x1,x2,...,xn) (II.21)

Благодаря оператору присваивания, проблемный специалист может записывать предписания на вычисление математических формул в почти привычном для него виде.

8. Операторы обработки данных.

подразделяют на три класса: операторы ввода данных, операторы вывода данных, операторы присваивания.

Оператор ввода позволяет устанавливать извне (с помощью устройств ввода) значение входной переменной. Аналогично, оператор вывода позволяет выводить на внешнее устройство значение выходной переменной. Тут возникает проблема преобразования внешнего представления единицы данных во внутреннее представление и обратно. В Фортране впервые были использованы своеобразные шаблоны - форматы, которые позволяли программировать эти преобразования. Наряду с другими способами, форматный ввод-вывод сохранился во многих современных языках.

9. Операторы изменения хода вычислительного процесса.

Для изменения порядка выполнения операторов присваивания был введен оператор безусловного перехода: GO TO N.

Предполагалось, что некоторые операторы могут быть пронумерованы (иметь метки) и N - одна из меток.

Оператор условного перехода появился в ФОРТРАНЕ, как бы сейчас сказали, в укороченной форме:

IF (<условие>) THEN <альтернатива да>,

где <альтернатива да> - произвольный оператор, чаще всего GO TO. В качестве условия используется логическое выражение, которое представляет собой специальную запись логической (булевской) функции. Достоинством такого решения была возможность задавать достаточно сложные условия перехода по сравнению с условием равенства -неравенства нулю для машинной команды условного перехода.

Недостатков укороченного оператора условного перехода было,по крайней мере, два. Во-первых, альтернатива состояла только из одного оператора. Этот недостаток был исправлен уже в АЛГОЛе, где определи- ли составной оператор как любую последовательность операторов, заключенную в операторные скобки:IF(условие) THEN begin S1;…; Sn end;

Второй недостаток связан с несимметрией оператора условного перехода, и этот недостаток был устранен введением в последующих языках программирования "структурного" оператора условного перехода: IF(условие) THEN <альтернатива да>ELSE<альтернатива нет>;

Также в последующих языках, кроме условного оператора двоичного выбора, был введен оператор множественного выбора case. Последний является обозначением системы вложенных операторов IF, определяющих множественный выбор.

10. Подпрограммы процедуры и подпрограммы функции.

теоретики и практики программирования изобрели механизм подпрограммы. Вместо статической подстановки, размножающей текст макроопределения, было предложено использовать динамическую подстановку. Суть ее состоит в том, что подпрограмма существует в единственном экземпляре и при каждом к ней обращении в тексте основной программы происходит динамическая подстановка. При этом управление передается на начало подпрограммы и обеспечивается возврат в основную программу после окончания работы подпрограммы

Подпрограммы в языках программирования

высокого уровня бывают двух типов. Подпрограммы функции предоставляют для программиста средства конструирования собственных стандартных функций, которые затем можно использовать в выражениях (арифметических и логических). Аналогично, подпрограммы процедуры являются средством конструирования векторных операторов присваивания.

11. Трансляция и компиляция.

Транслятором принято называть системную программу, преобразующую текст программы пользователя в семантически тождественный текст, записанный на языке более низкого уровня.

Трансляторы подразделяются на два класса: компиляторы и интерпретаторы. Выше мы описали работу транслятора компилирующего типа- компилятора.

Транслятор интерпретирующего типа - интерпретатор, по сути дела, представляет собой виртуальную машину "понимающую" язык программирования высокого уровня (Рис. II.50). Такая виртуальная машина состоит из аппаратуры компьютера и транслятора интерпретирующего типа, расширяющего возможности аппаратуры до "понимания" языка программирования высокого уровня.