Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**А. С. Кобайло**

**АРИФМЕТИКО-ЛОГИЧЕСКИЕ**

**ОСНОВЫ ЦИФРОВЫХ**

**ВЫЧИСЛИТЕЛЬНЫХ МАШИН И**

**АРХИТЕКТУРА КОМПЬЮТЕРОВ**

Конспект лекций по одноименной дисциплине

для студентов IT-специальностей

Минск 2017

УДК 004.2.031(075.8)

ББК 32.97.я73

К55

Рассмотрено и рекомендовано редакционно-издательским советом Белорусского государственного технологического университета

Рецензенты:

кандидат технических наук, доцент,

доцент кафедры программного обеспечения информационных технологий Белорусского

государственного университета информатики и радиоэлектроники

*П. Ю. Бранцевич;*

Заведующий кафедрой информационных технологий и экологического менеджмента ГУО «Институт бизнеса и менеджмента технологий» БГУ

*Б.В. Лесун*

**Кобайло, А. С.**

К55 Арифметико-логические основы цифровых вычислительных машин и архитектура компьютеров: конспект лекций по одноименной дисциплине для студентов IT-специальностей / А. С. Кобайло. – Минск : БГТУ, 2015. – 79 с.

Конспект лекций содержит основные сведения в области схемотехники ЦВМ, архитектурной организации микропроцессоров, основ программирования на языках низкого уровня. Рассмотрены вопросы организации обмена данными в микропроцессорных системах, а также структуры конкретных микропроцессоров.

Издание предназначено для студентов, изучающих дисциплину «Арифметико-логические основы цифровых вычислительных машин и архитектура компьютеров». Пособие также может быть полезно аспирантам, изучающим избранные аспекты вышеуказанных проблем.

**УДК 004.2.031(075.8)**

**ББК 32.97.я73**

© УО «Белорусский государственный

технологический университет», 2017

© Кобайло А. С., 2017

Содержание:

[ПРЕДИСЛОВИЕ 5](#_Toc471648302)

[1. Арифметические основы цифровых вычислительных машин 7](#_Toc471648303)

[2.5. Синтез логических схем по логическим выражениям 72](#_Toc471648304)

[2.5.1. Синтез логических схем в базисе И, ИЛИ, НЕ 72](#_Toc471648305)

[2.5.2. Логические базисы И-НЕ, ИЛИ-НЕ 74](#_Toc471648306)

[3. ЦИФРОВОЙ ЛОГИЧЕСКИЙ УРОВЕНЬ АРХИТЕКТУРЫ ЦВМ 80](#_Toc471648307)

[3.1. Элементы ЭВМ 80](#_Toc471648311)

[3.1.1. Логические элементы 80](#_Toc471648312)

[3.1.2. Запоминающие элементы 85](#_Toc471648314)

[3.2. УЗЛЫ ЭВМ 97](#_Toc471648317)

[3.2.1. Комбинационные узлы 97](#_Toc471648318)

[3.2.2. Накапливающие узлы 112](#_Toc471648324)

[3.2. Устройства ЦВМ 122](#_Toc471648328)

[3.3.1. Арифметико-логическое устройство 122](#_Toc471648329)

[3.4. Устройство управления 132](#_Toc471648332)

[3.4.1. Схемное устройство управления. 135](#_Toc471648334)

[3.4.2. Структурная схема микропрограммного устройства управления. 140](#_Toc471648336)

[3.5. Запоминающие устройства 142](#_Toc471648339)

[3.5.1. Классификация запоминающих устройств.. 143](#_Toc471648340)

[3.5.2. Построение ЗУ с заданной организацией.. 147](#_Toc471648341)

[4.АРХИТЕКТУРА МИКРОПРОЦЕССОРНЫХ СИСТЕМ 168](#_Toc471648342)

[4.1. Организация компьютерных систем 168](#_Toc471648343)

[4.1.1. Основные понятия 168](#_Toc471648344)

[4.1.2. Что такое микропроцессор 171](#_Toc471648345)

[4.1.3. Структура типового микропроцессора 178](#_Toc471648346)

[4.1.4. Логическая структура микропроцессора 183](#_Toc471648347)

[4.1.5. Типы архитектур 185](#_Toc471648348)

[4.2. Обмен данными в микропроцессорной системе 188](#_Toc471648349)

[4.2.1. Шинная структура связей 188](#_Toc471648350)

[4.2.2. Режимы работы микропроцессорной системы. 193](#_Toc471648351)

[1.4. Архитектура микропроцессорных систем. 199](#_Toc471648352)

[4.2. Архитектура микропроцессора 8086 204](#_Toc471648353)

[5. УРОВЕНЬ АРХИТЕКТУРЫ НАБОРА КОМАНД 212](#_Toc471648354)

[5.1. Система команд 212](#_Toc471648355)

[5.1.1. Классификация команд 212](#_Toc471648356)

[5.1.2. Режимы адресации 214](#_Toc471648357)

[5.1.3. Адресация операндов 215](#_Toc471648358)

[5.1.4 Регистры процессора 223](#_Toc471648362)

[5.1.5. Система команд процессора 229](#_Toc471648363)

[5.2. Система команд микропроцессора Intel 8086 238](#_Toc471648368)

[5.2.1. Способы адресации и форматы команд микропроцессора I 8086 238](#_Toc471648369)

[6.1. Два уровня распараллеливания 251](#_Toc471648370)

[6.2. Классификация параллельных ВС. 253](#_Toc471648371)

[6.1.2 «Фон-Неймановские» и «не-Фон-Неймановские» архитектуры 255](#_Toc471648372)

[6.1.3. Системы с общей и распределенной памятью 258](#_Toc471648373)

[6.2. Способы межмодульного соединения (комплексирования) 260](#_Toc471648374)

[6.3.1. Мультимикропроцессорные вычислительные системы 262](#_Toc471648375)

[6.4. Направление «мини-супер» призвано поддержать персональный компьютер 265](#_Toc471648376)

[6.5. Распределенный и разделяемый вычислительный ресурс второго уровня. Решающие поля 268](#_Toc471648377)

[6.6. Способы распараллеливания 271](#_Toc471648378)

[6.7. Векторные конвейеры. «Зацепление» векторов 279](#_Toc471648380)

[6.8. Выполнение операций на стеке 281](#_Toc471648381)

## ПРЕДИСЛОВИЕ

Учебно-методическое пособие по дисциплине «Арифметико-логические основы цифровых вычислительных машин и архитектура компьютеров» предназначено для обучения студентов по одноименной дисциплине, а также дисциплине «Компьютерные системы и сети».

Отметим, что задачи, связанные с применением арифметических и логических основ ЦВМ, возникают в самых различных сферах информационных технологий. Сюда можно отнести системы автоматизированного проектирования (САПР), в которых осуществляется синтез цифровых узлов вычислительных систем на базе законов алгебры логики и логико-комбинаторного подхода к синтезу структуры сложных объектов, оптимизация синтезированных вариантов объекта, выбор из множества альтернативных вариантов структуры проектируемого объекта оптимальных. Широко используются методы, основанные на алгебре логики, при синтезе элементов и устройств вычислительной техники различного назначения, синтезе математических моделей для информационных систем, синтезе алгоритмов. Важную роль эти методы играют при проектировании систем управления и обработки информации, встроенных систем, других сложных технических систем, в том числе и издательско-полиграфических комплексов. Логичным продолжением арифметико-логических основ ЦВМ являются разделы по архитектурной организации компьютерных систем. Материалы по данной тематике содержатся в фундаментальных изданиях отечественных и зарубежных авторов, однако, большинство из этих литературных источников стало библиографической редкостью или является недоступным широкому кругу обучающихся. В данной связи издание пособий по арифметико-логическим основам ЦВМ и архитектуры компьютерных систем является актуальным.

Первые разделы курса, вынесенного в название данного учебно-методического пособия, изложены в электронном средстве обучения «Кобайло А.С. Арифметические и логические основы цифровых вычислительных машин: учебно-методическое пособие для студентов заочной формы обучения специальности «Информационные системы и технологии (издательско-полиграфический комплекс)», изданного в 2014 г.; схемотехнические основы ЦВМ как цифровой логический уровень компьютера рассматриваются в конспекте лекций Кобайло А. С. по дисциплине «Арифметико-логические основы цифровых вычислительных машин и архитектура компьютеров». В данном учебно-методическом пособии содержатся основные сведения по важнейшим разделам соответствующих дисциплин, не вошедшим в указанные издания.

Пособие состоит из двух разделов:

– организация микропроцессорной системы;

– параллельные компьютерные системы.

В первом разделе рассматриваются вопросы организации компьютерных систем, архитектура ряда современных микропроцессоров, форматы и методы передачи данных в микропроцессорных системах, построение интерфейсов микропроцессорных систем, организация прерываний и прямого доступа к памяти.

Во втором разделе рассматриваются такие вопросы как классификация параллельных вычислительных систем (ВС), различные типы архитектур, организация памяти, способы комплексирования в ВС.

Содержание пособия соответствует важнейшим разделам учебных программ учреждения высшего образования по указанной учебной дисциплине «Арифметико-логические основы цифровых вычислительных машин и архитектура компьютеров» для студентов специальностей 1-40 05 01-03 – «Информационные системы и технологии (издательско-полиграфический комплекс)» и 1-98 01 03 – «Программное обеспечение безопасности мобильных систем» и учебной дисциплине «Компьютерные системы и сети» для студентов специальности 1-40.01 01 – «Программное обеспечение информационных технологий».Содержание пособия полностью соответствует содержанию одноименных тем учебной программы

Изучение студентами материала пособия будет способствовать углублению и расширению ими знаний по архитектурной организации компьютерных систем и окажет несомненную помощь при изучении практически всех специальных дисциплин согласно учебным планам соответствующих специальностей.

**1. Арифметические основы цифровых вычислительных машин**

* 1. **Системы счисления**
     1. **Понятие системы счисления**

В своей повседневной деятельности человек использует различные системы счисления, к которым относятся десятичная, римская, система исчисления времени и т. д. Все их можно подразделить на позиционные и непозиционные.

В непозиционных системах счисления «доля» цифры или ее вес в количественном измерении записанного числа не зависит от местоположения данной цифры в записи этого числа. Типичным примером такой системы счисления является римская. В ней используются цифры:

I V X L C D M – римские цифры

1 5 10 50 100 500 1000 – их десятичные эквиваленты

При количественной оценке числа его значение определяется как сумма значений цифр, составляющих запись числа, кроме пар, состоящих из цифры меньшего веса, предшествующей цифре большего веса, значение которой определяется как разность веса большей и меньшей цифр.

*Пример.*

Значение числа МММСМLIХ определяется как сумма:

1000 + 1000 + 1000 + (1000 *-* 100) + 50 + (10 *-* 1),

что соответствует десятичному эквиваленту 3959.

Количественная оценка числа, записанного в позиционной системе счисления, определяется как сумма произведений значения цифр, составляющих запись числа, умноженных на вес позиции, в которой располагается цифра.

Примером такой системы счисления является широко используемая десятичная система счисления.

*Пример.*

Количественная оценка десятичного числа 395910 определяется как

3 ⋅ 1000 + 9 ⋅ 100 + 5 ⋅ 10 + 9 ⋅ 1,

где 1000, 100, 10, 1 – соответственно веса четвертого, третьего, второго, первого разрядов записи оцениваемого числа.

*Десятичная* *система счисления* является также системой с равномерно распределенными весами, которые характеризуются тем, что соотношение весов двух любых соседних разрядов имеет для такой системы одинаковое значение. Это соотношение называется основанием системы счисления, которое далее будем обозначать как «*q*».

Общая запись числа в системе с равномерно распределенными весами имеет вид

*Nq = Аn Аn–*1 *... А*2 *А*1 *А*0*.*(1.1)

Значение такого числа определяется как

*Nq = Аn ⋅ qn + Аn–*1 *⋅ qn–*1 *+ Аn*–2 *⋅ qn*–1 *+ … А*2 *⋅ q*2 *+ А*1 *⋅ q*1 *+ А*0 *⋅ q*0, (1.2)

где *Аi* – цифра записи числа, удовлетворяющая условию

0 ≤ *Аi* ≤ (*q -* 1),

где *q* – основание системы счисления.

При *q* =10 *А* изменяется в диапазоне от 0 до 9.

Запись числа *N* в виде (1.1) называется кодированной, а запись в форме (1.2) – расширенной.

Помимо *q* = 10 (десятичная система счисления), возможны другие значения для основания системы счисления:

1. двоичная система счисления;
2. восьмеричная система счисления;
3. шестнадцатеричная система счисления и т. д.

В различных системах счисления в качестве цифр используются обозначения соответствующих цифр десятичной системы счисления – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, а в случае, когда десятичных цифр «не хватает» (для систем счисления с основанием *q*, большим чем 10), для цифр, превышающих 9, вводятся дополнительные обозначения, например, для *q* = 16 это будут обозначения А, В, C, D, E, F, которые соответствуют шестнадцатеричным цифрам (десятичные эквиваленты их равны соответственно 10, 11, 12, 13, 14, 15).

В связи с тем, что в дальнейшем изложении будут использоваться различные системы счисления, примем такое обозначение:

*Nq*– число *N*, представленное в системе счисления с основанием *q*.

***Примеры записи чисел в различных системах счисления***:

*N*2= 10011011 = 1 *⋅* 27 + 0 *⋅* 26 + 0 *⋅* 25 + 1 *⋅* 24 + 1 *⋅* 23 + 0 *⋅* 22 +   
+ 1 *⋅* 21 + 1 *⋅* 20;

*N*8 = 471025 = 4 *⋅* 85 + 7 *⋅* 84 + 1 *⋅* 83 + 0 *⋅* 82 + 2 *⋅* 81 + 5 *⋅* 80;

N16 = 84FE4A = 8 ⋅ 165 + 4 ⋅ 164 + F ⋅ 163 + E ⋅ 162 + 4 ⋅ 161 +   
+ A ⋅ 160;

*N*10= 35491 = 3 *⋅* 104 + 5 *⋅* 103 + 4 *⋅* 102 + 9 *⋅* 101 + 1 *⋅* 100.

На основании вышеизложенного можно заключить, что запись одного и того же числа в различных системах счисления будет тем длиннее, чем меньше основание системы счисления. Например,

*N* = 206310 = 1000000011112 = 40178 = 80F16.

При работе с различными системами счисления полезно помнить соотношения, приведенные в табл. 1.1 и 1.2.

Таблица 1.1

**Соответствие показателя степени двоичного числа значению**

**десятичного числа**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *n* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 2*n* | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 |

Таблица 1.2

**Соответствие символов различных систем счисления**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| №  п/п | *q* = 2 | *q* = 8 | *q* = 16 | *q* = 10 | Десятичный  эквивалент | Двоичный  Эквивалент |
| 1 | 0 | 0 | 0 | 0 | 0 | 0000 |
| 2 | 1 | 1 | 1 | 1 | 1 | 0001 |
| 3 | – | 2 | 2 | 2 | 2 | 0010 |
| 4 | – | 3 | 3 | 3 | 3 | 0011 |
| 5 | – | 4 | 4 | 4 | 4 | 0100 |
| 6 | – | 5 | 5 | 5 | 5 | 0101 |
| 7 | – | 6 | 6 | 6 | 6 | 0110 |
| 8 | – | 7 | 7 | 7 | 7 | 0111 |
| 9 | – | – | 8 | 8 | 8 | 1000 |
| 10 | – | – | 9 | 9 | 9 | 1001 |
| 11 | – | – | А | – | 10 | 1010 |
| 12 | – | – | B | – | 11 | 1011 |
| 13 | – | – | C | – | 12 | 1100 |
| 14 | – | – | D | – | 13 | 1101 |
| 15 | – | – | E | – | 14 | 1110 |
| 16 | – | – | F | – | 15 | 1111 |

Человек в своей практической деятельности наиболее часто использует *десятичную* систему счисления. *Двоичная система счисления* является удобной для обработки информации в ЭВМ. Промежуточное место между ними занимает *двоично*-*десятичная система* счисления, которая, в принципе, является десятичной, но отдельные десятичные цифры в ней записываются в виде набора двоичных разрядов. Существуют разныедвоично-десятичные системы. Например, десятичное число 804714 в двоично*-*десятичной системе представляется в виде 1000 0000 0100 0111 0001 0100. В дальнейшем для сокращения будем использовать название «двоично-десятичная система», имея в виду двоично-десятичную систему 8, 4, 2, 1. Цифры в обозначении разновидности двоично-десятичной системы указывают веса соответствующих двоичных разрядов.

* + 1. **Перевод чисел из одной системы счисления в другую**

Наличие различных систем счисления предполагает использование способов перевода записи числа из одной системы в другую. Для этой цели применяются следующие методы преобразований:

1. преобразования с использованием весов разрядов в исходной и в искомой записи числа;
2. деления (умножения) на новое основание;
3. с использованием особого соотношения заданной и искомой систем счисления.

***Метод преобразования с использованием весов разрядов***

Метод преобразования с использованием весов разрядов записи числа в исходной и в искомой системах предполагает применение расширенной записи числа (1.2) в некоторой системе счисления.

Метод имеет две разновидности в зависимости от того, какая система счисления (исходная или искомая) является более привычной. Если более привычной является искомая система, то на основании расширенной записи исходного числа подсчитываются значения ее отдельных разрядов в новой системе счисления. Далее полученные значения суммируются. Например, при преобразовании целого двоичного числа *N2* = 110011010 в десятичную систему счисления исходное число представляется в расширенной записи *N* = 28 + 27 +  
+ 24 + 23 + 21 и рассчитывается вес отдельных (ненулевых) двоичных разрядов в десятичной системе счисления:

256, 128, 16, 8, 2.

Затем искомая запись числа определяется как сумма весов всех ненулевых разрядов записи числа в заданной системе счисления:

256 + 128 + 16 + 8 + 2 = 410.

При преобразовании правильных дробей в принципе используется тот же подход, но при расчете весов отдельных разрядов берутся отрицательные степени основания счисления.

*Пример*.

Найти двоичный эквивалент десятичного числа:

43610 = \_\_\_\_?2.

*Решение*.

Первый (старший) разряд, имеющий значение 1 в искомой двоичной записи числа, будет разряд весом 28 = 256. С помощью остальных (младших) разрядов искомой записи числа необходимо представить значение 180 (180 – остаток, полученный как 436 *–* 256).

Второй разряд с весом 27 = 128 будет иметь в искомой двоичной записи числа значение 1. С помощью остальных (более младших) разрядов искомой записи числа необходимо представить значение 52 (остаток, полученный как 180 *-* 128).

Третий разряд с весом 26 = 64 будет иметь в искомой двоичной записи числа значение 0.

Четвертый разряд с весом 25 = 32 будет иметь в искомой двоичной записи числа значение 1, а остаток *–* 20.

Пятый разряд с весом 24 = 16 будет иметь в искомой двоичной записи числа значение 1, а остаток *–* 4.

Шестой разряд с весом 23 = 8 будет иметь в искомой двоичной записи числа значение 0.

Седьмой разряд с весом 22 = 4 будет иметь в искомой двоичной записи числа значение 1, а остаток *–* 0.

Восьмой разряд с весом 21 = 2 будет иметь в искомой двоичной записи числа значение 0.

Девятый разряд с весом 20 = 1 будет иметь в искомой двоичной записи числа значение 0.

Таким образом,

43610 = 1101101002.

*Пример*

Найти двоичный эквивалент числа 0,710 = ?2.

*Решение*

Предварительный результат ищется с точностью до пяти двоичных разрядов, причем пятый разряд используется только для округления при переходе к четырехразрядному окончательному результату.

Первый (старший) разрядом весом 2*-*1 = 0,5 искомой двоичной записи числа будет иметь значение 1. С помощью остальных (младших) разрядов искомой записи числа необходимо представить значение 0,2 (0,2 – остаток, полученный как 0,7 *-* 0,5 = 0,2).

Второй (старший) разряд с весом 2*-*2 = 0,25 в искомой двоичной записи числа будет иметь значение 0.

Третий разрядом с весом 2*-*3 = 0,13 в искомой двоичной записи числа будет иметь значение 1. С помощью остальных (более младших) разрядов искомой записи числа необходимо представить значение 0,07 (0,07 – остаток, полученный как 0,2 *-* 0,13).

Четвертый разрядом с весом 2*-*4 = 0,06 в искомой двоичной записи числа будет иметь значение 1, а остаток 0,01.

Пятый разряд с весом 2*-*5 = 0,03 искомой двоичной записи числа будет иметь значение 0.

Таким образом, десятичное число 0,710 = 0,101102.

После округления имеет место 0,710 = 0,10112.

***Метод деления (умножения) на новое основание***

Метод деления (умножения) имеет две разновидности соответственно для преобразования целых и дробных чисел.

***Преобразование целых чисел.*** Задачу представления числа *N*, заданного в системе *q*1, в системе счисления с основанием *q*2 можно рассматривать как задачу поиска коэффициентов полинома, представляющего собой расширенную запись числа *N* в системе счисления *q*2:

*Nq*1 *= а*0 *+ а*1 *⋅ q* 21 *+ а*2 *⋅ q*22 *+....+ аn-*2 *⋅ q*2 *n-*2*+ аn-*1 *⋅ q*2*n-*1*+*

*+ аn ⋅ q*2*n = Nq*2*.* (1.3)

Введем скобочную форму для выражения (1.3):

*Nq*1 = *Nq*2 = *а*0 + *q*2 (*а*1 +q2 (*а*2 +*q*2 (*а*3 + . +*q*2(*an* - 1 + *q*2(*аn*))…));

*Nn*

*N*(*n*-1)

*N*2

*N*1,

Обозначим выражение в первой скобке как *N*1, во второй скобке как *N*2, в третьей – как *N3* и т. д., выражение в (*n–*1)*-*й скобке – как *N*(*n–*1), выражение в *n-*й скобке – как *Nn*. Теперь, основываясь на выражении (1.3), можно утверждать, что при делении *Nq*1*/q*2 будет получена целая часть частного int(*Nq*1/*q*2) и остаток rest(*Nq*1/*q*2). Это можно записать:

*Nq*1/*q*2 → int(*Nq*1/*q*2) — целая часть частного *N*1, и остаток rest(*Nq*1/*q*2), равный *а*0.

Аналогично для остальных скобок:

*N1*/*q* 2 → int(*N*1/*q*2) равное *N*2 и остаток rest(*N*1/*q*2), равный а1;

*N*2/*q*2 → int(*N*2/*q*2) равное *N*3, и остаток rest(*N*2/*q*2), равный а2;

*N*(*n–*2)/*q*2 2 → int(*N*(*n–*2)/*q*1) равное *N*(*n–1)*, остаток rest(*N*(*n* = 2)/*q*1), равный а(*n–*2);

*N*(*n–1*)/*q*2 → int(*N*(*n–*1)/*q*2) = *Nn* = а*n* и остаток rest(*N*(*n–*1)/*q*2), равный   
*а*(*n–1*), при этом *Nn*< *q*2.

Отсюда вытекает правило формирования коэффициентов полинома (1.3) или разрядов записи заданного числа *N* в системе счисления с основанием *q*2:

* необходимо разделить исходное число *Nq*1 на новое основание *q*2, при этом получив целое частное и остаток;
* полученный остаток снова необходимо разделить на *q*2, процесс деления продолжается до тех пор, пока частное будет не меньше нового основания *q*2*.* Если очередное сформированное частное будет меньше, чем *q*2, то процесс формирования записи заданного числа в новой системе с основанием *q*2 считается законченным, а в качестве искомых разрядов новой записи числа используются результаты выполненных операций деления следующим образом:
* в качестве старшего разряда берется значение последнего частного, для остальных разрядов используются значения остатков в порядке, обратном порядку их получения.

*Пример*.

Найти запись в двоичной форме десятичного числа *N*10 = 436.

*Решение*.

Делим сначала исходное число *N*10, а затем получаемые частные на значение нового основания 2 до получения частного со значением, меньше чем 2:

436/2 → int(436/2) = 218 и rest (436/2) = 0;

218/2 → int(218/2) = 109 и rest (218/2) = 0;

109/2 →int(109/2) = 54 и rest (109/2) = 1;

54/2 → int(54/2) = 27и rest (54/2) = 0;

27/2 → int(27/2) = 13 и rest (27/2) = 1;

13/2→int(13/2) = 6 и rest (13/2) = 1;

6/2 → int(6/2) = 3 и rest (6/2) = 0;

3/2 → int(3/2) = 1 и rest (3/2) = 1.

Таким образом: 436 = 11 0110100.

***Преобразование дробных чисел***

Задача представления дробного числа *Mq*1, заданного в системе *q*1, в системе счисления с основанием *q*2, можно рассматривать как задачу поиска коэффициентов полинома, представляющего собой расширенную запись числа *M* в системе счисления *q*2:

*B*1 *⋅ q*2*–*1 *+ B*2 *⋅ q*2*–*2 *+ B*3 *⋅ q*2*–*3 *+ ....* *+ Bn–*2 *⋅ q*2*–(n–*2*) + B n–*1 *⋅ q*2*–(n–*1*) +*

+ *Bn ⋅ q*2*–n* = *Mq* (1.4)

Введем скобочную форму для выражения (1.4). Обозначим выражение в первой скобке как *M*1, во второй *–* как *M*2, в третьей *–* как *M*3 и т. д., выражение в (*n –* 1)*-*й скобке – как *M* *n–*1, выражение в *n-*й скобке – как *Mn*.

*Mq*2 *= Mq*1 *= q*2*–*1(*B*1 *+ q*2*–*1(*B*2 *+ q*2*–*1(*B*3*+...+q*2*–*1(*Bn–*1 *+ q2–*1(*Bn*))...))).

Число *Mq*1 *–* правильная дробь, поэтому при умножении *Mq*1 на *q*2 будет получено произведение, в общем случае состоящее из целой части int(*Mq*1 *⋅ q*2) и дробной части DF (*Mq*1 *⋅* q2). Использование введенных обозначений позволяет записать:

*Mq*1 *⋅ q*2 *=* (int(*M q*1 *⋅ q*2) *= B*1) *+* (DF(*Mq*1 *⋅ q*2) *= M*1),

аналогично для остальных скобок будем иметь следующее:

*M*1 *⋅ q*2 = (int(*M*1 *⋅ q*2) = *B*2) + (DF(*M*1 *⋅ q*2) = *M*2);

*M*2 *⋅ q*2 = (int(*M*2 *⋅ q*2) *= B*3) + (DF(*M*2 *⋅ q*2) *= M*3);

*M*3 *⋅ q*2 = (int*(M*3 *⋅ q*2) *= B*4) *+* (DF(*M*3 *⋅ q*2) *= M*4);

*Mn–*2 *⋅ q*2 = (int(*Mn*–2 *⋅ q*2) *= Bn–*1) *+* (DF*(Mn–*2 *⋅ q*2) *= Mn–*1);

*Mn–*1*⋅ q*2 *=* (int(*Mn–*1 *⋅ q*2) *= Bn*) *+* (DF(*Mn–*1 *⋅ q*2) *= Mn*);

*Mn ⋅ q*2 = (int(*Mn⋅ q*2) *= Bn+*1) *+* (DF(*Mn ⋅ q*2) *= Mn*+1).

Отсюда вытекает следующее правило формирования коэффициентов полинома, которые одновременно являются разрядами записи заданного числа *M* в системе счисления с основанием *q*2:

* определяется количество разрядов «*n*» в записи числа *Mq*2 в новой системе счисления;
* исходное число *Mq*1 умножается на *q*2, при этом будет получено смешанное число;
* дробная часть полученного произведения снова умножается на *q*2 и т. д.; процесс умножения повторяется *n* раз. В качестве искомых разрядов новой записи числа используются результаты выполненных операции деления следующим образом: в качестве первого старшего разряда искомой записи числа в новом основании берется значение целой части первого произведения, в качестве второго старшего разряда искомой записи числа в новом основании берется значение целой части второго произведения и т. д.

*Пример*.

Найти запись в двоичной форме десятичного числа *M*10 = 0,7.

*Решение*.

Определяем количество разрядов числа *M*2. Так как исходная запись числа содержит один десятичный разряд, то запись данного числа в двоичном основании должна содержать четыре разряда. Учитывая округление, ищется предварительный двоичный эквивалент с пятью разрядами.

Умножаем исходное число *M*10, а затем дробные части последовательно получаемых произведений на новое основание 2. Выполняется пять таких операций умножения, в результате получаем:

0,7 *⋅* 2 = 1,4 (int(0,7 *⋅* 2) = 1 и DF(0,7 *⋅* 2) = 0,4);

0,4 *⋅* 2 = 0,8 (int(0,4 *⋅* 2) = 0 и DF(rest (0,4 *⋅* 2) = 0,8);

0,8 *⋅* 2 = 1,6 (int(0,8 *⋅* 2) = 1 и DF(rest (0,8 *⋅* 2) = 0,6);

0,6 *⋅* 2 =1,2 (int(0,6 *⋅* 2) = 1 и DF(rest (0,6 *⋅* 2) = 0,2);

0,2 *⋅* 2 = 0,4 (int(0,2 *⋅* 2) = 0 и DF(rest (0,2 *⋅* 2) = 0,4).

Таким образом, 0,7 = 0,10110, а окончательный результат перехода в двоичную систему будет 0,710 = 0,10112.

***Метод с использованием особого соотношения оснований заданной и искомой систем счисления***

Данный метод применим в тех случаях, когда исходное *q*1 и новое *q*2 основания могут быть связаны через целую степень, т. е. когда выполняются условия: *q*1*m* = *q*2 (*условие 1*) или *q*2 *m* = *q*1 (*условие 2*).

Если имеет место *условие 2*, то для заданного в системе с основанием *q*1 числа *Nq*1 *= аn аn*-1 *аn*-2... *а*1*а*0 запись его в системе с новом основании *q*2 определяется следующим образом:

* каждому разряду *ai* исходной записи числа ставится в соответствие его *m-*разрядный эквивалент в системе счисления с основанием *q*2;

– искомая запись всего заданного числа формируется за счет объединения всех полученных *m*-разрядных групп.

Если имеет место *условие 1*, то запись заданного числа *N* =

= *anаn*-1*аn-*2...*а*1*а*0 в системе с новом основанием *q*2 формируется следующим образом:

– исходная запись числа разбивается на группы по *m* разрядов, двигаясь от точки вправо и влево (недостающие разряды в крайних группах (слева и справа) дополняются нулями;

– каждой полученной группе ставится в соответствие цифра новой системы счисления;

– искомая запись заданного числа в новой системе счисления образуется из цифр, соответствующих группам, на которые была разбита исходная запись.

*Пример*.

Найти двоичный эквивалент восьмеричного числа 67401.648.

*Решение.*

Основания исходной и новой систем счисления можно выразить через целую степень:

23 = 8.

Поэтому применяем третий метод для случая перехода из системы с большим основанием в систему с меньшим основанием. Ставим в соответствие каждой цифре исходной записи числа трехразрядный двоичный код (*триаду*):

6 7 4 0 1 6 4

110 111 100 000 001 110 100

Формируем окончательный результат посредством объединения полученных трехразрядных двоичных чисел в единый двоичный эквивалент:

67401.648 = 110111100000001.1101002.

*Пример*.

Найти шестнадцатеричный эквивалент двоичного числа

*N* = 11100101110110.1110110012.

*Решение*.

Основания исходной и новой систем счисления можно выразить через целую степень:

24 = 16.

Поэтому применяем третий метод для случая перехода из системы с меньшим основанием в систему с большим основанием. Разбиваем исходную запись числа на группы по четыре разряда (*тетрады*) вправо и влево от точки, в крайних левой и правой группах недостающие разряды заполняем нулями и каждой полученной группе из четырех разрядов ставим в соответствие цифру шестнадцатеричной системы счисления

0011 1001 0111 0110 . 1110 1100 1000

3 9 7 6 EC 8

Формируем окончательный результат посредством объединения полученных цифр в единый шестнадцатеричный эквивалент

11100101110110.1110110012 = 3976.EC816.

*Пример*.

Найти шестнадцатеричный эквивалент числа 67401.648, представленного в восьмеричной системе счисления.

*Решение*.

Основания исходной *q*1 и новой *q*2 систем счисления не могут быть связаны через целую степень, поэтому напрямую третий метод перехода неприменим. Однако существует система с двоичным основанием, для которой допустим третий метод перехода и восьмеричную (исходную для данного примера), и в шестнадцатеричную (новую систему для данного примера) системы счисления, т. к. 23 = = 8 и 24 = 16.

Поэтому в данном случае для решения поставленной задачи целесообразно использовать два быстрых перехода из восьмеричной системы счисления в двоичную (промежуточную), а затем из двоичной системы счисления в шестнадцатеричную третьим методом. Это будет гораздо быстрее, чем использовать для заданного преобразования первый или второй метод.

Таким образом, поставленная задача решается в следующем порядке:

67401.648 = 110 111 100 000 001 . 110 1002;

110 111 100 000 001 . 110 1002 = 0110 1111 0000 0001. 1101 00002 = 6 F 0 1. D 016, т. е.:

67401.648 = 6F01.D016.

*Пример*.

Найти двоичный эквивалент числа 674010.

*Решение*.

Основания исходной *q*1 и новой *q*2 систем счисления не могут быть связаны через целую степень, поэтому третий метод перехода неприменим. В принципе здесь целесообразно использовать второй метод – метод деления на новое основание. Однако в этом случае потребуется большое количество операций деления на два. Для сокращения количества операций деления может оказаться целесообразным решить эту задачу за счет перехода с использованием второго метода в промежуточную шестнадцатеричную систему счисления, а затем, используя третий метод, быстро перейти в заданную двоичную систему счисления:

1. выполняем переход в промежуточную систему счисления:

6740/16 = 421 (остаток 4);

421/16 = 26 (остаток 5);

26/16 = 1 (остаток 10);

1. в промежуточной системе счисления имеем:

674010 = 1А5416,

1. выполняем переход из промежуточной системы счисления в заданную:

1А5416 = 0001 1010 0101 01002.

Как видно из вышеприведенного, для заданного перехода потребовалось выполнить только 3 операции деления на 16 вместо 13-ти операций деления на 2.

***Определение разрядности двоичных кодов***

Рассмотрим основные соотношения, которые могут быть использованы для определения разрядности кода при переводе чисел из одной системы счисления в другую, в частности, при переводе записи числа в десятичной системе в ее двоичный эквивалент.При переводе целых чисел проблем,как правило, не возникает, т. к. перевод методом деления числа, представленного в исходной системе счисления, на новое основание автоматически дает нужный результат, апри использовании метода перевода с использованием весов разрядов достаточно знать целые степени двойки, при этом в общем случае используется соотношение

,

где *n*– количество разрядов двоичного кода числа *A*;

*A –* запись преобразуемого числа в десятичной системе счисления;

 – наименьшее целое, превышающее значение выражения, расположенного в скобках.

При преобразовании правильных дробей или смешанных чисел в простейшем случае может быть использовано положение, основанное на потребности для представления одного десятичного символа четырехразрядного двоичного эквивалента (тетрады). Однако, для общих случаев требуется более строгая методика определения или расчета разрядности дробной части двоичного числа. Такая методика вытекает из метода преобразования правильной десятичной дроби в двоичный эквивалент умножением числа в исходной системе счисления на новое основание и основывается на учете веса очередного десятичного разряда дробного числа в формировании отличного от нуля (единичного) двоичного разряда.Так, для преобразования одного десятичного разряда после точки требуется 4 шага умножения, формируя 4 двоичных разряда, 2-х разрядов – семь шагов, 3-х разрядов – 10 шагов и т.д. С увеличением разрядности преобразуемой десятичной дроби трудоемкость процедуры определения искомой разрядности увеличивается. Для устранения этогонеудобства возможно использование предварительно рассчитанной разрядности двоичных эквивалентов дробной части смешанных чисел или правильных дробей, представленных в десятичной системе счисления, т. е. фактически – значения показателя степени *n* десятичного числа 2*n*, разрядность которого по сравнению с разрядностью 2*n* – 1 увеличена на единицу.

Очевидно, что для перевода *k*-разрядной правильной десятичной дроби для получения первого ненулевого значения двоичного эквивалента в экстремальном случае необходимо выполнить 2*n*шагов умножения, где 2*n* – ближайшая целая степень двойки, превышающая верхнюю включенную границу диапазона представления*k*-разрядного десятичного числа. Тогда для формирования общего выражения для вычисления разрядности двоичной правильной дроби 0. *A*2используем развернутую запись верхней включенной границы *A*10maxисходного представления*A*10 как целого *k*-разрядного десятичного числа:

*A*10max = 9 ∙ 100 + 9 ∙ 101 + 9 ∙ 102 + … + 9 ∙ 10*k*– 1 = .

Зависимость между диапазоном представления числа *A*10и разрядностью *n*правильной двоичной дроби 0. *A*2 будет следующей:



Отсюда искомая разрядность *n*определится из выражения



**1.2. Двоичная арифметика**

**1.2.1. Операция сложения в двоичной системе счисления**

При выполнении любой операции результат ищется согласно соответствующим правилам, которые удобно представлять в табличной форме, где для всех возможных комбинаций значений одноразрядных операндов приводятся значения результата.

***Правила сложения в двоичной системе счисления***

|  |  |  |
| --- | --- | --- |
| + | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0*\** |

Все возможные значения первого слагаемого задаются во второй и третьей строках первой колонки; все возможные значения второго слагаемого – во второй и третьей колонках первой строки. На пересечении отмеченных значениями операндов строк и колонок располагается результат их сложения. В таблице знаком «*\**» отмечен случай, когда в текущем разряде результата получен ноль и имеет место перенос в ближайший старший разряд.

*Пример*.



В общем случае при формировании значения в текущем разряде результата приходится дважды применять приведенную таблицу сложения: первый раз при сложении соответствующих разрядов операндов, формируя так называемую поразрядную сумму, и второй раз *–* при сложении разряда сформированной поразрядной суммы и переноса, пришедшего из ближайшего младшего разряда.

При машинной реализации операции сложения сначала формируется поразрядная сумма операндов без учета переноса, далее формируется код переноса и затем с помощью специальных логических цепей учитываются возникшие переносы. При этом перенос, возникший в некотором разряде, может изменить не только ближайший старший разряд, но и целую группу старших разрядов. В худшем случае перенос, возникший в самом младшем разряде, может изменить значение старших разрядов сформированной поразрядной суммы вплоть до самого старшего.

При формировании поразрядной суммы и учете возникших переносов используется следующая классификация разрядов складываемых операндов:

* разряд, генерирующий перенос (оба операнда в этом разряде имеют «1»);
* разряд, пропускающий перенос (операнды в этом разряде име­ют разные значения);
* разряд, блокирующий распространение переноса (операнды в этом разряде имеют одинаковые значения).
  + 1. **Операция вычитания**

Правила вычитания представлены ниже в форме таблицы.

***Правила вычитания в двоичной системе счисления***

|  |  |  |
| --- | --- | --- |
| - | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1*\** | 0 |

Все возможные значения вычитаемого задаются во второй и третьей строках первой колонки; все возможные значения уменьшаемого – во второй и третьей колонках первой строки. На пересечении отмеченных значениями операндов строк и колонок располагается результат вычитания второго операнда из первого. В таблице знаком «*\**» отмечен случай, когда в текущем разряде результата получена единица при займе из ближайшего старшего разряда.

*Пример.*



Как видно из таблицы и приведенного примера, реализация операции вычитания не сложнее операции сложения.

В ЭВМ никогда в перечне выполняемых операций арифметического устройства не присутствует одновременно операция сложения и операция вычитания. При этом, как правило, присутствует только операции сложения. Что же касается операции вычитания, то она реализуется за счет прибавления к уменьшаемому значения вычитаемого, взятого с противоположным знаком.

* + 1. **Операция умножения**

Умножение в двоичной системе счисления задается в табличной форме.

***Правила умножения в двоичной системе счисления***

|  |  |  |
| --- | --- | --- |
| \* | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 0 | 1 |

Все возможные значения множимого задаются во второй и третьей строках первой колонки; все возможные значения множителя – во второй и третьей колонках первой строки. На пересечении отмеченных значениями операндов строк и колонок располагается результат умножения первого операнда на второй.

При умножении многоразрядных операндов, как правило (особенно в десятичной системе счисления), используется метод, при котором формирование произведения выполняется за счет суммирования частичных произведений, которые оформляются посредством умножения множимого на отдельные разряды множителя с учетом веса соответствующего разряда множителя.

Таблица умножения одноразрядных операндов в двоичной системе существенно упрощает задачу формирования частичного произведения:

– частичное произведение для разряда множителя равняется нулю, если этот разряд равен нулю;

– частичное произведение для разряда множителя равняется множимому, взятому с соответствующим весом, если разряд множителя равен единице.

При последовательном способе формирования частичных произведений последние могут рассчитываться поочередно для отдельных разрядов множителя начиная с младшего или старшего разряда. При десятичном основании, как правило, формирование частичных произведений осуществляется начиная с младшего разряда множителя.

*Пример*.

Найти произведение двоичных чисел 1011 и 1101, начиная формирование частичных произведений со старшего разряда множителя.

*Решение*.

При формировании частичных произведений, начиная со старшего разряда множителя, процесс формирования произведения заданных операндов можно представить следующим образом:

1 0 1 1

×1 1 0 1

1 0 1 1

+ 1 0 1 1

+ 0 0 0 0

+ 1 0 1 1

1 0 0 0 1 1 1 1

При реализации умножения рассматриваемым способом требуется использование *n*-разрядного регистра для хранения множимого, 2*n*-разрядного сумматора для подсчета промежуточных и конечного произведения и 2*n*-разрядного сдвигающего регистра для хранения множителя.

*Пример.*

Найти произведение двоичных чисел 1011 и 1101, начиная формирование частичных произведений с младшего разряда множителя и применяя учет сформированных частичных произведений по мере их формирования.

*Решение.*

1 0 1 1

× 1 1 0 1

0 0 0 0

+ 1 0 1 1

1 0 1 1

+ 0 0 0 0\_

0 1 0 1 1

+ 1 0 1 1\_\_\_

1 1 0 1 1 1

+ 1 0 1 1\_\_\_\_\_

1 0 0 0 1 1 1 1

Реализация данного метода умножения требует использовать 2*n*-разрядный сумматор для последовательного, от такта к такту, формирования 2*n*-разрядного произведения и 2*n*-разрядный регистр для хранения и сдвига влево множимого.

В данном примере для того, чтобы учесть то, что очередной разряд множителя имеет вес в два раза больший, чем предыдущий разряд, его частичное произведение учитывается со сдвигом множимого на один разряд влево при суммировании с промежуточным результатом. В таком случае говорят, что умножение выполняется со сдвигом множимого.

Однако для учета того, что очередное частичное произведение имеет вес, в два раза больший веса предыдущего частичного произведения, можно при суммировании сдвигать вправо промежуточный результат. В таком случае говорят: умножение выполняется со сдвигом промежуточного результата. При использовании данного подхода умножение чисел 1101 и 1011 представляется в виде следующих действий:

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| 1 разряд  2 разряд  3 разряд  4 разряд |  | множимое  множитель  промежуточный результат  (начальное значение)  1 результат сдвига  1  11 результат сдвига  11  111 результат сдвига  111  1111 результат сдвига |

В первой колонке приведены номера отрабатываемых разрядов множителя начиная с младшего. Эти номера отмечают строки, в которых учитывается частичное произведение, соответствующее этому разряду множителя. В этой же первой колонке расположены единицы переполнения, возникающие при суммировании промежуточного результата и очередного частичного произведения, сформированного для соответствующего отрабатываемого разряда множителя (в данном случае единица переполнения имеется при отработке четвертого, самого старшего, разряда множителя).

Вторая колонка отражает длину основной разрядной сетки (*n* = 4).

В третьей колонке представлены разряды промежуточных и конечного произведений, «вытолкнутых» за пределы основной разрядной сетки в процессе выполнения очередного сдвига промежуточного произведения.

Из приведенного примера видно, что выталкиваемые за пределы разряды промежуточных произведений в дальнейшем не изменяются и их значение не влияет на значение суммы, формируемой в пределах основной разрядной сетки. Поэтому для реализации этого метода умножения требуется *n-*разрядный сумматор, обеспечивающий суммирование только в пределах основной разрядной сетки.

Аналогично умножению начиная с младшего разряда множителя, при умножении со старших разрядов можно заменить сдвиг вправо множителя на сдвиг влево промежуточного произведения.

Операция умножения в общем случае дает точный результат – 2*n-*разрядное произведение, где *n-*разрядность операндов.

В ЭВМ при выполнении различных операций, в том числе и операции умножения, разрядность операндов и результатов одинаковая. Это означает, что при умножении правильных дробей последние   
(младшие) *n* разрядов отбрасываются, а старшие *n* разрядов округляются с учетом отбрасываемых младших. Например, произведение 0,1011 × 0,1101 = 0,10001111 представляется в *n-*разрядном варианте как 0,1001. В этом случае операция умножения считается приближенной.

Возможные методы реализации операции умножения можно классифицировать по двум признакам:

– начиная с какого разряда (со старшего или младшего) выполняется отработка множителя;

– что сдвигается – множимое или промежуточное произведение.

Используя эти два классификационных признака, можно выделить четыре метода умножения:

*– умножение с младших* разрядов множителя со сдвигом множимого; при реализации данного метода требуется 2*n*-разрядный сумматор, 2*n*-разрядный регистр промежуточного произведения, 2*n*-разрядный регистр для хранения и сдвига множимого и *n*-разрядный регистр для хранения множителя;

*– умножение с младших* разрядов множителя со сдвигом промежуточного произведения; при реализации данного метода требуется *n*-разрядный сумматор, 2*n*-разрядный регистр промежуточного произведения, *n*-разрядный регистр для хранения множимого и *n*-разрядный регистр для хранения множителя;

*– умножение со старшего* разряда множителя со сдвигом множимого; при реализации данного метода требуется 2*n*-разрядный сумматор, 2*n*-разрядный регистр промежуточного произведения, 2*n*-разрядный регистр для хранения и сдвига множимого и *n*-разрядный регистр для хранения множителя;

*– умножение со старшего* разряда множителя со сдвигом промежуточного произведения; при реализации данного метода требуется *n*-разрядный сумматор, 2*n*-разрядный регистр промежуточного произведения, *n*-разрядный регистр для хранения множимого и *n*-разрядный регистр для хранения множителя.

**1.2.4. Деление двоичных чисел**

Деление в принципе является неточной операцией, поэтому при её выполнении прежде всего устанавливается количество разрядов частного, которые подлежат определению.

Деление в двоичной системе счисления может выполняться точно так же, как и в десятичной, однако формирования частного двоичных операндов реализуется гораздо проще, чем в десятичной системе, т.к.:

1. упрощается процедура подбора очередной цифры вследствие того, что в двоичной системе очередной цифрой может быть одна из двух – либо 0, либо 1;
2. упрощается процедура умножения найденной цифры частного на делитель.

*Пример*.

Найти частное от деления двоичных чисел 0.1001 на 0.1101.

*Решение*.

По умолчанию считается, что разрядность результата и операндов одинаковая, поэтому окончательный результат должен иметь в данном случае 4 разряда. Учитывая необходимость округления, найдем дополнительный пятый разряд, на основании которого выполним округление.

0. 1 0 0 1 0.1101 – делитель

- 1 1 0 1 0.10110 – частное

1 0 1 0 0

- 1 1 0 1

1 1 1 0

- 1 1 0 1

1 – остаток

После округления получаем окончательный результат (частное): 0.1001 / 0.1101 = 0.1011.

**1.2.5. Арифметика с положительными двоично-десятичными числами**

В ЭВМ часто предусматривается обработка чисел не только в двоичной системе счисления, но в двоично-десятичной. При этом, как правило, стремятся реализовать двоично-десятичную арифметику по правилам двоичной с введением ограниченного количества коррекций.

Сложение двоично-десятичных чисел

Рассмотрим на конкретном примере реализацию этой операции.

*Пример*.

Найти сумму двух десятичных чисел с использованием двоично-десятичной системы счисления:

*A* = *D* + *C*,

где *D* = 3927;

*C* = 4856.

*Решение*.

Составляем двоично-десятичную запись для чисел *D* и *C*:

*D* =3927 = 0011 1001 0010 0111:

*C* = 4856 =0100 1000 0101 0110.

Найти значение *А* можно, реализовав следующую последовательность операций из двоичного сложения и операции коррекции:

\* \*\*

0011 1001 0010 0111 – *D*

+ 0100 1000 0101 0110 – *C*

1000 0001 0111 1101 – двоичная сумма

+ 0110 0110 – коррекция

1000 0111 1000 0011 – двоично-десятичная сумма

Для получения двоично-десятичной суммы *A* на основании результата сложения операндов по правилам двоичной арифметики необходимо добавить шестерку (0110) в те тетрады, из которых был перенос. В данном примере это вторая тетрада (отмечена \*). Необходимость такой коррекции обусловливается тем, что перенос, сформированный по правилам двоичного суммирования, унес из тетрады шестнадцать, а для десятичного сложения перенос должен был унести десять, т. е. перенос, сформированный по правилам двоичной арифметики, унес лишнюю шестерку.

Кроме этого шестерка добавляется в те тетрады, в которых получено значение, большее девяти. Такая коррекция обуславливается тем, что по правилам десятичной арифметики в таких тетрадах должен быть выработан перенос и, чтобы его выработать по правилам двоичной арифметики, в тетраду нужно добавить шестерку. Для рассмотренного примера такой тетрадой является и четвертая тетрада (отмечена \*\*).

*Пример*.

Найти разность двух десятичных чисел с использованием двоично-десятичной системе счисления:

*A* = *C* - *D*,

где *D* = 3927,

*C* = 4856.

*Решение*

Составляем двоично-десятичную запись для чисел *D* и *C*:

*D* = 392710 = 0011 1001 0010 01112/10;

*C* = 485610 = 0100 1000 0101 01102/10.

Найти значение *В* можно, реализовав следующую последовательность операций из двоичного сложения и операции коррекции:

\* \*

0100 1000 0101 0110 – *C*

- 0011 1001 0010 0111 – *D*

0000 1111 0010 1111 – двоичная сумма

- 0110 0110 – коррекция

0000 1001 0010 1001 – двоично-десятичная сумма

Для получения двоично-десятичной разности *A* на основании результата вычитания операндов по правилам двоичной арифметики необходимо вычесть шестерку (0110) из тетрад, в которые пришел заем. Это обусловливается тем, что заем, сформированный по правилам двоичного вычитания, приносит в тетраду шестнадцать, а для десятичного сложения заем должен был принести в тетраду десять, т. е. заем, сформированный по правилам двоичной арифметики, принес лишнюю шестерку. Для рассмотренного примера тетрадами, в которые пришел заем и в которых необходимо выполнить коррекцию (вычесть шестерку), являются вторая и четвертая тетрады (отмечены \*).

**1.3. Арифметика с алгебраическими числами**

**1.3.1. Кодирование алгебраических чисел**

Для представления чисел со знаком используются специальные коды:

– *прямой код*;

*– дополнительный код*;

*– обратный код*.

Во всех трех случаях используется следующий формат представления числа, содержащий два поля – поле знака и поле модуля (Рис 1.1).

|  |  |
| --- | --- |
| Поле знака | Поле модуля |

Рис. 1.1. Общий формат представления числа в ЭВМ

*Поле знака* представлено одним разрядом, в котором устанавливается 0, если число положительное, и 1, если число отрицательное.

*Поле модуля* отражает количественную оценку числа и для каждого кода формируется по*–*разному. Количество разрядов поля модуля определяется диапазоном изменения отображаемых чисел или точностью их представления.

В *прямом коде* запись целого числа *А* формируется по следующему правилу:



В *дополнительном коде* запись целого числа *А* формируется по следующему правилу:



где *n* – разрядность модульного поля;

*q* – основание системы счисления;

*qn* – максимальная невключенная граница диапазона изменения представляемых чисел, т. к. диапазон изменение чисел *А* определяется как *qn* > |*А*|≥ 0 .

Для случая правильной дроби запись числа *А* в дополнительном коде имеет вид:



где 1 – максимальная невключенная граница диапазона изменения представляемых чисел, т. е. диапазон изменение чисел *А* определяется как 1 > | *A* | ≥ 0.

В *обратном коде* запись целого числа *А* формируется по следующему правилу:



где *n* – разрядность модульного поля;

*q* – основание системы счисления;

(2*n* *-* 1) – максимальная включенная граница диапазона изменения представляемых чисел, т. е. диапазон изменения чисел *А* определяется как (*qn* *–* 1) ≥ | *A* | ≥ 0.

Для случая правильной дроби запись числа *А* в обратном коде имеет вид



где *n –* разрядность поля модуля;

1 *- q-n* – верхняя включенная граница представляемых чисел.

Т. о., диапазон изменения чисел *А* определяется с. о.:

(1 *– q-n*) ≥ | *А* | ≥ 0.

Легко показать, что перевод отрицательного числа из обратного или дополнительного кода в прямой выполняется по тем же правилам, что и перевод числа из прямого кода в обратный или *дополнительный*:

– для перевода отрицательного числа из обратного в *прямой код* необходимо дополнить его модуль до включенной границы;

– для перевода отрицательного числа из обратного в *прямой код* необходимо дополнить его модуль до невключенной границы.

**1.3.2. Дополнительный и обратный коды двоичных чисел**

При переводе двоичных чисел в качестве включенной и не включенной границы диапазона изменения абсолютных значений представляемых чисел используется соответственно 2*n* и 2*n* - 1.

Представление двоичных чисел в прямом и обратном кодах поясняется следующими примерами.

*Пример*.

Найди запись чисел *А* = 532 и *В* = -150 в прямом, дополнительном и обратном двоичных кодах.

*Решение*.

Найдем запись заданных чисел в двоичной системе:

*А* = 53210  = 10000101002, *В* = -15010 = -100101102.

Если считать, что представляются в заданных кодах только *А* и *В*, то разрядность *n* модульного поля должна соответствовать разрядности двоичной записи большего числа, т. е. *n* = 10.

Найдем запись заданных чисел в прямом коде:

[*A*]пк = 0.1000010100, [*B*]пк = 1.0010010110.

Найдем запись заданных чисел в дополнительном коде:

[*A*]дк = 0.1000010100,

[*B*]дк: для определения модульной части прибавим к невключенной границе диапазона (2*n* = 10000000000) число *В*:

10000000000 + (-10010110) = 1101101010.

Тогда [*B*]дк = 1.1101101010.

Найдем запись заданных чисел в обратном коде:

[*A*]ок = 0.1000010100,

[B]ок: для определения модульной части прибавим к включенной границе диапазона (2*n* - 1 = 1111111111) число *В*:

1111111111 + (-10010110) = 1101101001,

и тогда [*B*]ок = 1.1101101001.

Анализируя запись модульной части отрицательного числа *С* в обратном коде, можно заключить, что она представляет собой инверсию модульной части записи этого числа в прямом коде, т. е. 0 заменяется на 1, а 1 заменяется на 0. Отсюда вытекает правило формирования модуля обратного кода отрицательного двоичного числа.

*Для формирования модульной части записи отрицательного числа в обратном коде достаточно в модульной части записи этого числа в прямом коде взять обратные значения всех двоичных разрядов, т. е. необходимо проинвертировать модуль прямого кода.*

Переход от обратного кода отрицательного числа к представлению в прямом коде осуществляется по тому же правилу, т.е. необходимо проинвертировать модуль записи числа в обратном коде.

Если сравнить запись модульных частей дополнительного и обратного кодов отрицательного числа *В*, то можно заметить, что они отличаются на значение, соответствующее единицы младшего разряда. Отсюда вытекает правило формирования модуля дополнительного кода отрицательного числа.

*Для формирования модульной части записи отрицательного числа в дополнительном коде достаточно в модульной части записи этого числа в прямом коде взять обратные значения всех двоичных разрядов, т. е. необходимо проинвертировать модуль прямого кода и к полученному коду прибавить 1 в младший разряд.*

Переход от дополнительного кода отрицательного числа к прямому осуществляется по тому же правилу, т.е. необходимо проинвертировать модуль записи числа в дополнительном коде, и к полученному коду прибавить 1 в младший разряд.

При выполнении операций над числами со знаком в ЭВМ используются прямой, обратный и дополнительный коды. Как правило, информация в памяти хранится в прямом коде, а при выполнении операций применяется или обратный, или дополнительный код.

**1.3.3. Операции с двоичными числами в дополнительном коде**

При использовании *дополнительного* или о*братного* кода операция вычитания заменяется операцией сложения с изменением знака второго операнда. При сложении чисел, представленных в дополнительном коде, выполняется сложение разрядов, представляющих запись операндов, по правилам двоичной арифметики по всей длине записи чисел, не обращая внимание на границу, разделяющую знаковое и модульные поля. Переполнение знакового поля, т. е. перенос, возникший из крайнего левого разряда, игнорируется. В результате такого сложения будет получен *дополнительный код* суммы заданных операндов.

*Пример*.

Найти значения для *С*1, *С*2, *С*3, *С*4, определяемых выражениями:

*С*1 = *А* + *В*, *С*2 = *А* - *В*, *С*3 = *В* - *А*, *С*4 = -*А* - *В*,

если *А* = 5710, *В* = −21010. При выполнении операций использовать двоичный *дополнительный* код. Результат представить в *прямом* коде.

*Решение*.

Преобразуем заданные числа в двоичную систему счисления:

*А* = 5710 = 1110012, *В* = -21010 = -110100102.

Определим количество разрядов для модульной части записи чисел, учитывая не только значения используемых операндов, но и ожидаемые результаты выполнения заданных операций. Исходя из абсолютного значения операндов, разрядность представления модульной части *n* должна быть равна 8. Учитывая то, что в подлежащих реализации выражениях над числами выполняется только одна операция (или сложения, или вычитания), при этом возможно переполнение (возникновение переноса из старшего разряда), длину модульной части необходимо взять на один разряд больше, т. е. *n* = 9.

Избавляясь от операции вычитания, приводим заданные выражения к виду

*С*1 = *А* + *В*, *С*2 = *А* + (-*В*), *С*3 = *В* + (-*А*), *С*4 = (-*А*) + (-*В*).

Таким образом, в подлежащих реализации выражениях в качестве операндов присутствуют следующие величины: *A* , -*A*, *B*, -*B*. Представим их в прямом и дополнительном коде

[*А*]пк = 0.000111001, [*А*]дк = 0.000111001,

[−*А*]пк = 1.000111001, [-*А*]дк = 1.111000111,

[*В*]пк = 1.011010010, [*В*]дк = 1.100101110,

[−*В*]пк = 0.011010010, [-*В*]дк = 0.011010010.

Используя сформированный дополнительный код, реализуем выражения для *С*1,*С*2, *С*3, *С*4.

*С*1: 0.0 0 0 1 1 1 0 0 1 – [*A*]дк

+ 1.1 0 0 1 0 1 1 1 0 –[*B*]дк

1.1 0 1 1 0 0 1 1 1 – [*C*1]дк

1.0 1 0 0 1 1 0 0 1 – [*C*1]пк

*С*2: 0.0 0 0 1 1 1 0 0 1 – [*A*]дк

+ 0.0 1 1 0 1 0 0 1 0 – [-*B*]дк

0.1 0 0 0 0 1 0 1 1 – C2]дк = [*C*2]пк

*С*3: 1.1 1 1 0 0 0 1 1 1 – [-*A*]дк

+ 1.1 0 0 1 0 1 1 1 0 –[*B*]дк

~~1~~ 1.0 1 1 1 1 0 1 0 1 – [*C*3]пк

1.1 0 0 0 0 1 0 1 1 – [*C*3]пк..

При выполнении сложения в данном случае возникла единица переполнения знакового поля. При работе с дополнительным кодом она игнорируется (в примере она перечеркнута).

*С*4: 1.1 1 1 0 0 0 1 1 1 – [-*A*]дк

+ 0. 0 1 1 0 1 0 0 1 0 – [-*B*]дк

~~1~~ 0. 0 1 0 0 1 1 0 0 1 – [*C*4]дк = [*C*4]пк

В данном случае также возникло переполнение знакового поля, которое игнорируется.

* + 1. **Операции с двоичными числами в обратном коде**

При сложении чисел, представленных в *обратном* коде, выполняется сложение разрядов, представляющих запись операндов, по правилам двоичной арифметики по всей длине записи чисел, не обращая внимания на границу, разделяющую знаковое и модульные поля. Переполнение знакового поля, т. е. перенос, возникший из крайнего левого разряда, должен быть учтен как +1 в младший разряд полученной суммы. В результате такого сложения будет получен *обратный* код суммы заданных операндов.

*Пример*.

Найти значения для *С*1, *С*2, *С*3, *С*4, определяемых выражениями

*С*1 = *А* + *В*, *С*2 = *А* - *В*, *С*3 = *В* - *А*, *С*4 = -*А* - *В*,

если *А* = 5710, *В* = -21010. При выполнении операций использовать двоичный обратный код. Результат представить в прямом коде.

*Решение*

В данном примере используются те же выражения и те же операнды, что и в предыдущем примере, поэтому при его решение используются уже найденные ранее двоичные представления операндов и их прямые коды.

Обратные коды операндов имеют вид

[*А*]ок = 0.000111001, [-*А*]ок = 1.111000110,

[*В*]ок = 1.100101101, [-*В*]ок = 0.011010010.

Используя сформированный дополнительный код, реализуем выражения для *С*1, *С*2, *С*3, *С*4.

*С*1: 0.0 0 0 1 1 1 0 0 1 – [*A*]ок

+ 1.1 0 0 1 0 1 1 0 1 – [*B*]ок

1.1 0 1 1 0 0 1 1 0 – [*C*1]ок

1.0 1 0 0 1 1 0 0 1 – [*C*1]пк

*С*2: 0.0 0 0 1 1 1 0 0 1 – [*A*]ок

+ 0.0 1 1 0 1 0 0 1 0 – [-*B*]ок

0.1 0 0 0 0 1 0 1 1 – [*C*2]ок = [*C*2]пк

*С*3: 1.1 1 1 0 0 0 1 1 0 – [-*A*]ок

+ 1.1 0 0 1 0 1 1 0 1 – [*B*]ок

1 1.0 1 1 1 1 0 0 1 1

+ 1

1.0 1 1 1 1 0 1 0 0 – [*C*3]ок.

1.1 0 0 0 0 1 0 1 1 – [*C*3] пк

С4: 1.1 1 1 0 0 0 1 1 0 – [-A]ок

+ 0.0 1 1 0 1 0 0 1 0 – [-B]ок

1 0.0 1 0 0 1 1 0 0 0

+ 1

0. 0 1 0 0 1 1 0 0 1 – [C4]ок = [C4]пк

В данном случае также возникло переполнение знакового разряда, которое должно быть учтено как +1 в младший разряд сформированной суммы.

**1.3.5. Модифицированные коды**

При расчете разрядности *n* модульного поля весьма трудно бывает учесть диапазон значений результатов, особенно когда последовательность операции, представленных в подлежащих реализации выражениях, достаточно сложна.

При несоответствии выбранной разрядности *n* диапазону изменения представляемых чисел при выполнении операции сложении чисел с одинаковыми знаками возможно появление ситуации переполнения, когда подлежащий представлению результат выходит за диапазон представления, определенный некорректно выбранной разрядностью *n* поля модуля.

Например, в случае сложения двух чисел, представленных в обратном коде:

[*D*1]ок = 1.00110 и [*D*2]ок = 1.00110.

сумма этих чисел *F*1 = *D*1 + *D*2 будет подсчитана следующим образом:

*F*1:

1. 0 0 1 1 0

+1. 0 0 1 1 0

10. 0 1 1 0 0

+ 1

0. 0 1 1 0 1

Пример, выполненный по всем формальным правилам, дал «абсурдный» результат, так как получена положительная сумма двух отрицательных операндов. Аналогичная ситуация может возникать и при использовании дополнительного кода.

Ситуацию переполнения можно обнаруживать по факту появления «абсурдного» результата, но для этого необходимо помнить то, что в суммировании принимают участие операнды с одинаковыми знаками и знак полученного при этом результата отличен от знака операндов.

Более просто ситуация переполнения определяется при применении *модифицированного* кода (обратного или дополнительного). Модифицированные коды отличаются от базовых кодов только тем, что поле знака операндов имеет два разряда, и эти разряды имеют одинаковые значения:

00 – для положительных чисел;

11 – для отрицательных чисел.

Если в результате сложения чисел в модифицированном коде полученный результат имеет в поле знака одинаковые значения в обоих разрядах (00 или 11), то переполнения нет, если же разряды знакового поля имеют неодинаковые значения (10 или 01), то имеет место переполнение. При этом, если в поле знака имеет место значение 01 – результат положительный, а если 10, то полученный результат – отрицательный (основным носителем знака числа является левый разряд знакового поля).

*Пример*.

Найти значения выражений

*С*1 = *А* + *В*, *С*2 = *А* - *В*, *С*3 = *В* - А, *С*4 = -*А* - *В*,

используя *модифицированный* обратный код, если

[*А*]пк = 0.1010011,

[*B*]пк = 1.0111001.

*Решение*

*Модифицированный* обратный код для всех операндов, используемых в приведенных выражениях, имеет вид

[*А*]мпк = 00.1010011, [*А*]мок = 00.1010011, [-*А*]мок = 11.0101100,

[*B*]мпк = 11.0111001, [*B*]мок = 11.1000110, [-*B*]мок = 00.0111001.

Выполним действия, указанные в приведенных выражениях:

*С*1:

00.1010011 – [*А*]мок

+ 11.1000110 – [*B*]пок

100.0011001

+ 1

00.0011010 – [*С*1]мок = [*С*1]мпк

*С*2:

00.1010011 – [*А*]мок

+ 00.0111001 – [-*B*]пок

01.0001100

Результат положительный и имеет место переполнение.

*С*3:

11.0101100 – [-*А*]мок

+11.1000110 – [*B*]пок

110.1110010

+ 1

10.1110011

Результат отрицательный и имеет место переполнение.

*С*4:

11.0101100 – [-*А*]мок

+00.0111001 – [-*B*]пок

11.1100101 – [*С*4]мок

11.0011010 – [*С*4]мпк

При формировании *С*1 был получен положительный результат (без переполнения).

При формировании *С*4 был получен отрицательный результат (без переполнения).

Факт переполнения при формировании *С*3 и *С*2 устанавливается по наличию в разрядах знакового поля различных значений.

**1.4. Логические операции с двоичными кодами**

Над двоичными кодами могут выполняться различные логические операции, среди которых особое место занимают:

*– логическое суммирование* (обозначения – ИЛИ, ОR, «»);

*– логическое умножение* (обозначения – И, AND, «»);

*– отрицание* (обозначения – НЕТ, NOT, «**», т.е. штрих над отрицыемым *x*);

*– суммирование по модулю* 2 (обозначается mod 2, « »);

*– операции сдвига*.

**1.4.1. Логические операции**

Операция *логического суммирования* выполняется над двумя кодами и генерирует код той же разрядности, что и операнды, у которого в некотором *i*-м разряде находится единица, если хотя бы в одном операнде в *i*-м разряде имеет место единица.

*Пример.*

10001101  11110000 = 11111101.

Операция *логического умножения* выполняется над двумя кодами и генерирует код той же разрядности, что и операнды, у которого в некотором *i*-м разряде находится единица, если оба операнда в этом *i*-м разряде имеют единицу, и ноль во всех других случаях.

*Пример*.

 11110000 = 10000000.

Операция *суммирования по модулю 2* выполняется над двумя кодами и генерирует код той же разрядности, что и операнды, у которого в некотором *i*-м разряде находится единица, если два заданных операнда в *i*-м разряде имеют противоположные значения. Иногда эта операция называется «исключающее ИЛИ».

*Пример.*

10001101  11110000 = 01111101.

Операция *логического отрицания* выполняется над одним кодом и генерирует результирующий код той же разрядности, что и операнд, в некотором *i*-м разряде которого находится значение, противоположное значению в *i*-м разряде отрицаемого кода.

Операции *сдвига* в свою очередь, подразделяются на:

– логические сдвиги, которые имеют разновидности – сдвиг вправо, сдвиг влево, циклический сдвиг вправо, циклический сдвиг влево;

– арифметические сдвиги вправо и влево, выполнение которых зависит от знака и кода сдвигаемого числа.

**1.4.2. Логические сдвиги**

Сдвиг *влево* выполняется за счет установки в разряд значения, соответствующего исходному значению в ближайшем младшем разряде (освобождающийся самый правый т. е. самый младший, разряд заполняется 0, а «выталкиваемый» разряд пропадает). Например, код 11001110 после сдвига влево будет иметь вид 10011100.

Сдвиг *вправо* выполняется за счет установки в разряд значения, соответствующего исходному значению в ближайшем старшем разряде (в освобождающийся самый левый, т. е. самый старший, разряд заполняется 0, «выталкиваемый» разряд пропадает). Например, код 11001110 после сдвига влево будет иметь вид 01100111.

*Циклический сдвиг влево* выполняется за счет установки в разряд значения, соответствующего исходному значению в ближайшем младшем разряде (в освобождающийся самый правый, т.е. самый младший, разряд заносится значение старшего, т.е. самого левого разряда исходного кода). Например, код 11001110 после сдвига влево будет иметь вид 10011101.

*Циклический сдвиг вправо* выполняется за счет установки в разряд значения, соответствующего исходному значению в ближайшем старшем разряде (в освобождающийся самый левый т. е. самый старший, разряд заполняется значение в самом младшем разряде исходного кода). Например, код 11001110 после сдвига влево будет иметь вид 01100111.

**1.4.3. Арифметические сдвиги**

Арифметические сдвиги обеспечивают выполнение умножения (сдвиги влево) или операции деления (сдвиги вправо) двоичных кодов на два, точно так же, как сдвиги вправо и влево десятичного числа обеспечиваю выполнение деления и умножение на 10.

Арифметические сдвиги влево двоичного прямого кода выполняются в зависимости от того, какое сдвигается число – положительное или отрицательное.

Если *сдвигается положительное* число, то сдвиг (вправо или влево) выполняется как соответствующий логический сдвиг (влево или вправо), с той лишь разницей, что предусматриваются средства определения факта переполнения при сдвиге влево, что реализуется и при всех других арифметических операциях. При любом сдвиге вправо предусматриваются средства для округления после завершения нужного количества сдвигов и средства обнаружения обнуления сдвигаемой величины после очередного сдвига.

Арифметические сдвиги влево положительных двоичных чисел выполняются не зависимо от используемого кода (прямого, обратного, дополнительного). Его реализация иллюстрируются следующими примерами.

*Примеры*.

1. Найти результат арифметического сдвига влево на три разряда двоичного прямого кода числа [*А*]пк = 00.00000101

*Решение*.

Процесс выполнения заданного сдвига дает следующие промежуточные и конечное значения:

первый сдвиг: 00.00000101 ← 00.00001010;

второй сдвиг: 00.00001010 ← 00.00010100;

третий сдвиг: 00.00010100 ← 00.00101000.

1. Найти результат арифметического сдвига влево на четыре разряда двоичного прямого кода числа [*А*]пк = 00.00101.

*Решение*.

Процесс заданного сдвига дает следующие промежуточные и конечное значения:

первый сдвиг: 00.00101 ← 00.01010;

второй сдвиг: 00.01010 ← 00.10100;

третий сдвиг: 00.10100 ← 01.01000.

После третьего сдвига будет выработан сигнал переполнения, так как после очередного сдвига в разрядах знакового поля появятся разные значения. Таким образом, не считая процедуры определения переполнения, арифметический сдвиг влево выполняется точно так же, как и логический сдвиг влево.

3. Найти результат арифметического сдвига вправо на два разряда двоичного прямого кода числа [*А*]пк = 00.00000110.

*Решение.*

Процесс заданного сдвига дает следующие промежуточные и конечное значение:

первый сдвиг: 00. 00000110 → 00. 00000011;

второй сдвиг: 00. 00000011 → 00. 00000001;

После выполнения заданного количества сдвигов выполняется округление на основании последнего «вытолкнутого» разряда; в данном случае последний «вытолкнутый» разряд равен 1, поэтому конечный результат выполнения заданного сдвига будет равен 00.00000010.

4. Найти результат арифметического сдвига вправо на четыре разряда двоичного прямого кода числа [*А*]пк = 00.00000110.

*Решение*.

Процесс заданного сдвига дает следующие промежуточные и конечное значения:

первый сдвиг: 00.00000110 → 00.00000011;

второй сдвиг: 00.00000011 → 00.00000001;

третий сдвиг: 00.00000001 → 00.00000000.

После выполнения третьего сдвига будет выработан сигнал о получении нулевого результата. Оставшиеся сдвиги могут не выполняться.

Арифметические сдвиги отрицательных двоичных чисел, представленных в прямом коде

Арифметические сдвиги влево и вправо реализуются по-разному в зависимости как от знака числа, так и от используемого кода (прямого обратного, дополнительного).

При арифметическом сдвиге отрицательного двоичного числа, представленного в прямом коде, осуществляется соответствующий сдвиг только модульного поля записи числа.

Реализация этого типа сдвига иллюстрируется следующими примерами.

*Пример 1*.

Выполнить *арифметический сдвиг* влево двоичного числа *А* =

= 11. 001010 (соответствует 1010), представленного в модифицированном прямом коде.

*Решение.*

Заданный сдвиг, имеющий своей целью получение результата, в два раза превышающего по абсолютному значению значение исходного кода, дает в результате 11.010100 (2010), которое получается за счет логического сдвига влево *только модульной части* исходного кода.

Факт получения переполнения устанавливается по наличию единичного значения старшего разряда в сдвигаемом коде перед очередным сдвигом.

*Пример 2*.

Выполнить арифметический сдвиг вправо двоичного числа *А* = 11.01110 (1410), представленного в модифицированном прямом коде.

*Решение*.

Заданный сдвиг, имеющий своей целью получение кода, в два раза меньшего по абсолютному значению по отношению к значению исходного кода, дает в результате число 11.00111 (-710), которое получается за счет логического сдвига вправо только *модульной части* исходного кода.

При арифметическом сдвиге влево отрицательного двоичного числа, представленного в *обратном коде*, осуществляется циклический сдвиг исходного кода с контролем за переполнением, например, сдвиг влево отрицательного двоичного числа 11.1100110 (-2510), представленного в обратном коде, дает в результате 11.1001101 (-5010).

При арифметическом сдвиге вправо отрицательного двоичного числа, представленного в обратном коде, осуществляется сдвиг только модульной части записи числа с установкой единицы в освобождающийся разряд. При этом может осуществляется контроль за обнулением результата сдвига (появление единичных значений во всех разрядах) и округление результата после выполнения заданного количества сдвигов.

*Пример 3*.

Выполнить сдвиг вправо на четыре разряда двоичного числа 11.1001101 (десятичный эквивалент – -5010), представленного в обратном коде.

*Решение*.

Первый сдвиг дает 11.11001101 (-5010) → 11.11100110 (-2510).

Второй сдвиг дает 11.11100110 (-2510) → 11.11110011 (-1210).

Третий сдвиг дает 11.11110011 (-1210) → 11.11111001 (-610).

Четвертый сдвиг дает 11.11111001 (-610) → 11.11111100 (-310).

При выполнении сдвига вправо нечетного числа результат получается с точностью до младшего разряда кода.

После выполнения последнего, четвертого сдвига выполняется округление, при котором, если последний «вытолкнутый» разряд имел значение 0, к результату последнего сдвига прибавляется −1. Данное округление можно выполнить за счет прибавления единицы к прямому коду, соответствующему результату последнего сдвига исходного обратного кода.

В рассмотренном примере корректировать на единицу результат четвертого сдвига не надо, так как «вытолкнутый» разряд при последнем (четвертом) сдвиге равен единице. В данном случае конечный результат сдвига заданного отрицательного числа, представленного в обратном коде, равен 11.11111100.

При арифметическом сдвиге *влево* отрицательного двоичного числа, представленного в *дополнительном коде*, осуществляется логический сдвиг влево модуля исходного кода (освобождающийся разряд заполняется нулем) с контролем за переполнением, например, сдвиг влево отрицательного двоичного числа 11.11001110 (-5010), представленного в дополнительном коде, дает в результате 11.10011100 (-10010).

При арифметическом сдвиге *вправо* отрицательного двоичного числа, представленного в *дополнительном коде*, осуществляется логический сдвиг вправо модуля записи числа с установкой единицы в освобождающийся разряд. При этом может осуществляется контроль за обнулением результата сдвига (появление единичных значений во всех разрядах).

*Пример*.

Выполнить сдвиг вправо на четыре разряда двоичного числа 11.11001110 (десятичный эквивалент – -5010), представленного в дополнительном коде.

*Решение*.

Первый сдвиг дает 11.11001110 → 11.11100111 (-2510).

Второй сдвиг дает 11.11100111 → 11.11110011 (-1310).

Третий сдвиг дает 11.11110011 → 11.11111001 (-710).

Четвертый сдвиг дает 11.11111001 → 11.11111100 (-410).

При выполнении сдвига вправо нечетного целого числа результат получается с точностью до младшего разряда кода.

Арифметический сдвиг вправо может выполняться над *отрицательными числами с переполнением* (такие числа в модифицированном прямом, обратном или дополнительном коде имеют в знаковом поле 10). В этом случае после сдвига в знаковом поле будет 11, а в старшем разряде – 0, если число представлено в обратном или дополнительном коде, или 1, если число представлено в прямом коде.

*Пример 1*.

Выполнить сдвиг вправо на 2 разряда числа [*А*]пк = 10.01000110 (*А*10 = -326).

*Решение*.

1-й сдвиг: 10. 01000110 → 11.10100011 (-16310);

2-й сдвиг: 11.10100011 → 11.01010001 (-8110) и последний вытолкнутый разряд равен 1).

С учетом округления имеем окончательный результат:

[*А*2]пк=11.01010010.

*Пример 2*.

Выполнить сдвиг вправо на 2 разряда числа [*А*]ок = 10.10111001 (*А*10 = –326).

*Решение*.

1-й сдвиг: 10.10111001 → 11.01011100 (-6310);

2-й сдвиг: 11.01011100 → 11.10101110 (-8210).

*Пример 3*.

Выполнить сдвиг вправо на 2 разряда число [*А*]ок = 10. 10111010

(*А*10 – = -326).

*Решение*.

1-й сдвиг: 10.0111010 → 11.01011101 (-16310);

2-й сдвиг: 11.01011101 → 11.10101110 (-8110) и последний вытолкнутый разряд равен 1).

* 1. **Представление чисел в ЭВМ**

**1.5.1. Представление чисел с фиксированной точкой**

Числовая информация представляется в машине в форме с фиксированной или с плавающей точкой. При представлении с фиксированной точкой положение последней в записи числа фиксировано.

Как правило, при использовании фиксированной точки числа представляются в виде целого числа или правильной дроби, форматы которых приведены на рис. 1.1.

К заданному виду (целым числам или правильной дроби) исходные числа приводятся за счет введения масштабных коэффициентов.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Зн | 1р | 2р | 3р | 4р | .... | (*n*−1)р | *n*р | «.» |

*а*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Зн | «.» | 1р | 2р | 3р | 4р | .... | (*n*−1)р | *n*р |

*б*

Рис. 1.1. Формат числа: *а* – целого; *б –* дробного

Точка в записи числа не отображается, а так как она находится всегда в одном месте, то указание на ее положение в записи числа отсутствует. При *n*-разрядном представлении модульной части формат с фиксированной точкой обеспечивает диапазон изменения абсолютного значения числа *А*, для которого выполняется неравенство

2*n* > *A*≥ 0.

Одним из важнейших параметров представления чисел является ошибка представления. Ошибка представления может быть абсолютной ( или относительной (. Для формата с фиксированной точкой максимальные значения этих ошибок определяются следующим образом.

В случае целых чисел:

max = 0,5; max = max / *А*min = 0,5, где *А*min – минимальное, отличное от нуля, значение числа.

В случае дробных чисел:

max = 2-(*n*+1);*А*min**2−*n* ;max = max / *А*min = 2-(*n*+1) / 2−*n* = 0.5,

т.е. в худшем случае относительная ошибка при фиксированной точке может достигать сравнительно большого значения – 50%.

**1.5.2. Арифметические операции над числами, представленными в формате с фиксированной точкой**

К числу основных арифметических операций, непосредственно реализуемых в ЭВМ, относятся операции сложения, умножения, деления. Остальные операции (например, такие, как возведение в степень, извлечение квадратного корня) реализуются программным способом.

Выполнение операций с числами, представленными с фиксированной точкой, рассмотрено в рамках материала по выполнению операций с алгебраическими числами (п. 1.3).

Выполнение длинных операций, таких, как умножение и деление, реализуется в два этапа:

– на первом этапе формируется знак искомого результата,

1. на втором этапе, используя абсолютные значения операндов, ищем результата (произведение или частное), которому затем присваивается предварительно определенный знак.

Операнды, как правило, представлены в прямом коде, и знак результата, не зависимо от того, частное это или произведение, ищется за счет сложения по модулю 2 знаковых разрядов операндов. В результате этого знак результата положителен, если операнды имеют одинаковые знаки, или отрицательный, если операнды имеют разные знаки.

Выполнение второго этапа, т. е. умножение положительных чисел достаточно подробно изложено в п. 1.2.3.

**1.5.3. Представление чисел с плавающей точкой**

При представления числа *с плавающей точкой* число в общем случае представляет собой смешанную дробь и имеет формат, приведенный на рис. 1.2.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1р | 2р | 3р | ... | *K*р | «.» | (*k*+1)р | (*k*+2)р | ... | (*n*−1)р | *n*р |

Рис. 1.2. Формат представления числа с плавающей точкой

Местоположение точки в записи числа может быть различным, а так как сама точка в записи числа не присутствует, то для однозначного задания числа необходима не только его запись, но и информация о том, где в записи числа располагается точка, отделяющая целую и дробную части.

Поэтому в случае с плавающей точкой число *Х* представляется в виде двух частей:

*мантисса* (*х*м), отображающая запись числа, представляется в виде правильной дроби с форматом фиксированной точкой;

*порядок* (*х*п), отображающий местоположение в этой записи точки, представляется в виде целого числа с форматом фиксированной точки.

Количественная оценка числа *Х* определяется как

*Х* = *qx*п ⋅ *х*м ,

где *q* – основание системы счисления.

Для двоичной системы счисления имеет место

*Х* = *2x*п ⋅ *х*м.

При *s*-разрядном представлении модуля записи мантиссы и *k*-разрядном представлении модуля записи порядка форма с плавающей точкой обеспечивает диапазон изменения абсолютного значения числа *А*, для которого выполняется неравенство:

*х*пmax ⋅ *х*мmax = 2*p* ⋅ (1 – 2-*s*) ≥ *Х*≥ 0 ,

где *p* = 2*k* − 1.

В ЭВМ числа с плавающей точкой представляются в так называемой нормализованной форме, при которой в прямом коде мантисса нормализованного числа в старшем разряде модуля имеет ненулевое значение, а для двоичной системы счисления – нормализованная мантисса должна иметь в старшем разряде модуля прямого кода значение 1, т.е. для двоичной системы мантисса должна удовлетворять неравенству:

1 > *x*м ≥ 0,5.

Для плавающей точки максимальные значения абсолютной и относительной ошибок определяются следующим образом.

Максимальная абсолютная погрешность представления чисел:

max = 2-(*s*+1) ⋅ 2*p*;

Максимальная относительная погрешность:

max =  max / *А*min = 2-(*s*+1) ⋅ 2*p* / (*х*м min ⋅ 2*p*) = 2-(s+1) ⋅ 2*p* / ( 2-1 ⋅ 2*p*) =

= 2-(*s*+1) / (2-1) = 2-*s*.

Отсюда видно, что относительная ошибка при представлении чисел в форме с плавающей точкой существенно меньше, чем в случае с фиксированной точкой. Это, а также больший диапазон изменения представляемых чисел, является основным преимуществом представления чисел с плавающей точкой.

**1.5.4. Арифметика с плавающей точкой**

Операция сложения.

Операция сложения чисел предполагает наличие одинаковых масштабов складываемых величин. Для случая представления чисел с плавающей точкой это предполагает наличие одинаковых порядков у операндов, подлежащих суммированию. Поэтому при выполнении операции сложения чисел с плавающей точкой в общем случае должно быть реализовано три этапа:

1. выравнивание порядков;
2. сложение мантисс операндов, имеющих одинаковые порядки;
3. определение нарушения нормализации и при необходимости ее устранение.

*Пример*.

Найти разность *С*1 чисел *А* и *В*, представленных с плавающей точкой, если *А* и *В* представлены в виде порядков, соответственно [*a*п]пк и [*b*п]пк и мантисс, соответственно [*а*м]пк и [*b*м]пк, где [*а*п]пк =

= 1.001, [*а*м]пк = 1.11001, [*b*п]пк = 0.001, [*b* м]пк = 0.11100.

При выполнении операций использовать дополнительный модифицированный код.

*Решение.*

Начнем с выравнивания порядков.

Для этого из порядка первого числа вычитается порядок второго числа:

1.111 – [*а*п]дк

+ 1.111 – [-*b*п]дк

1.110 – разность порядков в дополнительном коде,

1.010 – разность порядков в прямом коде.

Так как знак разности порядков отрицательный, то в качестве общего порядка, а следовательно, и предварительного значения порядка искомого результата *С*1п`, берется порядок второго числа (*b*п). Для того чтобы взять в качестве порядка первого числа порядок второго числа, т. е. увеличить его порядок на 2, необходимо мантиссу этого меньшего числа умножить на 2−2, т. е. выполнить ее арифметический сдвиг на два разряда вправо.

Таким образом, будем иметь после выравнивания следующую форму представления операндов:

[*а*м`]пк  = 1.00110,

[*b*м`]пк = 0.11100.

После выравнивания порядков можно определить предварительное значение мантиссы *С*1` как

*С*1` = [*a*м`]пк − [*b*м`]пк.

11.11010 – [*а*м`]мдк

+ 11.00100 – [-*b*м`]мдк

~~1~~10.11110 – [*С*1`]мдк

10.00010 – [*С*1`]мпк.

Из записи [*С*1`]дк, полученной после вычитания мантисс операндов с выравненными порядками, видно, что нормализация представления результата нарушена. Поэтому для данного примера необходимо выполнить этап устранения нарушения нормализации.

В данном случае нарушение нормализации слева от точки, так как получено [*С*1`]пк с ненулевой целой частью (неодинаковые разряды в поле знака использованного модифицированного дополнительного кода). Для того чтобы привести полученную предварительную мантиссу к нормализованной форме, достаточно ее разделить на 2, то есть выполнить ее арифметический сдвиг вправо. В результате будем иметь окончательное значение мантиссы:

*С*1 = *С*1` ⋅ 10-1 = 10.00010 ⋅ 10-1 = 11.10001.

Деление мантиссы *С*1` на 2 сопровождается изменением ранее найденного предварительного значения порядка результата *С*1п` на +1.

00.001 – [*с*п`]мпк

+ 00.001 – +1

00.010 – [*с*1п]мпк.

После устранения нарушения нормализации и округления окончательный результат будет иметь вид

*С*1 → {[*c*1п]пк = 0.010, [*c*1м]пк =1.1001}.

Операция умножения.

С точки зрения представления чисел с плавающей точкой поиск произведения *С*2 =*А* ⋅ *В* сводится к поиску *С*2п и *С*2м, соответственно порядку и мантиссе произведения на основании порядка *а*п и мантиссы *а*м множимого и порядка *в*п и мантиссы *в*м множителя. Учитывая общую запись чисел с плавающей точкой, произведение двух операндов представляется в виде

*С*2 = *А* ⋅ *В* = 2*а*п ⋅ *а*м ⋅ 2*в*п ⋅ *в*м = 2*а* п+*в*п ⋅ *а*м ⋅ *в*м = 2*с*2п ⋅ *с*2м.

Отсюда вытекает, что порядок произведения определяется как сумма порядков сомножителей, а мантисса произведения – как произведение мантисс сомножителей. Однако, учитывая возможность нарушения нормализации при умножении мантисс, в результате указанных действий будет найдено предварительное значения порядка и мантиссы искомого произведения, и окончательное значение произведения будет найдено только после устранения нарушения нормализации.

Таким образом, имеем:

*С*2п` = *а* п + *b*п;

*С*2м`= *а* м  ⋅ *b*м.

Отсюда последовательность действий, обеспечивающих получение произведение двух чисел, заключается в следующем:

– определяется знак произведения как сумма по модулю два знаковых разрядов мантисс сомножителей;

– определяется предварительное значение порядка произведения посредством суммирования порядков сомножителей;

– определяется предварительное значение мантиссы произведения как произведения мантисс операндов;

– устраняется нарушение нормализации мантиссы произведения (если нарушение имеет место) соответствующей корректировкой предварительного значения порядка и мантиссы искомого произведения.

При формировании мантиссы произведения нормализованных чисел с плавающей точкой возможен только один вид нарушения нормализации – нарушение нормализации справа от точки с появлением нуля только в старшем разряде мантиссы.

*Пример*.

Найти произведение *С*2 чисел *А* и *В*, представленных с плавающей точкой, если *А* и *В* представлены в виде порядков, соответственно [*а*п]пк и [*в*п]пк и мантисс, соответственно [*а*м]пк и [*b*м]пк,

где [*а*п]пк = 1.010, [*а*м]пк = 1.1010, [*b*п]пк = 0.001, [*b*м]пк = 0.1001.

При выполнении операций использовать *обратный код*. При умножении мантисс использовать метод умножения, начиная *с младшего разряда* множителя со сдвигом промежуточного результата.

*Решение*.

Знак искомого произведения, представляемого знаком его мантиссы, отрицательный, так как знаки мантисс сомножителей неодинаковые.

Предварительное значение порядка произведения определяется следующим образом:

*С*2п` = *а*п + *b*п:

11.101 – [*а*п]мок

+ 00.001 – [*b*п]мок

11.110 – [*С2*п`]мок

11.001 – [*С*2п`]мпк, т. е. [*С*2п`]пк = 1.001.

Абсолютное значение предварительного значения мантиссы произведения определяется следующим образом:

*С*2м`:

0.1010 – *а*м

× 0.1001 – *b*м

.0000 – начальное значение промежуточного произведения

+ 1010 – первый младший разряд множителя равен единице

1010 – промежуточное произведение с учетом первого разряда

01010 – сдвинутое промежуточное произведение

001010 – второй разряд множителя равен нулю, поэтому выполняется только сдвиг

0001010 – третий разряд равен нулю, поэтому выполняется только сдвиг

+ 1010 – четвертый разряд равен единице

1011010 – промежуточное произведение с учетом старшего разряда

01011 010 – сдвинутое промежуточное произведение.

Таким образом,

[*С*м`]пк = 0.01011010.

С учетом округления имеем

[*С* м`]пк = 0.01011.

Мантисса произведения ненормализованная, поэтому необходимо сдвинуть мантиссу влево на один разряд, а предварительное значение порядка произведения уменьшить на единицу. После нормализации и округления с учетом ранее полученного знака окончательные значения мантиссы и порядка произведения будут следующими:

[*С*м]пк = 1.1011.

[*С*п]пк = 1.010.

Операция деления.

С точки зрения формирования частного представления чисел с плавающей точкой поиск частного *С*3 =*А* / *В* сводится к поиску *С*3п и *С*3м, соответственно порядку и мантиссы частного на основании порядка *а*п и мантиссы *а*м делимого и порядка *b*п и мантиссы *b*м делителя. Учитывая общую запись чисел с плавающей точкой, произведение двух операндов представляется в виде

*С*3 = *А* / *В* = 2*aп* ⋅ *а*м / (2*b*п ⋅ *b*м) = 2*а*п - *b*п ⋅ (*а*м / *b*м) = 2*с3*п ⋅ с3м.

Отсюда следует, что порядок частного определяется как разность порядка делимого и делителя, а мантисса – как частное от деления мантиссы делимого на мантиссу делителя. Однако, учитывая то, что при делении мантисс может произойти нарушение нормализации, в результате указанных действий будет найдено предварительное значения порядка и мантиссы искомого частного. Окончательные значения порядка и мантиссы частного будут определены после устранения нарушения нормализации в предварительном результате.

При формировании мантиссы частного нормализованных чисел с плавающей точкой возможен только один вид нарушения нормализации – нарушение нормализации слева от точки.

*Пример*.

Найти частное *С*3 от деления чисел *А* на *В*, представленных с плавающей точкой, если *А* и *В* представлены в виде порядков, соответственно

[*а*п]пк и [*b*п]пк и мантисс, соответственно [*а*м]пк и [*b*м]пк,

где [*а*п]пк = 1.010, [*а* м]пк = 1.1010, [*b*п]пк = 0.001, [*b*м]пк= 0.1001.

При выполнении операций использовать обратный код. При делении мантисс использовать метод деления без восстановления остатка. При вычитании порядков и формировании мантиссы частного использовать модифицированный обратный код.

*Решение*.

Знак искомого частного, представляемого знаком его мантиссы, отрицательный, так как знаки мантисс сомножителей не одинаковые.

Предварительное значение порядка [*С*3п`]ок частного определяется следующим образом:

*С*3п` = *а*п - *b*п:

11.101 – [*а*п]мок

+ 11.110 – [*b*п]мок

111.011

+ 1

11.011 – [*С*3п`]мок, т. е. [*С*3п`]пк = 1.011.

Абсолютное значение предварительного значения мантиссы частного ищется за счет выполнения шести тактов деления следующим образом:

00.1010 – [|*а*м]ок ,

+11.0110 – [-|*b*м]ок ,

100.0000

+1 – учет переноса (переполнения знакового поля) при сложении в обратном коде,

00.0001 – положительный остаток первого такта,

00.0010 – сдвинутый остаток,

+11.0110 – [-|*b*м]мок,

11.1000 – отрицательный остаток второго такта,

11.0001 – остаток после арифметического сдвига влево,

+ 00.1001 – [*b*м]мок,

11.1010 – отрицательный остаток третьего такта,

11.0101 – остаток после арифметического сдвига влево,

+ 00.1001 – [*b*м]мок,

11.1110 – отрицательный остаток четвертого такта,

11.1101 – остаток после арифметического сдвига влево,

+ 00.1001 – [*b*м]мок,

100.0110

+ 1

00.0111 – положительный остаток пятого такта

00.1110 – остаток после арифметического сдвига влево,

+11.0110 – [-*b*м]мок,

100.0100

+ 1

00.0101 – положительный остаток шестого такта,

00.1010 – остаток после арифметического сдвига влево.

Таким образом, учитывая знаки остатков, полученных на шести тактах, абсолютное предварительное значение мантиссы искомого частного равно:

[*С*3м`]пк = 1.00011,

с учетом округления:

[*С*3м`]пк = 1.0010.

Мантисса частного не нормализованная (нарушение нормализации слева от точки), поэтому необходимо сдвинуть мантиссу вправо на один разряд, а предварительное значение порядка частного увеличить на единицу. После нормализации окончательные значения мантиссы и порядка частного равны:

[*С*3м]пк = 0.1001,

[*С*3п]пк = 0.000.

* 1. **Задания**

Задание 1.

Вычислить в двоичном дополнительном коде

*C*1 = *A* + *B*, *C*2 = *A - B*, *C*3 = -*A* + *B*, *C*4 = -*A* - *B*,

где *A* = *XX*, *B* = *YY*. *XX* и *YY* – число и месяц рождения студента.

Результат представить в прямом коде.

Задание 2.

Вычислить в двоичном обратном коде

*C*1 = *A* + *B*, *C*2 = *A* - *B*, *C*3 = - *A* + B, *C*4 = - *A* - *B*,

где *C*1*, C*2*, C*3*,* C4 имеют тот же смысл, что и в задании1.

Задание 3.

Найти сумму *C* чисел *A* и *B* в формате с плавающей точкой, если *A* и *B* представлены в виде порядков, соответственно [*a*п]пк и [*b*п]пк и мантисс, соответственно [*a*м]пк и [*b*м]пк, где [*a*п]пк = *XX*2, [*b*п]пк = *YY*2, [*a*м]пк = *ZZ*2, [*b*м]пк = -*KK*2; *XX*2 – двоичный код номера зачетной книжки студента, *YY*2, *ZZ*2, *KK*2 – дата рождения (число, месяц, две последние цифры года) в двоичном коде.

При выполнении операции использовать *дополнительный код*.

Задание 4.

Найти произведение *C* чисел *A* и *B* в формате с плавающей точкой, если *A* и *B* представлены в виде порядков, соответственно [*a*п]пк и [*b*п]пк и мантисс, соответственно [*a*м]пк и [*b*м]пк, где [*a*п]пк = *XX*2, [*b*п]пк = -*YY*2, [*a*м]пк = *ZZ*2, [*b*м]пк = *KK*2. Здесь *XX*2, *YY*2, *ZZ*2, *KK*2 имеют тот же смысл, что и в задании 3.

При выполнении операции использовать *обратный код*. При умножении мантисс использовать метод умножения, начиная со *старшего разряда множителя* со сдвигом *промежуточного результата.*

**2. ЛОГИЧЕСКИЕ ОСНОВЫ ЦВМ**

* 1. **Основные понятия алгебры логики**

Алгебра логики используется при анализе и синтезе схем ЭВМ по двум причинам. Во*-*первых, это объясняется соответствием представления переменных и функций алгебры логики. Во*-*вторых, двоичным представлением информации и характером работы отдельных компонентов вычислительной техники. Эти компоненты могут пропускать или не пропускать ток, иметь на выходе высокий или низкий уровень сигнала (напряжения или тока).

Приведем основные понятия алгебры логики.

*Логическая переменная*–это такая переменная, которая может принимать одно из двух значений: истинно или ложно (да или нет, единица или ноль).

*Логическая константа –* это такая постоянная величина, значением которой может быть истинно или ложно (да или нет, единица или ноль).

*Логическая функция –* это такая функция, которая может принимать одно из двух значений: истинно или ложно (да или нет, единица или ноль) в зависимости от текущих значений ее аргументов, в качестве которых используются логические переменные.

Логическая функция может быть одного (*n* = 1) или нескольких (*n* > 2) аргументов. Значение логической функции определяется комбинацией конкретных значений переменных, от которых она зависит. Комбинация конкретных значений переменных (аргументов функции) называется набором. Количество различных наборов *N* для «*n*» переменных вычисляется по формуле *N* = 2*n*.

Зависимость логической функции от переменных может задаваться по*-*разному:

– словесным описанием;

– таблицей истинности;

– логическим выражением.

*Словесное описание* используется в случае сравнительно несложной логической функции.

*Таблица истинности* является универсальным средством задания логической функции. Она включает все наборы для заданного количества переменных, определяющих значение логической функции, с указанием значений, которые принимает функция для каждого набора. В одной таблице истинности может задаваться несколько логических функций, зависящих от одних и тех же переменных. Таблица истинности для нескольких функций *yi* трех переменных *х*1, *х*2, *х*3 может быть задана следующим образом (табл. 2.1)

Таблица 2.1

**Таблица истинности трех переменных**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | *х*1 | *х*2 | *х*3 | *y*1 | *y*2 | *y*3 | … | *yn* |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 |  | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 |  | 1 |
| 2 | 0 | 1 | 0 | 1 | 1 | 1 |  | 0 |
| 3 | 0 | 1 | 1 | 0 | 1 | 0 |  | – |
| 4 | 1 | 0 | 0 | 1 | 0 | 1 |  | 0 |
| 5 | 1 | 0 | 1 | 0 | 0 | 0 |  | 1 |
| 6 | 1 | 1 | 0 | 0 | 0 | 1 |  | – |
| 7 | 1 | 1 | 1 | 1 | 1 | 0 |  | 1 |

В приведенной таблице истинности во второй, третьей и четвертой колонках, помеченных соответственно *х*1, *х*2, *х*3 , приведены все возможные наборы этих переменных. В следующих колонках приводятся значения функций *y*1, *y*2, *yn* для каждого набора.

Логическая функция называется «*полностью определенной*», если для нее заданы значения по всем возможным наборам. Функция называется «*частично определенной*», если для некоторых наборов значения функции не заданы. В приведенной таблице истинности функции *y*1, *y*2, *y*3 являются полностью определенными, а функция *yn* – частично определенная (знак «–» означает неопределенность значения функции).

Максимальное количество полностью определенных функций от «*n*» переменных определяется как *M* = (22)*n* .

*Логическим выражением* называется комбинация логических переменных и констант, связанных элементарными базовыми логическими функциями (или логическими операциями), которые могут разделяться скобками.

Например, логическую функцию *у*1, определенную в вышеприведенной таблице истинности, можно представить в виде логического выражения



где «+», «·», а также верхнее надчеркивание – знаки базовых логических функций.

Набор элементарных логических операций, с помощью которых можно задать любую, сколь угодно сложную логическую функцию, называется *функционально полной системой логических функций*. Иногда такую систему называют *базисом*.

В качестве элементарных логических функций функционально полных систем этих функций используются функции одной или двух логических переменных.

Все возможные функции одной переменой приведены в табл. 2.2.

Таблица 2.2

**Функции одной переменной**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *x* | *y*0 | *y*1 | *y*2 | *y*3 |
| 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |

Из таблицы видно, что:

*y*0 = 0 – константа; *y*1 равна значению переменной; *y*2 равна значению, обратному значению переменной « *х* »; *y*3 = 1 – константа.

С точки зрения базовых функций интерес представляет только функция *y*2, она называется *функцией отрицания*, читается как «не *х*» и обозначается как «», т. е. можно записать *y*2 = .

Все возможные функции двух переменных приведены в табл. 2.3

Таблица 2.3

**Функции двух переменных**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | *х*1 | *х*2 | *y*0 | *y*1 | *y*2 | *y*3 | *y*4 | *y*5 | *y*6 | *y*7 | *y*8 | *y*9 | *y*10 | *y*11 | *y*12 | *y*13 | *y*14 | *y*15 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 2 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 3 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Информация по функциям двух переменных приведена в табл. 2. 4.

Таблица 2.4

**Булевы выражения для функций двух переменных**

| *y*i | Название функции | Чтение функции | Запись в виде  булевого  выражения | | |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | | |
| *y*0 | Const «0» |  | 0 |  | |
| *y*1 | Конъюнкция | и *х*1, и *х*2 | *х*1 · *х*2; | *х*1*х*2; *х*1 & *х*2 | |
| *y*2 | Запрет по *х*2 | неверно, что, если *х*1, то *х*2 |  |  | |
| *y*3 | *F*(*х*1) | функция одной переменной | *х*1 |  | |
| *y*4 | Запрет по *х*1 | неверно, что, если *х*2 , то *х*1 |  |  | |
| *y*5 | *F*(*х*2) | функция одной переменной | *x*2 |  | |
| *y*6 | Неравнозначности | *х*1 не равно *х*2 | + | |  |
| *y*7 | Дизъюнкция | или *х*1, или *х*2 | *х*1 + *х*2 |  | |
| *y*8 | Пирса | ни *х*1, ни *х*2 |  |  | |
| *y*9 | Равнозначности | *х*1 равно *х*2 | + | |  |
| *y*10 | *F*(*х*2) | Не x2 |  |  | |
| *y*11 | Импликация | если *х*2 , то *х*1 |  |  | |
| *y*12 | *F*(*x*1) | Не х1 |  |  | |
| *y*13 | Импликация | если *х*1, то *х*2 |  |  | |
| *y*14 | Шеффера | неверно, что и *х*1, и *х*2 |  |  | |
| *y*15 | Const ( = 1) |  | 1 |  | |

Наиболее распространенной в алгебре логики является функционально полная система логических функций, которая в качестве базовых логических функций использует функцию одной переменной «НЕ» (функция отрицания) и две функции двух переменных – «И» (конъюнкция или логическое умножение) и «ИЛИ» (дизъюнкция или логическое сложение). Эта система получила название *система* *булевых* *функций*, или *булевый базис*. В алгебре логики имеется целый раздел *Алгебра Буля*, посвященный этому базису.

В вышеприведенной таблице, описывающей функции от двух переменных, в последней колонке приведены варианты записи этих функций в булевом базисе. Среди перечисленных формул наиболее известными являются так называемые «Функция Пирса» (другие названия – ИЛИ-НЕ, «Отрицание дизъюнкции») и «Функция Шеффера» (по-другому – И-НЕ, «Отрицание конъюнкции»). Они также представляют собой функционально-полные системы логических функций и могут записываться в следующем виде (в этом случае их принято называть «Стрелка Пирса» и «Штрих Шеффера» соответственно):

.

Справедливость утверждения о функциональной полноте этих функций будет показана в из следующих подразделов.

**2.2. Основные понятия булевой алгебры**

В алгебре Буля логические выражения включают логические операции И, ИЛИ, НЕ, которые могут быть использованы в самых различных сочетаниях. При оценке значения такого выражения необходимо решить его для конкретного набора переменных. В алгебре Буля применяется следующая приоритетность выполнения операций: сначала рассчитываются значения имеющих место отрицаний и скобок, затем выполняется операция И (логическое умножение); самый низший приоритет имеет операция ИЛИ (логическая сумма).

При работе с булевыми логическим выражениями используются следующие законы, правила и операции.

*Переместительный* (коммутативный) *закон*. Закон справедлив как для конъюнкции, так и для дизъюнкции.

*х*1 + *х*2 + *х*3 + *х*4 = *х*4 + *х*3 + *х*2 + *х*1 – от перемены мест логических слагаемых сумма не меняется.

*х*1*х*2*х*3*х*4 = *х*4*х*3*х*2*х*1 – от перемены мест логических сомножителей их произведение не меняется.

Этот закон справедлив для любого количества логических операндов.

*Сочетательный* (ассоциативный) *закон* справедлив как для конъюнкции, так и для дизъюнкции.

*х*1 + *х*2 + *х*3 + *х*4 = (*х*2 + *х*3) +*х*1 + *х*4 = (*х*1 + *х*4 ) + (*х*2 + *х*3) – при логическом сложении отдельные слагаемые можно заменить их суммой.

*х*1 *х*2 *х*3 *х*4 = (*х*2 *х*3)*х*1*х*4 = (*х*1 *х*4) (*х*2 *х*3) – при логическом умножении отдельные логические сомножители можно заменить их произведением.

*Распределительный* (дистрибутивный) *закон.*

(*х*1 + *х*2) *х*3 = *х*1*х*3 + *х*2*х*3;

(*х*1 + *х*2) (*х*1 + *х*3) = *х*1 + *х*2 *х*3.

*Правило де Моргана.*

* –* отрицание суммы равно произведению отрицаний;

** – отрицание произведения равно сумме отрицаний.

*Операция склеивания.*

 *–* операция *склеивания* для конъюнкций, где *А* *–* переменная или любое логическое выражение.

 *–* операция склеивания для дизъюнкций.

Если в качестве *А* используется *простая конъюнкция*, т. е. конъюнкция, представляющая собой логическое произведение переменных и их отрицаний, то имеет место



Как видно, в результирующем выражении количество переменных на единицу меньше, чем в склеенных конъюнкциях. Количество переменных в простой конъюнкции называется *рангом конъюнкции*, т.е. операция склеивания, примененная к простым конъюнкциям, дает результат с рангом, на единицу меньшим ранга исходных конъюнкций.

*Операции с отрицаниями.*

 *–* двойное отрицание равносильно отсутствию отрицания;

**;

*.*

*Операции с константами.*

*х*1 + 1 = 1, *х*1 + 0 = *х*1,

*х*1 ⋅1 = *х*1, *х*1 ⋅ 0 = 0.

*Операции с одинаковыми операндами.*

*х*1 + *х*1 + *х*1 + *х*1 + ... + *х*1 = *х*1;

*х*1 *х*1 *х*1 .... *х*1 = *х*1 при любом числе повторений.

Законы, правила и операции алгебры *Буля* могут быть доказаны путем логического рассуждения, однако такое доказательство применимо только для простейших случаев. Доказать справедливость того или иного правила можно, если с помощью различных преобразований привести правую часть правила к выражению в левой части (или наоборот). Универсальным приемом доказательства является использование таблицы истинности. Это основано на том утверждении, что два выражения (правая и левая часть правила или закона) *эквивалентны*, если они принимают одинаковые значения на всех наборах логических переменных.

Например, правило двойного отрицания, которое справедливо не только относительно одной переменной, но и любого логического выражения, можно доказать следующим рассуждением: если неверно утверждение, что выражение ложно, то очевидно утверждение, что это выражение истинно.

Доказать справедливость распределительного закона в интерпретации выражением (*х*1 + *х*2)(*х*1 + *х*3) = *х*1 + *х*2*х*3 можно за счет приведения левой части к выражению правой части, раскрыв скобки:

(*х*1 + *х*2)(*х*1 + *х*3) = *х*1*х*2 + *х*1*х*1+*х*1*х*3 + *х*2*х*3 = *х*1*х*2 + *х*1 + *х*1*х*3 +   
+ *х*2*х*3 = *х*1(*х*2 + 1+ *х*3) + *х*2*х*3.

Помня, что логическая сумма с одним слагаемым, равным константе «1», равна «1», можно записать *х*1 + *х*2*х*3.

Используем таблицу истинности для доказательства правила де Моргана в варианте  – отрицание суммы равно произведению отрицаний.

Составим таблицу истинности для правой и левой частей и составляющих их функций (табл. 2.5).

Таблица 2.5

**Таблица истинности для правила де Моргана**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *х*1 | *х*2 | *х*1 + *х*2 |  |  |  |  |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |

Из таблицы истинности видно, что правая и левая части доказываемого правила принимают одинаковые значения на всех наборах, следовательно они эквивалентны.

*Функционально полной системой булевых функций* (ФПСБФ) называется совокупность таких булевых функций (*f*1, *f*2, ..., *fk*), посредством которых можно записать произвольную булеву функцию *f*. Как уже было сказано, ФПСБФ являются «Стрелка Пирса» и «Штрих Шеффера».

**2.3. Записи функций алгебры логики (ФАЛ) в различных формах, их взаимосвязь**

Одну и ту же логическую функцию можно представить различными логическими выражениями. Среди множества выражений, которыми представляется логическая функция, особое место занимают две канонические формы: *совершенная конъюнктивная нормальная форма* (СКНФ) и *совершенная дизъюнктивная нормальная форма* (СДНФ).

*Совершенная дизъюнктивная нормальная форма* представляет собой дизъюнкцию простых конъюнкций, где под термином простая *конъюнкция* имеется в виду конъюнкция переменных или их отрицаний. В СДНФ простые конъюнкции содержат все переменные в своей прямой или инверсной форме и отражают собой наборы, на которых представляемая функция имеет единичное значение. Такие конъюнкции называются *конституентами* *единицы* рассматриваемой функции. Поэтому СДНФ представляет собой дизъюнкцию (логическую сумму), слагаемыми которой являются конституенты единицы. Общая запись СДНФ функции *y* имеет вид

*y* = ∨ *х*1δ1 *х*2δ2 *х*3δ3... *х*(*n–*1)δ(*n–*1) *хn*δ*n*,

**

СДНФ легко сформировать на основе таблицы истинности. иде таблицы истинностиНапример, если функции задаются в в (табл. 2.6), то СДНФ для них будет иметь следующий вид.

*y*1*=;*

*y*2*=;*

*y*3*=*.

*Совершенная конъюнктивная нормальная форма* – это конъюнкция простых дизъюнкций, где под термином простая дизъюнкция

Таблица 2.6

**Таблица истинности СДНФ**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | *х*1 | *х*2 | *х*3 | *y*1 | *y*2 | *y*3 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 0 | 0 | 0 |
| 6 | 1 | 1 | 0 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 | 0 | 1 |

имеется в виду дизъюнкция переменных или их отрицаний. В СКНФ простые дизъюнкции содержат все переменные в своей прямой или инверсной форме и представляют собой отрицание конституент нуля. Общая запись СКНФ функции *y* имеет вид

*y* = ∧ (*х*1δ1 + *х*2δ2 +*х*3δ3 + ... + *х*(*n–*1)δ(*n–*1) *+ хn*δ*n*),

**

СКНФ легко сформировать на основе таблицы истинности. Например, для функций из предыдущей таблицы (табл. 2.6) имеем:







СКНФ строится на основе конституент нуля. *Конституента нуля* представляет набор логических переменных, на котором логическая функция принимает значение «0».

Каждая скобка в приведенных выражениях представляет собой отрицание конституенты нуля соответствующей функции, а запись функции в виде конъюнкции таких скобок представляет собой условие, при котором отсутствуют все конституенты нуля определяемой функции, при выполнении которого функция имеет единичное значение.

Например, для *y*1 выражение первой скобки представляет собой отрицание набора значений переменных второй строки, на котором функция *y*1 имеет нулевое значение. Выражение второй скобки представляет собой отрицание набора значений переменных четвертой строки, на которой функция *y*1 также имеет нулевое значение, выражение третьей скобки – отрицание набора значений переменных пятой строки, на котором функция *y*1 имеет нулевое значение.

Из вышеизложенного видим, что любую функцию можно представить или в СДНФ, или в СКНФ, а т. к. эти формы представлены в базисе Буля, то отсюда следует, что этот базис (базис И, ИЛИ, НЕ) является *функционально полным*.

Если функция задана в СДНФ и требуется найти ее СКНФ, то такой переход можно выполнить, составив по заданной СДНФ таблицу истинности для этой функции. На основе полученной таблицы составляется СКНФ заданной функции.

Однако в некоторых случаях может оказаться более удобным подход, который поясняется следующим примером.

*Пример*.

По заданной СДНФ функции:

****

найти запись этой функции в СКНФ.

*Решение.*

Запишем логическое выражение отрицания заданной функции, т. е. найдем логическое условие, при котором эта функция имеет нулевое значение. В качестве такого выражения можно взять дизъюнкцию конъюнкций, где каждая конъюнкция представляет собой конституенту нуля заданной функции. Очевидно, что конституенты нуля – это те наборы, которые не являются наборами, соответствующими конституентам единицы, использованным в СДНФ. Таким образом, можно записать

****

Эту запись можно интерпретировать как словесное описание функции: функция *у* равна нулю, если имеет место хотя бы одна из конституент нуля.

В этой записи представлена дизъюнкция тех наборов, которые не использовались в записи функции *y*3. Возьмем отрицание правой и левой частей полученного уравнения и применим к правой части правило де Моргана:

.

Применим *правило де Моргана* к отрицаниям конъюнкций, полученным в правой части:

.

Полученная запись для *у* является искомой СКНФ.

**2.4. Минимизация функций алгебры логики**

Учитывая то, что одну и ту же логическую функцию можно представить различными выражениями, перед реализацией функции в виде логической схемой весьма важным является выбор из всех возможных выражений, соответствующих данной функции, самого простого. Решить эту проблему можно за счет использования процедуры минимизации логического выражения.

***Методы минимизации:***

1. методом Квайна;
2. с использованием диаграмм Вейча (или карт Карно);
3. не полностью определенных (частично определенных) функций;
4. конъюнктивных нормальных форм;
5. методом кубического задания функций алгебры логики;
6. методом Квайна*-*Мак*-*Класски;
7. с использованием алгоритма извлечения (Рота);
8. методом преобразования логических выражений.

Далее мы рассмотрим методы, употребляемые наиболее часто.

**2.4.1. Минимизация методом Квайна**

В качестве исходной формы представления логического выражения используется СДНФ. Если подлежащее минимизации выражение имеет другую форму, то приведение к СДНФ осуществляется за счет открытия скобок, избавления от отрицаний логических выражений, более сложных чем отрицание переменной (используется правило де Моргана).

Метод Квайна выполняется в два этапа.

*Первый этап* имеет своей целью получение тупиковой формы, представляющей собой дизъюнкцию, в качестве слагаемых которой используются конъюнкции (каждая из них не склеивается ни с одной другой конъюнкцией, входящей в это выражение). Такие конъюнкции называются *простыми импликантами*.

Данный этап выполняется за счет реализации отдельных шагов. На каждом шаге на основании выражения, полученного на предыдущем шаге, выполняются все возможные операции склеивания для пар имеющихся конъюнкций. Каждый шаг понижает ранг исходных конъюнкций на единицу. Шаги повторяются до получения тупиковой формы.

*Второй этап* имеет своей целью устранение из тупиковой формы всех избыточных простых импликант, что дает в результате минимальное логическое выражение.

*Пример.*

Найти методом Квайна минимальное выражение для функции *y*:

**

*Решение*

1*-*й этап:

**

Над конъюнкциями проставлены их номера; в скобках под каждой конъюнкцией (*i–j*) указывают, что данная конъюнкция является результатом склеивания *i-*й и *j-*й конъюнкций исходного выражения.

К результатам склеивания логически добавлен ни с чем не склеенный пятый член исходного выражения; несколько одинаковых конъюнкций представляются одной конъюнкцией.

Последнее выражение получено из предыдущего посредством удаления повторяющихся членов.

2*-*й этап:

На основании исходного выражения и полученной тупиковой формы составляется и заполняется импликантная таблица (табл. 2.7).

Колонки приведенной таблицы помечены конституентами единицы, имеющимися в исходном логическом выражении.

Строки таблицы помечены простыми импликантами полученной тупиковой формы.

Звездочками в каждой строке отмечены те конституенты единицы, которые покрываются соответствующей простой импликантой (практически отмечаются те конституенты единицы, которые включают простую импликанту как свою составную часть).

Анализируя покрытия простыми импликантами конституент единицы заданной функции, составляем ее минимальное выражение:

.

Минимальное выражение *y*min формируется за счет последовательного включения простых импликант. При этом используется следующая приоритетность включения импликант в формируемое минимальное выражение:

Таблица 2.7

**Импликантная таблица**

|  | \_ \_  *х*1*х*2*х*3*х*4 | \_ \_  *х*1*х*2*х*3*х*4 | \_ \_  *х*1*х*2*х*3*х*4 | \_ \_  *х*1*х*2*х*3*х*4 | \_ \_ \_  *х*1*х*2*х*3*х*4 | \_  *х*1*х*2*х*3*х*4 | \_ ­\_  *х*1*х*2*х*3*х*4 | \_  *х*1*х*2*х*3*х*4 | \_  *х*1*х*2*х*3*х*4 | \_ \_ \_  *х*1*х*2*х*3*х*4 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| *\_*  *х*2*х*3 | \* |  |  |  |  |  | \* |  | \* | \* |
| \_  *х*2*х*3 |  | \* | \* |  | \* | \* |  |  |  |  |
| \_  *х*2*х*4 |  | \* |  | \* | \* |  |  | \* |  |  |
| \_  *х*3*х*4 |  |  |  | \* |  |  | \* | \* |  | \* |
| \_ \_  *х*1*х*2*х*3 |  |  | \* |  | \* |  |  |  |  |  |

* простая импликанта является единственной, покрывающей одну из колонок;
* если импликант вышеуказанного типа нет, то выбирается импликанта, покрывающая большее количество еще не покрытых колонок.

Последовательность включения простых импликант в приведенное минимальное выражение:

 – единственная импликанта, покрывающая колонку 1, при этом из дальнейшего рассмотрения исключаются все колонки, покрываемые этой импликантой, т. е. колонки 1, 7, 9, 10;

 – покрывает максимальное число колонок, оставшихся для рассмотрения (колонки 2, 3, 5, 6). Эти колонки из дальнейшего рассмотрения исключаются;

****–покрывает оставшиеся две колонки (4, 8); после исключения этих двух колонок для рассмотрения не останется ни одной колонки, не покрытой уже включенными в формируемое выражение простыми импликантами. Поэтому простые импликанты  и  найденной тупиковой формы являются избыточными и в минимальном логическом выражении для заданной функции не присутствуют.

* + 1. **Минимизация диаграммами Вейча**

Минимизация этим методом предполагает использование специальных форм – диаграмм Вейча (или карт *Карно*).

Карта Карно для «*n*» логических переменных представляет собой множество квадратов (клеток), объединенных в близкую к квадрату прямоугольную форму. Каждая такая клетка соответствует одному набору логических переменных, причем наборы двух соседних клеток должны отличаться на значение одной переменной (их наборы образуют склеивающиеся конъюнкции).

На рис. 2.1 приведены карты Карно для *n* = 1, 2, 3. На рис.2.1, *а*, 2.1,*б* показана разметка колонок и строк, а также указан для каждой составляющей клетки соответствующие ей набор. Разметка колонок (строк) указывает, какие значения данная переменная имеет в клетках, находящихся в данной колонке (строке). На рис. 2.1, *в* приведен пример компактной разметки карты, соответствующей карте на рис.2.1, *б*. Здесь помечаются колонки (строки), в которых соответствующая переменная имеет прямое значение. На рис. 2.1, *г* приведена карта Карно для *n* = 3, сформированная посредством зеркального отображения карты Карно для *n* = 2 (рис. 2.1, *в*) относительно правой границы. Этот прием универсальный; его можно использовать для построения карты для заданного «*n*» на основании имеющейся карты Карно для «*n* - 1» переменной. Клетка, отмеченная знаком «\*», соответствует набору **. Карты Карно используются для представления и минимизации логических функций.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | *x*2 | *x*1*x*2 | *x*2 | *x*2 |  |  | *x*2 |  |  |  |  |
|  |  |  | *x*1 |  |  |  |  |  |  |  | \* |  |
|  |  |  |  |  |  |  |  |  |  | |  |  |
|  |  |  |  |  |  |  |  |  | *x*3 |  |  |  |

*а б в г*

Рис. 2.1. Карта Карно: *а* – для одной переменной, *б* – для двух переменных, *в*– сокращенная для 2-х переменных, *г* – для 3-хпеременных

Записываемая функция должна быть представлена в СДНФ. Запись функции в карту осуществляется за счет установки «1» в клетки карты, соответствующие конституентам единиц записываемой функции.

Для выполнения минимизации представленной в карте Карно функции необходимо выполнить два этапа:

* охватить множество клеток карты Карно контурами;
* записать минимальное выражение для заданной функции в виде дизъюнкции конъюнкций, где каждая конъюнкция соответствует одному из введенных на карте контуров.

Охват клеток карты контурами выполняется с соблюдением следующих правил:

* контур должен иметь прямоугольную форму;
* в контур может входить количество клеток, равное целой степени числа «2»;
* в контур могут входить клетки, являющиеся логическими соседями;
* в контур необходимо включить максимальное количество клеток с учетом вышеприведенных требований;
* контурами необходимо охватить все клетки с единичными значениями;
* контуров должно быть минимальное количество;
* количество клеток в контуре должно быть равно 2ΔR, где ΔR –разность ранга (дельта ранга) конституент единицы заданной функции и ранга конъюнкции, соответствующей контуру.

*Логическими соседями* являются такие две клетки, наборы которых отличаются только одной переменной – в одном эта переменная должна иметь прямое, в другом – обратное значение.

Для того чтобы быть логическими соседями, клеткам достаточно быть геометрическими соседями. Считая, что карта является пространственным объектом и заворачивается по горизонтали и вертикали, сливаясь своими крайними горизонтальными и крайними вертикальными границами, можно считать, что соответствующие крайние горизонтальные и вертикальные клетки являются геометрическими соседями. Логическими соседями могут быть клетки, которые не являются геометрическими соседями. К числу таких клеток относятся клетки, которые по горизонтали или вертикали симметричны относительно линий зеркального отображения, использованных при переходе от «*n*» к «*n*+1» переменным.

Запись минимального выражения по заданной функции имеет вид дизъюнкции простых конъюнкций, соответствующих контурам на карте, и формируется следующим образом:

* конъюнкция, соответствующая контуру, должна включать только те переменные, которые имеют постоянное значение во всех клетках, охваченных рассматриваемым контуром,
* или по другому: в конъюнкцию, соответствующую контуру, не должны входить переменные, которые имеют разные значения для клеток, охваченных рассматриваемым контуром.

Например, если задана логическая функция «*y*» трех переменных в виде выражения



то ее запись в карту Карно будет иметь вид, приведенный на рис.2.2.

*x*3

x2

*x*1

1

1

1

1

Рис. 2.2. Карта Карно для 3-х переменных

Для функции, заданной в карте Карно, приведенной на рис. 2.2, контуры имеют вид, приведенный на рис. 2.3.

Для примера, контур 1 представлен на рисунке в виде двух клеток: клетки, соответствующей набору *x*1*x*2*x*3, и клетки, соответствующей набору , поэтому данному контуру будет соответствовать конъюнкция*x*1 *x*2.

Минимальное логическое выражение для функции имеет вид:

*y* = *x*1 *x*2 + *x*1 *x*3 + *x*2*x*3.

1 2 3

*x*1  x1

1

1

1

1

1

*x*2

2

1

*x*3

3

Рис. 2.3. Сформированные контуры для заданной функции

Конъюнкции минимального выражения помечены внизу цифрами, соответствующими номерам контуров, которые они представляют.

На рис.2.4, a, б приведены карты Карно для четырех и пяти переменных.

На рис.2.5, c приведена карта Карно для шести переменных. В этой карте приведен пример расположения четырех геометрически соседних клеток с единичными значениями, которые нельзя объединить единым контуром (1). Эти, рядом лежащие клетки, необходимо охватить двумя контурами (2, 3). Действительно, конъюнкция для неправильного контура 1 имеет вид  и дельта ранга R при ранге шесть конституент единицы исходного выражения, составляет значение «3», отсюда количество клеток, входящих в контур, должно быть равно третьей степени двойки. Это требование не выполняется (контур охватывает только четыре клетки), следовательно контур 1 введен неправильно. В приведенной ситуации необходимо для охвата рассматриваемых клеток использовать два контура, соответственно контур 2 и контур 3.

*Пример*

Минимизировать функцию «y», заданную в карте Карно, приведенной на рис.2.6.

*Решение*

Карта Карно с введенными контурами, приведена на рисунке рис.2.7.

*x*1

*x*1

*x*1

*x*1

*x*4

*x*2

*x*2

*x*4

х2

*x*2

*х*3

*х*3

*x*3

a)

б)

Рисунок 2.4 Карты Карно для *n* переменных: a – для *n*=4, б – для *n*=5.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | 1 | 1 | 1 | 1 |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

*x*4

*x*2

*x*2

2

3

1

*x*6

*x*5

*х*1

*x*4

*х*1

*x*3

*x*3

*х*1

Рисунок 2.5 Карта Карно для 6переменных.

*х*1

*х*1

*х*1

1

1

1

1

1

1

1

1

*x*2

*x*4

*x*6

1

1

1

*x*2

1

1

1

1

1

1

*x*4

*x*2

c)

*x*3

*x3*

*x*5

Рисунок 2.6 Карта Карно для заданной функции «у».

3

3

4

*х*1

*х*1

*х*1

*x*2

**1**

**1**

**1**

**1**

*x*4

7

**1**

**1**

2

**1**

**1**

6

7

*x*6

6

5

1

**1**

**1**

**1**

1

*x*2

**1**

**1**

5

**1**

*x*2

*x*4

**1**

**1**

**1**

4

3

3

*x*3

*x*3

*x*5

Рисунок 2.7 Карта Карно для заданной функции с введенными контурами.

Минимальное выражение для «у», составленное по введенным контурам, имеет вид

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *у*= | \_ \_ \_  x1 x2 x3 x4 +  1 | x3 x4 x5 x6 +  2 | x1x2 x3x4 +  3 | \_  x1x2 x3x4 x5+  4 |
| .  + | \_ \_ \_  x1x2x3x4x5+  5 | \_  x1x2x3x5x6+  6 | x1x3 x4 x6 .  7 |  |

Под каждой конъюнкцией указан номер контура, которому она соответствует.

**2.5. Синтез логических схем по логическим выражениям**

**2.5.1. Синтез логических схем в базисе И, ИЛИ, НЕ**

Логические схемы строятся на основе логических элементов, набор которых определяется заданным логическим базисом.

Для базиса Буля в качестве логических элементов используются элементы, реализующие базовые логические функции И, ИЛИ, НЕ, которые имеют приведенные на рис. 2.4 обозначения.

При синтезе схемы по логическому выражению, составляющие логические операции представляются в виде соответствующих логических элементов, связи между которыми определяются последовательностью выполнения логических операций в заданном выражении.

определяется заданным логическим базисом.

И ИЛИ НЕ

*x*1

*y = x1  x2*

&

1

&

*x*1

*y* = *x*1

*x*1

*y* = *x*1+ *x*2

*x*2

*x*2

НЕ

*y* = *x*1

1

*x*1

Рис. 2.4. Базовые логические элементы

*Пример*

Синтезировать логическую схему в базисе И, ИЛИ, НЕ, реализующую логическое выражение

|  |  |
| --- | --- |
| *y*1= | \_\_\_\_\_\_\_\_\_\_\_\_\_\_  (*x*1*x*2 + *x*1*x*3  + *x*2 *x*3)(*x*1 + *x*2 + *x*3) *+x*1 *x*2 *x*3. |

*Решение*

Входными сигналами синтезируемой схемы являются *x*1, *x*2, *x*3, а выходным – *y*1.

Реализацию заданного выражения в виде логической схемы можно начать или с последней операции, или с первой.

Последней операцией в заданном выражении является операция логического сложения двух операндов

|  |
| --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (*x*1*x*2 + *x*1*x*3  + *x*2 *x*3)(*x*1 + *x*2 + *x*3) и *х*1 *x*2 *x*3, |

поэтому для ее реализации требуется элемент ИЛИ (1) с двумя входами, на выходе которого будет сформирован сигнал, соответствующий y1, если на его входы будут поданы эти два слагаемые (например, первое слагаемое на второй вход, а второе слагаемое на первый вход).

На первый вход выходного элемента ИЛИ подается логическое произведение *х*1 *x*2 *x*3**,** для реализации которого необходимо использовать логический элемент И с тремя входами, на которые подаются входные переменные *х*1, *x*2, *x*3. Аналогичным образом рассматривается последовательность формирования выражения

|  |
| --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (*x*1*x*2 + *x*1*x*3 + *x*2*x*3)(*x*1 + *x*2 + *x*3), |

которое соответствует сигналу, подаваемому на второй вход элемента ИЛИ (1). В результате синтезируется схема для заданного выражения, приведенная на рис. 2.5.

1

*x*1 *x*2 *x*3

&

&

&

1

&

*y*

&

&

1

Рис. 2.5. Логическая схема для заданного выражения

**2.5.2. Логические базисы И-НЕ, ИЛИ-НЕ**

Булевый базис не является единственной функционально полной системой логических функций. Среди других наибольшее распространение получили базис И-НЕ и базис ИЛИ-НЕ.

Чтобы доказать логическую полноту любого базиса, достаточно показать, что в этом базисе можно реализовать базовые функции И, ИЛИ, НЕ.

Для базиса И-НЕ в качестве базового элемента используется элемент приведенный на рисунке рис. 2.6,а.

*y* =

\_\_\_

= *x*1 *x*2

*y* =

\_\_\_\_\_

= *x*1+ *x*2

*x*1

*x*2

*x*1

*х*2

1

&

*а*  *б*

Рис. 2.6. Базовые элементы: *а –* И-НЕ; *б* – ИЛИ-НЕ

Реализация с помощью функции И-НЕ базовых функций алгебры Буля осуществляется следующим образом.

ИЛИ: ;

И: 

Функция НЕ реализуется с помощью схемы И-НЕ с одним входом.

На рис. 2.7. приведена схемная реализация функций И, ИЛИ, НЕ в базисе И-НЕ.

*y* = *x*1+*х* 2

*x*1

*x*2

*y*= *x*1

*x*1

&

&

ИЛИ И НЕ

&

&

*y* = *x*1\**х*2

*x*1

*х*2

&

&

*а б в*

*Рис. 2.7. Реализация булевых функций в базисе И-НЕ: а – функция ИЛИ; б – функция* И; *в –* функция НЕ

Реализация с помощью логической функции ИЛИ-НЕ базовых функций алгебры Буля осуществляется следующим образом.

ИЛИ: ;

И: 

Функция НЕ реализуется с помощью схемы ИЛИ-НЕ с одним входом.

На рис. 2.8. приведена схемная реализация операций И, ИЛИ, НЕ в базисе ИЛИ-НЕ

*x*1

x2

1

*y* = *x*1.*х*2

1

*x*1

*х*2

*y*=

\_

= *x1*

*x*1

1

*y* =

= *x*1+*х*2

1

1

1

И ИЛИ НЕ

*а* *б* *в* в)

Рис. 2.8. Реализация булевых функций в базисе ИЛИ-НЕ: *а –* функция И; *б –* функция ИЛИ; *в –* функция НЕ

**2.5.3.Синтез логических схем в базисах И-НЕ, ИЛИ-НЕ**

При синтезе логических схем в заданном базисе логических элементов (например, в базисах И-НЕ, или ИЛИ-НЕ) целесообразно предварительно исходное выражение привести к форме, в которой в выражении будут использованы только логические операции, соответствующие используемым логическим элементам в заданном базисе.

*Пример*.

Синтезировать логическую схему в базисе И-НЕ, соответствующую выражению

.

*Решение*

Используя правило де Моргана преобразуем исходное выражение таким образом, чтобы последней операцией было отрицание и в выражение были бы только операции И.



Полученное выражение, представленное в виде вложенных операций И-НЕ, позволяет синтезировать соответствующую логическую схему в заданном базисе, которая приведена на рис. 2.9.

*Пример*.

Синтезировать логическую схему в *базисе* ИЛИ*-*НЕ, соответствующую выражению

.

*Решение*

****Используя правило де Моргана, преобразуем исходное выражение таким образом, чтобы последней операцией было отрицание и в выражение были бы только операции ИЛИ.

*x*1 *x*2 *x*3

&

&

&

&

&

&

&

*y*

&

&

&

Рис. 2.9. Реализация логического выражения в базисе И–НЕ



Полученное выражение, представленное в виде вложенных операций ИЛИ-НЕ, позволяет легко синтезировать соответствующую логическую схему в заданном базисе, которая приведена на рисунке рис. 2.10.

*x*1 *x*2 *x*3

1

1

1

1

1

1

1

1

1

1

**1**

**1**

**1**

1

*y*

**1**

**1**

**1**

**1**

**1**

**1**

**1**

Рис. 2.10. Реализация логического выражения в базисе ИЛИ–НЕ

* 1. **Задания**

*Задание 1*. Синтезировать логическую схему по выражению согласно варианту в базисах

а) И, ИЛИ, НЕ,

б) И-НЕ,

в) ИЛИ-НЕ.

0. () + ();

1. ()()+;

2. ()();

3. ()() + ;

4. ()() + ;

5. () + ();

6. ()();

7. ()() + ;

8. ()();

9. ()+ ().

№ варианта соответствует последней цифре номера зачетной книжки студента.

*Задание 2*. Минимизировать методом Квайна и с помощью диаграммы Вейча логическое выражение соответственно варианту (1 и 2 – для нечетных и четных номеров зачетной книжки):

1.



2.



1. **ЦИФРОВОЙ ЛОГИЧЕСКИЙ УРОВЕНЬ** **АРХИТЕКТУРЫ ЦВМ**

4. 1. Элементы ЭВМ
      1. Логические элементы

### Большинство современных ЦВМ состоят из двух и более архитектурных уровней (до шести).

Элементы ЭВМ, являющиеся представителями низшего иерархического уровня архитектуры компьютера – цифрового логического уровня (схемотехнического), реализованные на радиотехнических деталях, представляют собой мельчайшие компоненты, на основе которых строятся более крупные составляющие вычислительной машины.

Можно выделить три основные разновидности элементов – логические элементы, запоминающие, специальные.

Логические элементы, так же как и элементы алгебры логики, реализуют логические функции, но эти функции, оставаясь сравнительно простыми, все же сложней, чем базовые функции в алгебре логики. В одном логическом элементе может быть реализовано несколько простых функций. Кроме того, логические элементы характеризуются дополнительными параметрами, такими, как количество входов, нагрузочная способность (количество входов других элементов, к которым можно подключать выход данного элемента).

На рис. 3.1 приведены примеры некоторых логических элементов.





*б*



*а*



**1**

**1**

**1**

**1**

**&**

**&**

**&**

**&**

**&**

**1**

Рис. 3.1 Примеры логических элементов:

*а* – ИЛИ-И-НЕ; *б* – И-ИЛИ-НЕ

На выходах элементов указаны логические выражения для выходных сигналов в соответствии с приведенными входными сигналами. На рис. 3,*б* приведен логический элемент с инверсными входами (в логическом выражении сигнал по такому входу используется в обратном значении).

Примеры реализации простейших логических элементов с помощью диодно*-*резисторной схемы приведены на рис. 3.2.

*+U*

*U*вх1. – *x*1

*U*вх2. – *x*2 *R*1

*+U*

*+ U+ U*

*U*вх1. – *x*1

*U*вх2. – *x*2 *R*1

*U*вых

(*у*= *х*1 + *х*2)

*U*вых

(*у*=*х*1*х*2)

*U*вых

(*у*= *х*1 + *х*2)

*U*вых

(*у*=*х*1*х*2)

*D*1 *D*2 *R*2

*D*1 *D*2 *R*2

*D*1*D*2*R*2

*U*вх1*. U* вх2

*U*вх1. *U*вх2

*U*вх1. *U*вх2

*а* *б*

Рис. 3.2. Электрические схемы реализации логическихэлементов:

*a*– реализация логических элементов "И"; *б* – реализация элемента "ИЛИ"

На рис. 3.2,*а* приведена реализация логических элементов И. Реализация элемента ИЛИ приведена на рис.3.2,*б*. Схемы логических элементов построены с условием, что логическая «1» соответствует высокому уровню («+»), а логический ноль – низкому уровню напряжения, близкому «земле». Это соответствие используется и в других реализациях. На рис. 3.2,*а* соотношение сопротивления резисторов *R*1 и *R*2 при заданном напряжении «+*U*» выбирается таким образом, что без учета шунтирующего действия диодных цепочек напряжение на выходе имеет значение высокого уровня (уровня, соответствующего логической «1»). Источники входных сигналов *U*вх1 и *U*вх2 имеют малое внутреннее сопротивление. Поэтому, если один или оба источника подают низкий уровень (логический «0»), то из-за шунтирующего воздействия диодных цепочек нарезисторе*R*2 на выходе будет иметь место низкий уровень напряжения, соответствующий логическому нулю. Высокий уровень на выходе (логическая «1») будет иметь место только тогда, когда на оба входа подаются единицы, так как соответствующие им высокие уровни напряжения закрывают оба диода. Таким образом, единица на выходе будет иметь место только тогда, когда и *x*1, и *x*2 имеют единичные значения. Это означает, что рассматриваемая схема реализует логику И.

Для схемы на рис. 3.2,*б* соотношение сопротивления резисторов *R*1 и *R*2 при заданном напряжении «+*U*» выбирается таким образом, что без учета воздействия диодных цепочек напряжение на выходе имеет значение низкого уровня (уровня, соответствующего логическому «0»). Если хотя бы один или оба источника входных сигналов подают высокий уровень (логическая «1»), то этот высокий уровень проходит через открытый диод и появляется на выходе. Низкий уровень, т.е. логический «0», будет иметь место только тогда, когда оба входных сигнала имеют низкий уровень. Это означает, что рассматриваемая схема реализует логику ИЛИ.

На рис.3.3 приведены примеры реализации логических функций НЕ (рис. 3.3,*а*) и ИЛИ-НЕ (рис. 3.3,*б*) на транзисторах. Транзисторы обозначены символом «*Т*».

+*U*

+*U*

*R*

*R*





*T*

*Т*1*Т*2 *Т*3

*х*3

*x*1

*x*1

*х*2

*а*

*б*

Рис. 3.3**.** Примеры реализации логических элементов:

*а* – элемент И-НЕ;*б* – элемент ИЛИ-НЕ

На рис. 3.3,*а* транзистор открыт, следовательно, на его коллекторе напряжение, близкое к нулевому уровню, будет иметься тогда, когда на его базе высокий уровень логической единицы, и, наоборот, транзистор закрыт, а следовательно на его коллекторе высокий уровень тогда, когда входной сигнал соответствует низкому уровню нуля. Выходом схемы является коллектор транзистора, поэтому выходной сигнал реализует функцию НЕ.

На рис. 3.3, *б* на выходе схемы «*y*» будет низкий уровень (логический нуль) тогда, когда открыт хотя бы один транзистор *T*1, *T*2, *T*3, т. е. тогда, когда хотя бы одна из входных переменных *x*1, *x*2, *x*3 имеет значение логической единицы. Это означает, что выходной сигнал «*y*» зависит от входных сигналов по логике ИЛИ-НЕ.

На рис. 3.4 приведены примеры реализации логических функции И-НЕ и функции И на транзисторах.

На рис. 3.4,*а* на выходе схемы «*y*» будет низкий уровень (логический нуль) только тогда, когда открыты оба транзистора *T*1, *T*2 , т.е. тогда, когда обе входные переменные *x*1, *x*2 имеют значение логической единицы. Это означает, что выходной сигнал «*y*» зависит от входных сигналов по логике И-НЕ.

На рис.3.4,*б* приведена схема, использующая много эмиттерный транзистор *T*3 Транзистор такого типа пропускает ток только тогда, когда имеет место высокий уровень на его базе и низкий уровень хотя бы на одном из его эмиттеров. В приведенной схеме на базу *T*3 подается постоянный высокий уровень (логическая константа, равная «1»). В этом случае на выходе схемы «*y*» будет низкий уровень (логический нуль) тогда, когда есть условия протекания тока хотя бы по одному из его эмиттеров, т.е. хотя бы одна из входных переменных *x*1, *x*2, *x*3 имеет значение логического нуля.

+*U*

*R*

+*U*

*y = x*1*x*2

*T*1

*R*

*x*2

*y = x*1*x*2*x*3

*T*2

*x*3

*x*2

*x*2

*x*1

+(1)

*а*

*б*

Рис. 3.4. Примеры реализации логических функций:

*а* – на транзисторной логике; *б* – на транзисторно-транзисторной логике

Если на все эмиттеры подается логическая единица, то *T*3 закрыт, а на выходе схемы имеет место высокий уровень напряжения, т.е. логическая единица. Это означает, что выходной сигнал «*y*» зависит от входных сигналов по логике И.

3.1.2. Запоминающие элементы

В качестве запоминающих элементов в ЦВМ используются триггеры.

Триггер – электронная схема, обладающая двумя устойчивыми состояниями. Переход из одного устойчивого состояния в другое происходит скачкообразно под воздействием управляющих сигналов. При этом также скачкообразно изменяется уровень напряжения на выходе триггера. Триггеры служат основой для построения регистров, счетчиков и других элементов, обладающих функцией хранения. Главной частью любого триггера является запоминающая ячейка (ЗЯ).

Схема запоминающей ячейки на элементах "И-НЕ" представлена на [рис. 1.5](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.5).

*S*

&

*Q*

1

&

*Q*

*R*

2

Рис. 3.5. Запоминающая ячейка на элементах "И-НЕ"

Входной сигнал *S* (Set) служит для установки ЗЯ в состояние "1" (*Q* = 1, = 0). Сигнал (Reset) устанавливает ЗЯ в состояние "0" (= 0, = 1). Активными значениями для них являются сигналы низкого уровня. Пусть на входы ЗЯ поданы сигналы= 0, = 1. Тогда при любом исходном состоянии ЗЯ на выходе элемента 1 установится высокий уровень напряжения. Так как на входы элемента 2 поступают значения и *R*, то на его выходе будет сигнал низкого уровня. Таким образом, ЗЯ перейдет в состояние "1".

Аналогично при = 1, = 0 запоминающая ячейка перейдет в состояние = 0, = 1, то есть в "0".

Если = 1, = 1, то состояние ЗЯ будет определяться ее предыдущим состоянием. Если ЗЯ находилась в состоянии "1", то сигнал = 0, поступая на вход элемента 1, подтвердит состояние его выхода = 1. На входы элемента 2 поступят сигналы только высокого уровня. Поэтому его выход будет находиться в состоянии = 0, то есть не изменится. Если ЗЯ находилась в состоянии "0", то сигнал *Q* = 0, поступая на вход элемента 2, подтвердит состояние его выхода *Q* = 1. В свою очередь, выход элемента 1 также останется без изменения. Таким образом, эта комбинация входных сигналов соответствует режиму хранения. Если на входы *S* и *R* поданы сигналы низкого уровня (*S* = *R* = 0), то сигнал на выходах элементов 1 и 2 будет высоким (*Q*= = 1). При переводе ЗЯ в режим хранения (*S* = *R* = 1), выходы элементов 1 и 2 могут установиться в произвольное состояние. Поэтому комбинация сигналов *S* = *R* = 0 на управляющих входах не используется.

Работа триггерной схемы определяется не таблицей истинности, как для комбинационной логической схемы, а таблицей переходов. Таблица переходов показывает изменение состояния триггера при изменении состояния входных сигналов в зависимости от его текущего состояния. Таблица переходов запоминающей ячейки, показанной на [рис. 3.5](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.5), представлена в [табл. 3.1](http://www.intuit.ru/department/hardware/archhard2/1/2.html#table.1.3).

# **Таблица переходов**

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 3.1 | | | |
| *S* | *R* | *Q*(*t*+1) | Функция |
| 0 | 0 | *х* | Запрещенная комбинация |
| 0 | 1 | 1 | Установка в "1" |
| 1 | 0 | 0 | Установка в "0" |
| 1 | 1 | *Q*(*t*) | Хранение |

Аналогичная запоминающая ячейка может быть построена на элементах "ИЛИ-НЕ".

Такие запоминающие ячейки можно рассматривать как простейшие асинхронные триггеры, на базе которых строятся синхронные триггерные схемы.

Триггеры можно классифицировать по различным признакам, например так, как это показано на [рис. 3.6](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.6).

ТРИГГЕРЫ

по типу синхронизации

по способу управления

по внутренней организации

асинхронные синхронные

синхронные

*RS*

одноступенчатые

двухступенчатые

*JK*

**…**

*D*

cдинамической

синхронизацией

cо статической

синхронизацией

Рис. 3.6.  Классификация триггерных схем

Триггер называется синхронным, если его таблица переходов хотя бы по одному управляющему входу реализуется под воздействием синхронизирующего сигнала. Рассмотрим организацию синхронного одноступенчатого триггера ([рис. 3.7](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.7)).

Основу синхронного одноступенчатого триггера составляет рассмотренная выше запоминающая ячейка (элементы 1, 2). Комбинационная схема преобразует управляющие сигналы триггера, а также, для некоторых типов триггеров, сигналы *Q*и с выходов ЗЯ в сигналы S и *R* на входах запоминающей ячейки.

**Комационная схембина**

Управляющие входы

**Синхросигнал**

*S*

*Q*

*R*

*Q*

&

1

&

2

Рис. 3.7.  Обобщенная схема синхронного одноступенчатого триггера

Синхросигнал *C* разрешает передачу на входы элементов 1 и 2 таких значений сигналов *S* и *R*, которые устанавливают ЗЯ в то или иное состояние. Неактивное значение синхросигнала обеспечивает на входах запоминающей ячейки состояние управляющих сигналов *S* = *R* = 1, что соответствует для нее режиму хранения.

Схема синхронного одноступенчатого RS-триггера приведена на [рис. 3.8](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.8). Его таблица переходов представлена в [табл. 3.2](http://www.intuit.ru/department/hardware/archhard2/1/2.html#table.1.4).

*S*

***S***

**&**

*Q*

**&**

**3**

**1**

***C***

**&**

**&**

*R*

*Q*

***R***

**2**

**4**

Рис. 3.8.  Синхронный одноступенчатый RS-триггер

# **Таблица переходов**

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 3.2 | | | |
| *R* | *S* | *Q*(*t*+1) | Функция |
| 0 | 0 | *Q*(*t*) | Хранение |
| 0 | 1 | 1 | Установка в "1" |
| 1 | 0 | 0 | Установка в "0" |
| 1 | 1 | *х* | Запрещенная комбинация |

Еще раз подчеркнем, что данная таблица переходов будет реализовываться лишь при активном уровне синхросигнала *C* (для данной организации это *C* = 1). При *C* = 0 выходы элементов 3 и 4 (см. [рис. 1.8](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.8)) будут в состоянии "1", что соответствует режиму хранения запоминающей ячейки, реализованной на элементах 1 и 2.

Представленный на [рис. 3.8](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.8) триггер имеет статическую синхронизацию, при которой управляющие сигналы активизируют входы *S* и *R* запоминающей ячейки во время высокого уровня сигнала на входе синхронизации. Его условно-графическое обозначение (УГО) приведено на [рис. 3.9,*а*](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.9). Условно-графические обозначения триггеров, использующих другие типы синхронизации, приведены на [рис. 1.9,*б,в,г*](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.9) (на примере RS-триггера).

На [рис. 3.9,*б*](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.9) представлено УГО триггера со статической синхронизацией в случае, если активный уровень синхросигнала низкий. Условно-графические обозначения триггеров с динамической синхронизацией показаны на [рис. 3.9,*в*](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.9) и [3.9,*г*](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.9). В первом случае изменение состояния триггера под воздействием поступивших управляющих сигналов происходит только в момент переключения синхронизирующего сигнала с низкого уровня на высокий, а во втором – при переключении с высокого на низкий уровень. При постоянном значении уровня синхросигнала состояние выхода триггера с динамической синхронизацией не меняется при любых изменениях управляющих сигналов на его входах.

*S*

*C*

*R*

*T*

*S*

*C*

*R*

*T*

*S*

*C*

*R*

*T*

*S*

*C*

*R*

*T*

а б в г

Рис. 3.9.  Условно-графические обозначения RS-триггера с различной

синхронизацией:

*а*–статическая синхронизация; *б*– статическая инверсная синхронизация; *в*– динамическая синхронизация передним фронтом синхросигнала;*г*– динамическая синхронизация задним фронтом синхросигнала

Идеализированная (без учета задержек) временная диаграмма работы *RS*-триггеров с различными типами синхронизации приведена на [рис. 1.10](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.10).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***C*** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ***R*** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ***S*** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ***Q*** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ***Q6*** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ***Q6*** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ***Q1*** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Рис. 3.10.  Временная диаграмма работы RS-триггера с различными типами синхронизации:

*Qа*– статическая синхронизация; *Qб*– статическая инверсная синхронизация; *Qв* –динамическая синхронизация передним фронтом синхросигнала;*Qг*– динамическая синхронизация задним фронтом синхросигнала

Как отмечалось выше, синхронный триггер, помимо управляющих входов, воздействующих на его состояние при подаче сигнала синхронизации, может иметь входы, которые воздействуют на его состояние непосредственно. Обычно они используются для установки триггера в то или иное начальное состояние перед подачей последовательности синхросигналов. Схема синхронного RS-триггера с асинхронными входами установки в "0" и в "1" приведена на [рис. 3.11](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.11), а его условно-графическое обозначение – на [рис. 3.12](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.12).

*S*

&

&

*S*

*Q*

1

3

*C*

&

&

*Q*

*R*

4

2

*R*

Рис. 3.11.  Синхронный одноступенчатый RS-триггер с асинхронными установочными входами

S

S

C

R

R

T

Рис. 3.12.  Условно-графическое обозначение синхронного одноступенчатого

RS-триггера с асинхронными установочными входами

Сигналы, поступающие по асинхронным входам *S* и *R*, подаются непосредственно на входы запоминающей ячейки, образованной элементами 1 и 2, минуя цепь, управляемую синхросигналом (элементы 1 и 2), и вызывают переключение запоминающей ячейки согласно [табл. 3.3](http://www.intuit.ru/department/hardware/archhard2/1/2.html#table.1.3).

Триггеры некоторых типов используют значения выходного сигнала для формирования управляющих сигналов на входах запоминающей ячейки (см. [рис. 3.7](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.7)). Это может привести к непредсказуемой последовательности его переключений. При построении отдельных схем на основе триггеров, например,регистров сдвига, необходимо, чтобы значения выходных сигналов триггера не изменялись на то время, пока производится их запись и значения его выходных сигналов в другой триггер, но сам этот триггер должен воспринимать значения с выхода некоторой другой триггерной схемы. Эти, а также некоторые другие ситуации требуют особых подходов к организации триггера, основным из которых является создание двухступенчатых триггеров.

Двухступенчатый RS-триггер ([рис. 3.13](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.13) и [рис. 3.14](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.14)) строится на основе двух одноступенчатых триггеров с прямой статической синхронизацией. Информация в первую ступень триггера (элемент 1) заносится во время действия высокого уровня синхросигнала. После того как синхросигнал на входе принимает низкое значение, элемент 1 переходит в режим хранения, а значение высокого сигнала на выходе инвертора 3 обеспечивает запись состояния триггера 1 в триггер 2. Идеализированная временная диаграмма работы двухступенчатого RS-триггера приведена на [рис. 3.15](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.15).

Q2

*Q1*

*qS2*

*qS1*

*T*

*1*

T

2

*S*

*C*

*R*

*S*

*C*

*R*

Q2

*Q’*

*CИ*

*qR2*

*qR1*

*CИ*

&

Рис. 3.13. Схема двухступенчатого RS-триггера

*qR*

*qS*

*CИ*

*S*

*C*

*R*

*TT*

*Q*

*Q*

Рис. 3.14.  Условно-графическое обозначение двухступенчатого RS-триггера

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *C* |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| *C* |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| *S* |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| *R* |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| *Q’* |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| *Q’* |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| *Q* |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| *Q* |  |  |  |  |  |  |  |  |  |

Рис. 3.15.Временная диаграмма работы двухступенчатого RS-триггера

Следует отметить, что первая ступень представляет собой одноступенчатый триггер, реализующий заданную таблицу переходов, в то время как вторая ступень – это всегда одноступенчатый синхронный RS-триггер. Например, на [рис. 3.16](http://www.intuit.ru/department/hardware/archhard2/1/2.html#image.1.16) показана схема двухступенчатого JK-триггера.

*Q*

*J*

*Q*

*T*

2

*T*

1

*J*

*C*

*K*

*S*

*C*

*R*

*C*

*Q*,

*K*

*Q*

*C*

1

3

Рис. 3.16.  Двухступенчатый JK-триггер

***Т-триггер.***Т-триггерпредставляет собой триггер, имеющий один вход «Т», поступление единичного сигнала на который переводит Т-триггер в состояние, противоположное его исходному состоянию (фигурально говоря, по каждому входному сигналу триггер «кувыркается», меняя свое состояние на противоположное). На рис. 3.17 приведена реализация Т-триггера на базе двухтактного RS-триггера (*а*) и временная диаграмма его работы (*б*). Имеющиеся на схеме обратные связи создают ситуацию, при которой сигналы на входах *R* и *S* стремятся перевести триггер в состояние, противоположное текущему. Поэтому при приходе очередного сигнала *qT* триггер воспринимает сигналы, имеющиеся на его входах. Выходные сигналы триггера изменяются после снятия единичного сигнала на его входе *qT*, так как триггер двухтактный.

*Q*

*q*t

*Q*

*TT*

*S*

*C*

*R*

R

*q*t

*а б*

Рис. 3.17. T-триггер:

*а –* условно-графическое обозначение; *б –* временная диаграмма

Т-триггер можно рассматривать как счетчик, считающий по модулю два количество импульсов, поступающих на его вход. Действительно, если в исходном состоянии триггер находится в «0», то при поступлении на его вход нечетного количества импульсов триггер будет находиться в «1», а при четном в– «0», что соответствует суммированию по модулю «2» количества поступающих импульсов.

***JK-триггер.*** Реализация JK-триггера и соответствующая таблица истинности приведена на рис. 3.18. Вход «*J*» – это вход установки «1», вход «*K*» – вход установки «0».

Из приведенной схемы видно, что сигналы *qJ*или*qK*, стремящиеся установить триггер, соответственно, в «1» или «0», поступают на соответствующий вход «*S*» или «*R*» базового RS-триггера только тогда, когда его исходное состояние противоположно тому, в которое стремится перевести JK-триггер комбинация входных сигналов. В противном случае сигналы *qJ*или*qK* на соответствующий вход *S* или *R* базового триггера не поступают. В связи с этим комбинация входных сигналов «1, 1» не является запретной, так как в этом случае на соответствующий вход базового RS-триггера поступит только тот сигнал, который стремится установить триггер в состояние, противоположное его исходному состоянию. Этот момент отражен в таблице истинности:при комбинации входных сигналов «1,1» триггер меняет исходное состояние, то есть работает как T-триггер.

|  |  |  |  |
| --- | --- | --- | --- |
|  | *qJ* | *qK* | *Q*(*t*+1) |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 1 |
| 3 | 1 | 1 | *Q(t)* |

*&*

*TT*

*Q*

*Q*

**Q**

**\_**

**Q**

*S*

*C*

*R*

**R**

*q*J

СИ

*q*k

*&*

*а* *б*

Рис.3.18. JK-триггер:

*а* – схема реализации; *б* – таблица переходов

Таким образом, JK-триггер представляет собой универсальный триггер, объединяющий в себе свойства и RS-триггера и Т-триггера.

***D-триггер.***D-триггер подругому называют элементом задержки. Схема его реализации на базе RS-триггера приведена на рис. 3.19. Использование подачи сигнала установки «1» через логику НЕ на вход установки «0» приводит к тому, что на входы *R* и *S* базового RS-триггера подаются сигналы, имеющие противоположные значения.

*q*D

*S*

*C*

*R*

*T*

**\_**

**Q**

*Q*

*Q*

**\_**

**Q**

*СИ*

&

Рис.3.19. D-триггер

Это означает, что, когда есть «1» на входе *q*D, на входах базового триггера будут сигналы: «1»***–***на входе *S* и «0»***–***на входе *R*. Поэтому по переднему фронту сигнала СИ в триггере устанавливается «1», если есть «1» на входе *q*D, в противном случае в триггере будет установлен «0». Состояние, которое имеется у триггера в момент заднего фронда сигнала СИ, будет сохранятьcя («задерживаться») до поступления очередного сигнала синхронизации СИ. Таблицы переходов JK- и D-триггеров приведены в таблиц.[3.](http://www.intuit.ru/department/hardware/archhard2/1/2.html#table.1.5)3 и [3.](http://www.intuit.ru/department/hardware/archhard2/1/2.html#table.1.6)4 соответственно.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 3.3  **Таблица переходов JK-триггера** | | | | |
| *J* | | *K* | *Q*(*t*+1) | Функция |
| 0 | | 0 | *Q*(*t*) | Хранение |
| 0 | | 1 | 0 | Установка в "0" |
| 1 | | 0 | 1 | Установка в "1" |
| 1 | | 1 | *Q*(*t*) | Инвертирование предыдущего состояния |
| Таблица 3.4  **Таблица переходов D-триггера** | | | | |
| *D* | *Q*(*t*+1) | | Функция | |
| 0 | 0 | | Установка в "0" | |
| 1 | 1 | | Установка в "1" | |

## **3.2. УЗЛЫ ЭВМ**

3.2.1. Комбинационные узлы

Узлы ЭВМ можно подразделить на два типа – комбинационные и накапливающие.

Узел ЭВМ представляет собой совокупность нескольких логических схем и, в общем случае, элементов памяти, формирующих выходные сигналы, соответствующие нескольким логическим функциям от входных сигналов.

Характерной особенностью узлов *комбинационного* типа является то, что их выходные сигналы определяются только действующими в данный момент входными сигналами (не зависят от «истории» входных сигналов).

Характерной особенностью узлов *накапливающего* типа является то, что их выходные сигналы определяются не только действующими в данный момент входными сигналами, но и тем, какие входные сигналы поступали на узел ранее, т.е. зависят от «истории» входных сигналов. Свойство хранить историю обеспечивается у накапливающих узлов наличием памяти, представленной некоторой совокупностью запоминающих элементов.

Рассмотрим некоторые примеры комбинационных узлов.

Дешифратор. Дешифратором называется комбинационная схема, имеющая *n* входов и 2*n* выходов и преобразующая двоичный код на своих входах в унитарный код на выходах. Унитарным называется двоичный код, содержащий одну и только одну единицу, например 00100000. Условно-графическое обозначение дешифратора на три входа приведено на [рис.](http://www.intuit.ru/department/hardware/archhard2/1/#image.1.1) 3.20.

0

1

2

0

1

2

0DC

3

5

4

6

7

Рис. 3.20.Условно-графическое обозначение трехвходового дешифратора

Номер разряда, в котором устанавливается "1" на выходе дешифратора, определяется кодом на его входах. Ниже приведена таблица истинности трехвходовогодешифратора ([таблиц. 3.](http://www.intuit.ru/department/hardware/archhard2/1/#table.1.1)5).

|  |
| --- |
| Таблица 3.5  **Таблица истинности трехвходового дешифратора** |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Входы | | | Выходы | | | | | | | |
| 2 | 1 | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Реализация дешифратора в одноэлементном базисе «Штрих Шеффера» достаточно проста, так как таблица истинности для любого выхода имеет только одну единицу. На [рис. 3.2](http://www.intuit.ru/department/hardware/archhard2/1/#image.1.2)1 представлена схема формирования сигнала на одном из выходов дешифратора (сигнал *f*5 на выходе 5).

*x*0

\_\_

*f*1

*x*1

*x*1

&

&

*f*5

&

*x*2

1

2

3

Рис. 3.21*.* Схема формирования сигнала на выходе 5 трехвходового дешифратора

Из представленной схемы видно, что фактически логику преобразования выполняет лишь элемент 2, в то время как элемент 1 служит для получения инверсии сигнала *x*1, а элемент 3 преобразует полученное на элементе 2 инверсное значение функции в прямое. Многие элементы хранения, например триггерные схемы, позволяют получать сигнал в парафазном коде, то есть имеют два выхода, на одном из которых сигнал имеет прямое, а на другом – инверсное значение. Это позволяет избавиться от элемента 1 в схеме. Если предположить, что значения выходных сигналов имеют инверсный вид по отношению к представленному в [табл. 3.](http://www.intuit.ru/department/hardware/archhard2/1/#table.1.1)5, то отпадает необходимость в элементе 3. В большинстве реальных интегральных микросхем реализованы именно дешифраторы с инверсными выходами. Обозначение такого дешифратора показано на [рис. 3.](http://www.intuit.ru/department/hardware/archhard2/1/#image.1.3)22.

0

1

2

3

4

5

6

7

DC

0

1

2

Рис. 3.22.Условно-графическое обозначение дешифратора с инверсными выходами

На выходах такого дешифратора образуется унитарный код, содержащий один и только один ноль. Например, если входные сигналы имеют значение 1102=610, то выходы дешифратора, представленного на [рис. 1.3](http://www.intuit.ru/department/hardware/archhard2/1/#image.1.3), будут находиться в состоянии 10111111, то есть выход 6 будет иметь значение, отличное от остальных выходов.

Дешифраторы широко применяются в различных устройствах компьютеров. Прежде всего, они используются для выбора ячейки запоминающего устройства, к которой производится обращение для записи или считывания информации. При этом часть разрядов адресного кода может дешифрироваться дешифраторами, выполненными в виде отдельных интегральных схем, а другая часть разрядов (обычно младшая) дешифрируется с помощью дешифраторов, встроенных непосредственно в БИС запоминающего устройства. Кроме того, дешифраторы находят применение в устройстве управления для определения выполняемой операции, построения распределителей импульсов и в других блоках.

Шифратор. Шифратор – схема, имеющая 2*n* входов и *n* выходов, функции которой во многом противоположны функции дешифратора ([рис. 3.2](http://www.intuit.ru/department/hardware/archhard2/1/#image.1.4)3). Эта комбинационная схема в соответствии с унитарным кодом на своих входах формирует позиционный код на выходе ([табл. 3.](http://www.intuit.ru/department/hardware/archhard2/1/#table.1.2)6).

0*CD*

1

2

3

0

1

Рис. 3.23.Условно-графическое обозначение шифратора на 4 входа

|  |
| --- |
| Таблица 3.6  **Таблица истинности шифратора на 4 входа** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Входы | | | | Выходы | |
| 3 | 2 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | - | 0 | 1 |
| 0 | 1 | - | - | 1 | 0 |
| 1 | - | - | - | 1 | 1 |

Сумматор по модулю 2. Сумматор по модулю 2 вырабатывает на своем входе сигнал логической единицы, если количество его входов с сигналом логической единицы нечетное. На рис. 3.24,*а* приведена схема сумматора по модулю 2 на два входа, а на рис. 3.24,*б* – его условное обозначение. На рис. 3.24,*в* приведен сумматор по моду­лю 2 с восьмью входами, построенный по принципу каскадирования из двухвхо­довых сумматоров по модулю 2.

Сигнал на выходе схемы (см. рис. 3.3,*в*) *у* определяется логическим выра­жением

.

&

y=x1x2+x2x1

1

m2

*x*1

*x*2

*y*

&

*а б*

m2

m2

*x*1

*x*2

*x*3

*x*4

*x*5

*x*6

*x*7

*x*8

m2

m2

*y*

m2

m2

m2

*в*

Рис.3.24. Сумматор по модулю 2:

*а –* схема на логических элементах; *б –* условно-графическое изображение;

*в*– схема восьмивходового сумматора по модулю 2

Мультиплексор. Мультиплексор реализует функцию подключения одного из нескольких входов к единственному выходу. На рис. 3.25,а приведена схема мультиплексора, а на рис. 3.25,б  *–*  его условное обозначение.

&

&

&

&

Выход

1

Вход 1

Вход 4

Вход 3

Вход 2

1

2

3

4

1

2

с

DC

*y*1

*y*2

*а*

**MS**

Вход

Выход 1

*y*

*б*

3.25. Мультиплексор:

*а*– реализация на основе дешифратора; *б* – условно-графическое обозначение

Приведенная схема осуществляет коммутацию четырех *одноразрядных* вхо­дов на один *одноразрядный* выход. Входы управления у1, у2 определяют номер одного из че­тырех входов, который нужно логически соединить с выходом.

При необходимости построения мультиплексора, обеспечивающего коммутацию четырех *п*-разрядных входов на один *п*-разрядный выход, необходимо использовать параллельное подключение к выходу одного дешифратора *n* логических элементов И-ИЛИ.

1

&

1

&

вход 1,np

1

&

выход 1-го p

вход 1,1p

1

2

*y*1

*y*2

c

1

2

3

4

DC

&

&

&

&

1

вход 2,1p

выход n-го p

вход 3,1p

выход 3-го p

выход 2-го p

СИ

вход 4,1p

Демультипексор. Демультиплексор выполняет функцию логического подключения одного входного канала к одному из нескольких выходных каналов, т. е. его функция яв­ляется обратной по сравнению с функцией, реализуемой мультиплексором. На рис. 1.26,а приведена схема демультиплексора, реализованного на дешифраторе.

Выход 1

Выход 2

Выход *m*

Вход

1

2

.

.

.

*m*

1

2

.

n

*c*

**DC**

*y*1

*y*2

*yn*

*а*

*a*

Выход 1, *np*

1

2

.

.

.

*m*

1

23

.

n

*c*

**DC**

1

2

.

.

.

*m*

1

23

.

n

*c*

**DC**

1

2

.

.

.

*m*

1

23

.

*n*

*c*

**DC**

Выход 2, *np*

Выход *m*, *np*

3p

2p

Выход *m*, 2*p*

Вход , 1*p*

v

1p

Выход *m*, 1*p*

*б*

Рис. 3.26. Схема демультиплексора:

*а –* реализация на основе дишифратора; *б –* демультиплексор на*n*разрядов

На входы*y*1…,*y*nподается код номера входа, к которому нужно логически подсоединить вход демультиплексора, которым является вход синхронизации дешифратора. Демультиплексор, приведенный на схеме, осуществляет коммутацию одного одноразрядного входа на один из четырех одноразрядных выходов.

На рис. 3.26, *б* приведена схема демультиплексора, обеспечивающего комму­тацию *n*-разрядного входа на один из *m n*-разрядных выходов. Схема включает *m* дешифраторов (по числу разрядности входного и выходных каналов). Каждый дешифратор имеет по *m* выходов (по количеству выходов демультиплексора). Разрядами коммутируемого входа являются входы синхронизации соответствую­щих дешифраторов. Одноименные информационные входы дешифраторов объе­динены, на них подаются соответствующие разряды кода, определяющего номер выходного канала.

*Сумматор*

Одноразрядный *двоичный сумматор* обеспечивает сложение одноименных разрядов операндов с учётом переноса, поступающего из ближайшего младшего разряда. Сумматор вырабатывает значение соответствующего разряда суммы (*S*) и перенос (*P*), который должен быть учтен в соседнемстаршем разряде. Синтез схемы реализующей функции одноразрядного сумматора можно выполнить на основании таблицы истинности, приведенной на Рис. 0‑1.

Исходя из реализуемой функции, сумматор представляет собой логический узел с двумя выходами (выход суммы S и выход переноса Р, имеющую три входа:

а - разряд первого операнда;

b - разряд второго операнда;

р - перенос из младшего разряда.

На основании таблицы истинности можно записать логические выражения для формируемых суммы и переноса, которые будут иметь вид:

N

п.п.

**a**

**b**

**p**

**P**

**S**

**0**

**0**

**1**

**0**

**1**

**1**

**0**

**0**

**0**

**0**

**0**

**0**

**0**

**1**

**0**

**0**

**2**

**1**

**0**

**1**

**1**

**1**

**3**

**0**

**1**

**0**

**0**

**0**

**4**

**1**

**1**

**0**

**1**

**1**

**5**

**0**

**1**

**1**

**0**

**1**

**6**

**0**

**1**

**1**

**1**

**1**

**7**

**1**



Полученные функции наиболее удобно минимизировать с помощью карты Карно, так как количество переменных невелико. Карты Карно с представленными в них функциями S и P приведены на Рис. **0‑2** a) и b)

На основании представления функции S в карте можно заключить, что логическое выражение для этой функции не минимизируется.

Минимизированная функция переноса с учетов введенных контуров имеет вид:

P = ab + ap + dp.

b

b

0

0

1

0

0

1

1

1

0

1

0

1

1

0

1

0

**«**S» «P»

a

a

p

p

b )

a )

Рис. 3.27

В виду того, что функция P и S формируются в одном и том же узле, при формировании S целесообразно использовать средства, примененные для реализации функции Р. С этой целью рассмотрим функцию Р как переменную для функции S. Тогда модифицированная функция S, зависящая теперь от четырех переменных a,b,p,P, будет записываться в карту Карно для четырех переменных. На рис 2.2.9-априведена такая карта с записью в ней функции S (а) и 4 контура, используемые для её минимизации (b). В приведенной карте часть клеток, соответствующих наборам переменных, на которых функция не определена, заполнена отметкой «-». Таких клеток восемь. К их числу относятся клетки, соответствующие следующим наборам переменных:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| \_ \_  *aвр P,*  (8) | \_ \_  *авр* ,  (9) | \_ \_  *аврР*,  (10) | \_ \_  *аврР*,  (11) | \_ \_  *аврР*,  (12) | \_ \_  *аврР*,  (13) | \_ \_ \_  *аврР*,  (14) | \_  *аврР*.  (15) |

В наборах 11 – 13, 15 одновременно присутствуют единичные значения на двух иболее из трех переменных a,b,p и есть «0» переменной Р, что не возможно исходя из логики формирования Р по переменным a,b,p.

На наборах 8 - 10, 14 присутствует единичные значения не более, чем у одной из трех переменных a,b,p и есть «1» переменной Р, что не возможно исходя из логики формирования Р по переменным a,b,p.

При охвате клеток контурами, клетки с отметкой «-» можно включать в контура наряду с клетками, имеющими единичные значения. На основании четырех контуров на карте, приведенной на рис.2.2.9-b), можно составить минимизированное логическое выражение для функции S, которое имеет вид:



4 2 1 3

Таким образом, определение функции S, как функции четырех переменных, позволило получить для её представления более простое выражение, чем исходная СДНФ для этой функции.

*-*

1

-

-

1

-

-

0

-

0

0

1

-

0

1

b

a

p

p

-

1

3

2

4

-

-

-

1

1

1

-

0

-

-

0

-

0

0

1

-

p

p

b

a

*а б*

Рис.3.28. Карта Карно для модифицированной функции *S*:

*а* – карта для функции 4-х переменных; *б* – карта с контурами

Рис. 0‑3 приведена схема одноразрядного двоичного сумматора, реализующая выведенные логические выражения для суммы *S* и переноса *P*.

p

&

&

&

1

&

&

1

1

&

P

S

a

b

1

2

3

SM

S

P

*а б*

**Рис.3.29**

Многоразрядный двоичный сумматор строится на основе одноразрядных сумматоров с введением соответствующих связей между разрядами. На рис. 3.29 приведена простейшая схема такого сумматора. На схеме приведена часть сумматора, относящаяся к некоторому *i*-ому разряду и его соседей: (*i*+1)-й соседний младший разряд и (*i*-1)-й соседний старший разряд.

Приведенная схема многоразрядного сумматора называется сумматором с последовательным переносом. Схема очень простая, но обладает малым быстродействием из-за последовательного учета переноса, возникшего в младшем разряде, в непрерывной цепочки старших разрядов, имеющих значение поразрядной суммы, равное единицы (такие разряды называются «разряды, пропускающие перенос»). Худший случай имеет место тогда, когда перенос, возникший в младшем разряде, распространяется до самого старшего разряда формируемой суммы.

(i+1)-ый разряд

i-ый разряд

(i-1)-ый разряд

От (i+2)-го

ai+1

bi+1

Si+1

1

2

3

SM

S

P

a

bi

Si

1

2

3

SM

S

P

ai-1

bi-1

Si-1

1

2

3

SM

S

P

к (i-2)-му

Выход суммы

Разряды операндов

Рис. 3.30

На Рис. 0‑4 представлена схема сумматора со сквозным переносом. В этом сумматоре, перенос, пришедший из младшего разряда на сумматор i-ого разряда, поступает на третий вход этого сумматора и одновременно, если поразрядная сумма, сформированная в i-ом сумматоре, равна «1», проходит на следующий (i-1)-ый сумматор.

Разряды суммы

(i +1) -ыйi -ый (i-1) -ый

**1**

**1**

**1**

к (i -2)-ому

от(i+2)-го

(i+2)-го

**&**

**&**

**&**

*S*

**P**

*SM*

1

2

3

*S*

**P**

*SM*

1

2

3

*S*

**P**

*SM*

1

2

3

(i+1)-ый разряд i-ый разряд (i-1)-ый разряд

a(i+1). ai. a(i-1).

b(i+1). bi. b(i-1).

Разряды операндов А и В

Схема работает в два такта.

На первом такте формируется поразрядная сумма и генерируются поразрядный перенос.

На втором такте разрешается распространение переносов по разрядам. При этом выработка сигналов переноса на отдельных сумматорах блокируется.

***Программируемая логическая матрица***

Программируемая логическая матрица (ПЛМ представляет собой комбинационный узел, обеспечивающий формирование нескольких функциональных зависимостей на основании заданных переменных. Вид функциональных зависимостей программируется.

Программируемая логическая матрица реализует функциональные зависимости в виде дизъюнкции простых конъюнкций. Структурная схема ПЛМ имеет вид, приведенный на Рис. 0‑5.

ПЛМ

выходные функции

Входные переменные

КМ

DМ

**Рис. 3.31**

В состав ПЛМ входят дизъюнктивная (ДМ) и конъюнктивная матрицы (КМ). КМ формирует множество не повторяющихся конъюнкций, используемых во всех формируемых логических функциях. ДМ для каждой выходной функции формирует логическую сумму соответствующих конъюнкций.

Пример ПЛМ приведен на . На пересечении горизонтальных и вертикальных шин конъюнктивной матрицы, обозначенных кружком, располагаются цепочки, состоящие из диода (D) и легкоплавкой перемычки (ЛП), в дизъюнктивной матрицы в кружках, обозначающих точку пересечения горизонтальных и вертикальных шин, располагаются цепочки, включающие транзистор (Т) и легкоплавкую перемычку. На горизонтальные шины конъюктивной матрицы помечены логическими выражениями формируемых ими конъюнкций. На каждой вертикальной шине дизъюнктивной матрицы реализована логика ИЛИ.

Приведенная матрица реализует следующую логику для выходных функций:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *у*1= | *x*1*x*2*x*3*x*4 **+** | **\_ \_**  *x*1*x*2*x*4 **+** | **\_**  *x*1*x*2**.** |  |  |  |
| *y*2= | *x*1*x*2*x*3*x*4**;** |  |  |  |  |  |
| *y*3= | *x*2*x*4 **+** | **\_ \_**  *x*1*x*2*x*4 **+** | **\_ \_**  *x*3*x*4**.** |  |  |  |
| *y*4= | *x*1*x*2*x*3*x*4 **+** | *x*2*x*4 **+** | **\_**  *x*1*x*2 **.** |  |  |  |
| *y*5= | **\_ \_ \_**  *x*1*x*2*x*3*x*4 **+** | **. \_ \_**  *x*1*x*2*x*3*x*4**.** |  |  |  |  |

Программирование ПЛМ выполняется следующим образом. При производстве создается одним из методов интегральной технологии заготовка ПЛМ, в которой на пересечениях горизонтальных и вертикальных шин имеет место цепочка - диодная в конъюнктивной и транзисторная в дизъюнктивной матицах. Пользователь в зависимости от логики, которую он собирается реализовать, удаляет «ненужные» перемычки. Удаление цепочки осуществляется посредством пропускания по соответствующей горизонтальной и вертикальной шинам мощного тока, который разогревает и испаряет соответствующую легкоплавкую перемычка.

. \_ \_ \_ \_ R1 *x*1*x*1 *x*2*x*2*x*3 *x*3*x*4*x*4

.



*y*1*y*2 *y*3*y*4*y*5

+U

**Рис. 3.32**

3.2.2. Накапливающие узлы

Для накапливающего узла характерна зависимость выходных сигналов не только от входных, но и от состояния, в котором находился данный узел в момент воздействия входного сигнала. Это означает, что такие узлы могут хранить «исто­рию» входных сигналов, т. е. узлы данного типа обладают памятью. К типовым на­капливающим узлам, используемым в вычислительной технике, относятся узлы: счетчики и регистры.

Счетчики. Счетчиком называется электронная схема, предназначенная для подсчета числа сигналов, поступающих на его счетный вход. Счетчики используются в устройстве управления компьютера при построении распределителей импульсов и организации циклов, в счетчиках команд для формирования адреса выполняемой команды при естественном порядке выполнения и в некоторых других устройствах ЭВМ. Также счетчики широко применяются в качестве самостоятельных узлов в различных системах цифровой автоматики.

Суть работы счетчика заключается в изменении на единицу зафиксированного в нем значения с приходом каждого счетного сигнала. Счетчики принято подразделять на суммирующие, вычитающие и реверсивные. Суммирующие счетчики увеличивают свое значение, вычитающие – уменьшают, а реверсивные могут работать как на прибавление, так и на вычитание в зависимости от сигналов управления. Параметром, определяющим информационную емкость счетчика, является модуль пересчета, равный числу внутренних состояний. Это значение проставляется на УГО после аббревиатуры CT.

Пример асинхронного трехразрядного двоичного суммирующего счетчика приведен на [3.33](http://www.intuit.ru/department/hardware/archhard2/2/#image.2.1), а его условно-графическое обозначение – на [рис.](http://www.intuit.ru/department/hardware/archhard2/2/#image.2.2) 3.34. Для построения этого счетчика использованы JK-триггеры с динамической синхронизацией по спаду синхросигнала. Каждый JK-триггер в счетчике включен в режим инвертирования своего состояния при переключении синхросигнала с высокого уровня на низкий (см. [табл. 3.5](http://www.intuit.ru/department/hardware/archhard2/1/2.html#table.1.5)). Идеализированная временная диаграмма работы этого счетчика показана на [рис.](http://www.intuit.ru/department/hardware/archhard2/2/#image.2.3) 3.35.

*Q*2

*T*

*T*

*T*

*J*

*C*

*K*

*J*

*C*

*K*

*J*

*C*

*K*

*Q*0

*Q*1

1

1

1

*C*

1

1

1

Рис. 3.33.  Схема асинхронного трехразрядного счетчика

СТ8

+1

1

0

2

Рис. 3.34.  Условно-графическое обозначение трехразрядного суммирующего счетчика

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | |  | 2 | |  | 3 | |  | 4 | |  | | 5 |  | | 6 |  | | 7 | |  | 8 |  | 9 |  |
| *C* |  | |  |  | |  |  | |  |  | |  | |  |  | |  |  | |  | |  |  |  |  |  |
|  |  | |  |  | |  |  | |  |  | |  | |  |  | |  |  | |  | |  |  |  |  |  |
| *Q*0 |  | | 1 | | | 0 | | | 1 | | | 0 | | | 1 | | | 0 | | | | 1 | | 0 | | 1 |
|  |  | |  |  | |  |  | |  |  | |  | |  |  | |  |  | |  | |  |  |  |  |  |
| *Q*1 | 0 | | | | | 1 | | | | | | 0 | | | | | | 1 | | | | | | 0 | | |
|  |  | |  |  | |  |  | |  |  | |  | |  |  | |  |  | |  | |  |  |  |  |  |
| Q2 | 0 | | | | | | | | | | | 1 | | | | | | | | | | | | 0 | | |
|  | |  | | |  | | |  | | |  | |  | | |  | | |  | |  | | |  | | |
| 0 | | 1 | | | 2 | | | 3 | | | 4 | | 5 | | | 6 | | | 7 | | 0 | | | 1 | | |

Рис.3.35.  Временная диаграмма работы счетчика

Быстродействие асинхронного счетчика определяется максимальной задержкой от изменения сигнала на его счетном входе до полного установления состояния всех его выходов. Проведем оценку быстродействия на примере переключения выходов счетчика после поступления восьмого синхросигнала на его вход ([рис.](http://www.intuit.ru/department/hardware/archhard2/2/#image.2.4) 1.30), так как именно в этом такте время переключения выходов счетчика будет максимальным.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 8 | |  |  |  |  |
| *С* |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| *Q*0 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| *Q*1 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| *Q*2 |  |  |  |  |  |  |
|  |  |  |  | =3 |  |  |

Рис. 3.36.  Оценка быстродействия асинхронного счетчика

По фронту 1-го сигнала*С*(8) с задержкой сигнала, равной *tT*, на триггере *Q*0 (см. [рис. 2.1](http://www.intuit.ru/department/hardware/archhard2/2/#image.2.1)) происходит изменение сигнала на выходе *Q*0. Это изменение, в свою очередь, приведет к переключению сигнала *Q*1 с соответствующей задержкой относительно переключения *Q*0. Вслед за этим с задержкой сигнала на следующем триггере переключится сигнал *Q*2. То есть общее время задержки переключения сигналов на выходе трехразрядного счетчика составит 3*tT*. Очевидно, что для *n*–разрядного счетчика время задержки составит

*t*CT = *nt*T.

Таким образом, с увеличением разрядности асинхронного счетчика увеличивается его задержка и, следовательно, уменьшается быстродействие.

Этот недостаток устраняется в счетчиках, работающих по синхронной схеме. В них за счет дополнительных управляющих комбинационных схем обеспечивается одновременное переключение всех разрядов при поступлении сигнала на счетный вход (с задержкой, равной задержке одного триггера вне зависимости от разрядности счетчика).

Обычно счетчик имеет вход установки в нулевое состояние (асинхронный сброс составляющих его триггеров в "0"). Некоторые счетчики имеют цепи установки в произвольное начальное состояние, с которого уже будет начинаться операция счета.

Регистры. Регистр – внутреннее запоминающее устройство процессора или внешнего устройства, предназначенное для временного хранения обрабатываемой или управляющей информации. Регистры представляют собой совокупность триггеров, количество которых равняется разрядности регистра, и вспомогательных схем, обеспечивающих выполнение некоторых элементарных операций. Набор этих операций, в зависимости от функционального назначения регистра, может включать в себя одновременную установку всех разрядов регистра в "0", параллельную или последовательную загрузку регистра, сдвиг содержимого регистра влево или вправо на требуемое число разрядов, управляемую выдачу информации из регистра (обычно используется при работе нескольких схем на общую шину данных) и т.д.

***Регистр хранения.*** Регистры хранения используются для приема, хранения и выдачи многоразрядного кода. Они представляют собой совокупность одноступенчатых триггеров (как правило, D-типа) с общим входом синхронизации. Иногда в регистре имеется также и общий вход асинхронной установки всех триггеров в "0". Схема четырехразрядного регистра хранения приведена на [рис.](http://www.intuit.ru/department/hardware/archhard2/2/2.html#image.2.5) 1.37, а его условно-графическое обозначение – на [рис.](http://www.intuit.ru/department/hardware/archhard2/2/2.html#image.2.6) 1.38.

*D T*

*C*

*R*

*x0*

*y0*

*x0*

*y0*

*x*1

D T

*C*

*R*

*y*1

*y*1

*x*2

*y*2

*y*2

*D T*

*C*

*R*

*x*3

*y*3

*y*3

*D T*

*C*

*R*

*C*

*C*

*R*

*R*

Рис. 1.37.  Структура четырехразрядного регистра хранения с асинхронным входом установки в "0"

0

1

2

3

*C*

*R*

0

1

2

3

*RG*

Рис. 1.38.  Условно-графическое обозначение четырехразрядного регистра хранения с асинхронным входом установки в "0"

Регистр сдвига. Регистр сдвига – регистр, обеспечивающий помимо хранения информации сдвиг влево или вправо всех разрядов одновременно на одинаковое число позиций. При этом выдвигаемые за пределы регистра разряды теряются, а в освобождающиеся разряды заносится информация, поступающая по отдельному внешнему входу регистра сдвига. Обычно эти регистры обеспечивают сдвиг кода на одну позицию влево или вправо. Но существуют и универсальные регистры сдвига, которые выполняют сдвиг как влево, так и вправо в зависимости от значения сигнала на специальном управляющем входе или при подаче синхросигналов на разные входы регистра. Регистр сдвига может быть спроектирован и таким образом, чтобы выполнять сдвиг одновременно не на одну, а на несколько позиций.

Регистры сдвига строятся на двухступенчатых триггерах. Схема четырехразрядного регистра, выполняющего сдвиг на один разряд от разряда 0 к разряду 3, показана на [рис.](http://www.intuit.ru/department/hardware/archhard2/2/2.html#image.2.7) 1.39, а его условно-графическое обозначение – на [рис.](http://www.intuit.ru/department/hardware/archhard2/2/2.html#image.2.8) 1.40. Ввод информации в данный регистр – последовательный через внешний вход D0. Регистр имеет вход асинхронной установки всех разрядов в "0". Для наглядности каждый двухступенчатый регистр представлен двумя одноступенчатыми с соответствующей организацией синхронизации первой и второй ступеней. Пунктиром обозначен реальный двухступенчатый триггер.

*Q0*

*T*

*S*

*C*

*R*

*R*

*T*

*S*

*C*

*R*

*R*

*Q0'*

*D*

*T*

*S*

*C*

*R*

*R*

*Q3'*

*С*

*R*

*T*

*S*

*C*

*R*

*R*

*Q3*

1

*T*

*S*

*C*

*R*

*R*

*Q1*

*Q1'*

*T*

*S*

*C*

*R*

*R*

*T*

*S*

*C*

*R*

*R*

*Q2*

*T*

*S*

*C*

*R*

*R*

*Q2'*

1

Рис. 3.39.  Структура регистра сдвига

*D RG*

0

*C*

*R*

*Q*

0

1

2

3

Рис. 3.40.  Условно-графическое обозначение четырехразрядного регистра сдвига с асинхронным входом установки в "0"

Идеализированная временная диаграмма работы регистра сдвига, структура которого представлена на [рис.](http://www.intuit.ru/department/hardware/archhard2/2/2.html#image.2.7) 3.40, показана на [рис.](http://www.intuit.ru/department/hardware/archhard2/2/2.html#image.2.9) 3.41. Предполагаем, что начальное состояние регистра следующее: *Q*0=0, *Q*1=1, *Q*2=1, *Q*3=0.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *C* |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| *R* |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| *D0* |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| *Q0'* |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| *Q0* |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| *Q1'* |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| *Q1* |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| *Q2'* |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| *Q2* |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| *Q3'* |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| *Q3* |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Рис. 3.41.  Временная диаграмма работы регистра сдвига

Работа регистра сдвига в каждом периоде сигнала синхронизации разбивается на две фазы: при высоком и при низком значении синхросигнала.

1. При высоком уровне синхросигнала проводится запись значения выхода (*i* – 1)-го разряда регистра в первую ступень *i*-го разряда. Вторая ступень каждого разряда сохраняет свое прежнее значение. В этой фазе состояние первой ступени *i*-готриггера повторяет состояние второй ступени (*i* – 1)-го триггера. Вторые ступени каждого триггера, а, следовательно, и выходы регистра в целом, остаются неизменными.

2. При низком уровне синхросигнала значение, записанное в первой ступени каждого триггера, перезаписывается в его вторую ступень. Запись в первую ступень триггера запрещена. В этой фазе состояния первой и второй ступеней каждого триггера становятся одинаковыми.

Поступление сигнала *R* = 0 вне зависимости от значения сигнала на входе синхронизации*С* и сигнала на входе *D*0 устанавливает все разряды регистра в нулевое состояние.

* 1. Устройства ЦВМ

Устройства ЭВМ представляют уровень микроархитектуры компьютера, отражающий, как и цифровой логический уровень, аппаратное обеспечение компьютера.

3.3.1. Арифметико-логическое устройство

Обобщенная структура АЛУ. Классическая ЭВМ состоит из трех основных устройств: арифметико-логического устройства, устройства управления и запоминающего устройства. Рассмотрим особенности организации этих устройств. Прежде всего, рассмотрим структуру арифметико-логического устройства.

Структурная схема АЛУ представлена на рис. 3.42.

Ш1

*Rn*

*Ra*

ОУк

ОУ1

*R2*

*yR*

*xR*

Упр. АЛУ

*R1*

Ш 2

ysm

хsm

Y

SM

Ш 3

МР

X

Cч

RF

yRF

хRF

усч

хсч

Рис. 1.36. Структура АЛУ

АЛУ состоит из следующих типовых узлов:

–регистры (R), служащие для хранения операндов и результатов;

–сумматор (SM), служащий для выполнения операции суммирования мно­горазрядных кодов;

–операционные узлы (ОУ), служащие для выполнения логических операций;

–мультиплексор (MS);

–счетчик (Сч), обеспечивающий подсчет тактов длинных операций;

–регистр флажков (RF), служащий для фиксации особой информации, ха­рактеризующей полученный результат.

Для передачи информации между отдельными узлами используются шины Ш1 – Ш3. Шина Ш3 обеспечивает также связь с запоминающими устройствам ЗУ (ЭВМ).

Управляющий блок осуществляет выработку множества управляющих сиг­налов Y, обеспечивающих выполнение элементарных операций (микроопераций) типовыми узлами операционного блока.

При работе управляющая часть АЛУ использует код заданной операции (например сложение, умножение, вычитание и т. п.), а также информацию о со­стоянии операционного блока, представленную в виде множества Х признаков, формируемых типовыми узлами.

К признакам, вырабатываемым регистром и посылаемым в управляющую часть, относятся:

– «ноль регистра» (*R*{0...*n*} = 0) характеризует состояние, при котором во всех разрядах регистра имеет место нулевое значение;

–«ноль знака» (*R*{зн} = 0) – в знаковом разряде регистра находится зна­чение 0;

–«единица старшего разряда» (*R*{1} = 1) – в старшем разряде регистра на­ходится значение единица;

–«единица младшего разряда» (*R*{*n*} = 1) – в младшем разряде регистра находится значение единица.

К микрооперациям, которые может выполнять регистр при поступлении со­ответствующего управляющего сигнала у*i*, относятся:

–прием кода;

–выдача прямого кода;

–выдача инверсного кода;

–установка единицы в некотором разряде регистра;

–обнуление знакового разряда;

–сдвиг кода влево;

–сдвиг кода вправо;

–обнуление регистра (во все разряды регистра устанавливается нулевое значение).

К признакам, вырабатываемым счетчиком и посылаемым в управляющую часть, относятся:

–«ноль счетчика» («0» Сч) –характеризует состояние, при котором во всех разрядах регистра имеет место нулевое значение;

–«переполнение счетчика» –при поступлении очередного счетного сигна­ла счетчик переходит от максимального значения к значению «0».

Счетчик может выполнять следующие операции, инициируемые по управ­ляющим сигналам, поступающим из управляющего блока:

–установка нуля в счетчике;

–установка в счетчике некоторого начального значения;

–установка режима счета (обратный или прямой счет);

–изменение находящегося в счетчике текущего значения на единицу.

К признакам, вырабатываемым сумматором и посылаемым в управляющую часть, относятся:

–признак нулевого результата;

–единичных значений во всех разрядах результата;

–единицы в первом знаковом разряде результата;

–единицы во втором знаковом разряде результата;

–переноса из старшего разряда сумматора;

–наличия в тетраде значения, большего 9;

–межтетрадного переноса.

Каждому из перечисленных состояний может соответствовать отдельный разряд (флажок) в регистре флажков.

Сумматор может выполнять следующие микрооперации, инициируемые по управляющим сигналам, поступающим из управляющего блока:

–прием кода двух операндов на свои входы;

–формирование поразрядной суммы операндов, поступающих на его входы;

–генерирование поразрядного переноса;

–распространение переносов через разряды поразрядной суммы, пропус­кающие перенос;

–прибавление единицыв младший разряд;

–прибавление корректирующих кодов в тетрады при сложении двоично-­десятичных кодов.

Выполнение любой арифметической операции в АЛУ реализуется за счет выполнения определенной последовательности микроопераций в узлах операци­онной части АЛУ. Такие последовательности образуют алгоритм выполнения операций на уровне микроопераций. Подробнее эти вопросы будут рассмотрены в следующей лекции.

Синтез АЛУ.В современных ЭВМ арифметико-логическое устройство не является самостоятельным схемотехническим блоком. Оно входит в состав микропроцессора, на котором строится компьютер. Однако знание структуры и принципов работы АЛУ весьма важно для понимания работы компьютера в целом.

Для лучшего понимания этих вопросов проведем синтез арифметического устройства, предназначенного для выполнения только одной операции – умножения чисел с фиксированной запятой, заданных в прямом коде, со старших разрядов множителя и сдвигом множимого. В ходе этого процесса также обратим внимание на особенности использования рассмотренных выше основных схемотехнических элементов ЭВМ.

Синтез АЛУ проходит в несколько этапов.

1. Выбор и/или описание метода, по которому предполагается выполнение операции.
2. Составление алгоритма соответствующих действий в соответствии с выбранным (заданным) методом.
3. Определение набора составляющих АЛУ функциональных элементов и узлов, а также разрядность узлов исходя из алгоритма и формата исходных данных.
4. Определение связей между элементами и узлами и отображение их на схеме.
5. Установление порядка функционирования устройства и разработка временной диаграммы управляющих сигналов, которые должны быть поданы на АЛУ от устройства управления.

Пусть операнды имеют вид:

[*X*]пк = *x*0*x*1*x*2…*xn;*

[*Y*]пк = *y*0*y*1*y*2…*yn,*

где *x*0, *y*0 – знаковые разряды.

Операция умножения чисел с фиксированной запятой, заданных в прямом коде, со старших разрядов множителя выполняется по следующей формуле:

Sign =Sign + Sign

||=

Sign *Z*= 0 + 1=1

Алгоритм вычислений представлен на [рис.](http://www.intuit.ru/department/hardware/archhard2/3/#image.3.1) 3.42.

Каждой переменной, представленной в алгоритме, в схеме должен соответствовать элемент хранения. Разрядность модуля произведения равна сумме разрядностей сомножителей. Умножение двоичного числа на2– *i* обеспечивается сдвигом этого числа вправо на соответствующее количество разрядов. Переход к анализу очередного разряда множителя (*i* = *i* + 1) может быть обеспечен сдвигом регистра множителя на один разряд в сторону старших разрядов.

Исходя из этого, определим состав оборудования, необходимого для реализации АЛУ заданного типа для *n* = 4 ([табл.](http://www.intuit.ru/department/hardware/archhard2/3/#table.3.1)3.7).

начало

*i = 1*

|*Z*| = 0

*УС1, УС3, УС6, УС7, УС8*

Такт 1

*SingZ = =SignX*⊕*SignY*

*УС9*

Такт 2

*|X|=|X|·2* -1

*УС9*

Такт 3

да

*y*i = 0

нет

*УС5*

*|Z|=|Z|+*0

*|Z|=|Z|+|X|*

*УС5*

*|X|=|X|·2* -1

*УС2*

*i=i+*1

*УС4*

Такт 5

*i* >*n*

нет

да

конец

Рис. 3.43.  Алгоритм операции умножения чисел с фиксированной запятой, заданных в прямом коде, со старших разрядов множителя

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 3.7  **Состав оборудования АЛУ** | | | |
| Схема | Разрядность | Функции | Управляющий сигнал |
| Регистр модуля множимого RGX | 8 | Загрузка. Сдвиг в сторону младших разрядов | УС1 УС2 |
| Регистр модуля множителя RGY | 4 | Загрузка. Сдвиг в сторону старших разрядов | УС3 УС4 |
| Регистр модуля результата RGZ | 8 | Загрузка. Установка в "0" | УС5 УС6 |
| Триггер знака множимого TX |  | Загрузка | УС7 |
| Триггер знака множителя TY |  | Загрузка | УС8 |
| Триггер знака результата TZ |  | Загрузка | УС9 |
| Сумматор | 8 | Суммирование содержимого регистра модуля результата с кодом множимого или с нулевым кодом в зависимости от значения *yi* | – |
| Комбинационные схемы |  | Получение на входе сумматора сигналов "0" или содержимого RGX в зависимости от значения *yi* | – |

Структурная схема устройства представлена на [рис.](http://www.intuit.ru/department/hardware/archhard2/3/2.html#image.3.2) 3.42.

Временная диаграмма управляющих сигналов, поступающих на арифметико-логическое устройство, показана на [рис.](http://www.intuit.ru/department/hardware/archhard2/3/2.html#image.3.3) 3.43.

SignY

УС8

D

C

Y

T

SignX

УС7

D

C

X

T

*D*1

0

1

2

3

*D*3

*C*1

*C*2

*Y*

*RG*

*DO*

0

1

2

3

|*Y*|

=I

SignZ

УС9

D

C

Z

T

Рис. 3.43.  Структурная схема арифметического устройства для выполнения операции умножения со старших разрядов множителя чисел, заданных в прямом коде

|*X*|

*D*1

0

1

2

3

4

5

6

7

*C*1

*C*2

*X*

*RG*

*DO*

*D*0

0

1

2

3

4

5

6

7

|  |
| --- |
| &  0 |
| &  1 |
| &  2 |
| &  3 |
| &  4 |
| &  5 |
| &  6 |
| &  7 |

УС1

УС2

0

1

2

3

4

5

6

7

С0

SM

0

1

2

3

4

5

6

7

*D*1

0

1

2

3

4

5

6

7

*C*1

R

Z

*RG*

DI

0

1

2

3

4

5

6

7

*DO*

0

1

2

3

4

5

6

7

УС5

УС6

0

1

2

3

4

5

6

7

0

1

2

3

4

5

6

7

0

1

2

3

4

5

6

7

0

1

2

3

4

5

6

7

Рис. 3.44(продолжение).  Структурная схема арифметического устройства для выполнения операции умножения со старших разрядов множителя чисел, заданных в прямом коде

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 |  | 2 |  | 3 |  | 4 |  | 5 |  | 6 |  | 7 |  | 8 |  | 9 |  | 10 |  | 11 |  |
| ТИ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| УС1,  УС3,  УС6,УС7,УС8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| УС9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| УС2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| УС4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| УС5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Рис. 3.45.  Временная диаграмма управляющих сигналов

***Работа схемы***

Такт 1. Загрузка модулей операндов в регистры RGX, RGY, а их знаков – в триггеры TX и TY. Сброс в "0" регистра результата RGZ.

Такт 2. Запись знака результата в триггер TZ.

Такт 3. Сдвиг регистра RGX на один разряд вправо. Через время, равное задержке на переключение регистров и комбинационных схем, на выходе комбинационного сумматора и, следовательно, на входе регистра RGZ устанавливается результат 0+y1• |X|•2-1.

Такт 4. Загрузка RGZ: |Z|=|Z|+y1• |X|•2-1.

Такт 5. Сдвиг RGX на 1 разряд вправо: |X| = |X|•2-1.

Сдвиг RGY на 1 разряд влево: i=i+1.

Устройство управления проверяет условие окончания операции: i>n.

Такты (6,7), (8,9), (10,11)... Повтор действий тактов (4,5) с анализом других значений *yi*. В такте 10 в регистре RGZ формируется модуль произведения. Такт 11 используется лишь для определения условия окончания операции умножения.

3.4. Устройство управления

Для реализации любой команды компьютера необходимо на соответствующие управляющие входы любого устройства компьютера подать определенным образом распределенную во времени последовательность управляющих сигналов. Часть цифрового вычислительного устройства, предназначенная для выработки этой последовательности, называется устройством управления.

Устройство управления. Коды операции команд программы, воспринимаемые управляющей частью микропроцессора, расшифрованные и преобразованные в ней, дают информацию о том, какие операции надо выполнить, где в памяти расположены данные, куда надо направить результат и где расположена следующая за выполняемой команда.

Управляющее устройство имеет достаточно средств для того, чтобы после восприятия и интерпретации информации, получаемой в команде, обеспечить переключение (срабатывание) всех требуемых функциональных частей машины, а также для того, чтобы подвести к ним данные и воспринять полученные результаты. Именно срабатывание, т. е. изменение состояния двоичных логических элементов на противоположное, позволяет посредством коммутации вентилей выполнять элементарные логические и арифметические действия, а также передавать требуемые операнды в функциональные части микроЭВМ.

Устройство управления в строгой последовательности в рамках тактовых и цикловых временных интервалов работы микропроцессора (такт **–** минимальный рабочий интервал, в течение которого совершается одно элементарное действие; цикл **–** интервал времени, в течение которого выполняется одна машинная операция) осуществляет: выборку команды; интерпретацию ее с целью анализа формата, служебных признаков и вычисления адреса операнда (операндов); установление номенклатуры и временной последовательности всех функциональных управляющих сигналов; генерацию управляющих импульсов и передачу их на управляющие шины функциональных частей микроЭВМ и вентили между ними; анализ результата операции и изменение своего состояния так, чтобы определить месторасположение (адрес) следующей команды.

Любое действие, выполняемое в операционном блоке, описывается некоторой микропрограммой и реализуется за один или несколько тактов. Элементарная функциональная операция, выполняемая за один тактовый интервал и приводимая в действие управляющим сигналом, называется микрооперацией. Например, в спроектированном АЛУ для умножения чисел в первом такте выполняются следующие микрооперации: TX = 0, TY = 0, RGX = |*X*|, RGY = |*Y*|, RGZ = 0. Совокупность микроопераций, выполняемых в одном такте, называется микрокомандой (МК). Если все такты должны иметь одну и ту же длину, а именно это имеет место при работе компьютера, то она устанавливается по самой продолжительной микрооперации. Микрокоманды, предназначенные для выполнения некоторой функционально законченной последовательности действий, образуют микропрограмму. Например, микропрограмму образует набор микрокоманд для выполнения команды умножения.

Устройство управления предназначено для выработки управляющих сигналов, под воздействием которых происходит преобразование информации в арифметико-логическом устройстве, а также операции по записи и чтению информации в/из запоминающего устройства.

Устройства управления делятся на:УУ с жесткой, или схемной логикой иУУ с программируемой логикой (микропрограммные УУ).

В устройствах управления первого типа для каждой команды, задаваемой кодом операции, строится набор комбинационных схем, которые в нужных тактах вырабатывают необходимые управляющие сигналы.

В микропрограммныхУУ каждой команде ставится в соответствие совокупность хранимых в специальной памяти слов **–** микрокоманд. Каждая из микрокоманд содержит информацию о микрооперациях, подлежащих выполнению в данном такте, и указание, какое слово должно быть выбрано из памяти в следующем такте.

Особенности программного и микропрограммного управления

В микропроцессорах используют два метода выработки совокупности функциональных управляющих сигналов: программный и микропрограммный.

Выполнение операций в машине сводится к элементарным преобразованиям информации (передача информации между узлами в блоках, сдвиг информации в узлах, логические поразрядные операции, проверка условий и т.д.) в логических элементах, узлах и блоках под воздействием функциональных управляющих сигналов блоков (устройств) управления. Элементарные преобразования, неразложимые на более простые, выполняются в течение одного такта сигналов синхронизации и называются микрооперациями.

В аппаратных (схемных) устройствах управления каждой операции соответствует свой набор логических схем, вырабатывающих определенные функциональные сигналы для выполнения микроопераций в определенные моменты времени. При этом способе построения устройства управления реализация микроопераций достигается за счет однажды соединенных между собой логических схем, поэтому ЭВМ с аппаратным устройством управления называют ЭВМ с жесткой логикой управления. Это понятие относится к фиксации системы команд в структуре связей ЭВМ и означает практическую невозможность каких-либо изменений в системе команд ЭВМ после ее изготовления.

При микропрограммной реализации устройства управления в состав последнего вводится ЗУ, каждый разряд выходного кода которого определяет появление определенного функционального сигнала управления. Поэтому каждой микрооперации ставится в соответствие свой информационный код **–**микрокоманда. Набор микрокоманд и последовательность их реализации обеспечивают выполнение любой сложной операции. Набор микроопераций называют микропрограммами. Способ управления операциями путем последовательного считывания и интерпретации микрокоманд из ЗУ (наиболее часто в виде микропрограммного ЗУ используют быстродействующие программируемые логические матрицы), а также использования кодов микрокоманд для генерации функциональных управляющих сигналов называют микропрограммным, а микроЭВМ с таким способом управления **–** микропрограммными или с хранимой (гибкой) логикой управления.

К микропрограммам предъявляют требования функциональной полноты и минимальности. Первое требование необходимо для обеспечения возможности разработки микропрограмм любых машинных операций, а второе связано с желанием уменьшить объем используемого оборудования. Учет фактора быстродействия ведет к расширению микропрограмм, поскольку усложнение последних позволяет сократить время выполнения команд программы.

Преобразование информации выполняется в универсальном арифметико-логическом блоке микропроцессора. Он обычно строится на основе комбинационных логических схем.

Для ускорения выполнения определенных операций вводятся дополнительно специальные операционные узлы (например, циклические сдвигатели). Кроме того, в состав микропроцессорного комплекта (МПК) БИС вводятся специализированные оперативные блоки арифметических расширителей.

Операционные возможности микропроцессора можно расширить за счет увеличения числа регистров. Если в регистровом буфере закрепление функций регистров отсутствует, то их можно использовать как для хранения данных, так и для хранения адресов. Подобные регистры микропроцессора называются регистрами общего назначения (РОН). По мере развития технологии реально осуществлено изготовление в микропроцессоре 16, 32 и более регистров.

В целом же, принцип микропрограммного управления (ПМУ) включает следующие позиции:

1) любая операция, реализуемая устройством, является последовательностью элементарных действий **–** микроопераций;

2) для управления порядком следования микроопераций используются логические условия;

3) процесс выполнения операций в устройстве описывается в форме алгоритма, представляемого в терминах микроопераций и логических условий, называемого микропрограммой;

4) микропрограмма используется как форма представления функции устройства, на основе которой определяются структура и порядок функционирования устройства во времени.

ПМУ обеспечивает гибкость микропроцессорной системы и позволяет осуществлять проблемную ориентацию микро- и миниЭВМ.

3.4.1. Схемное устройство управления.

Устройство управления схемного типа ([рис. 1.3](http://www.intuit.ru/department/hardware/archhard2/4/#image.4.1)9) состоит из:

– датчика сигналов, вырабатывающего последовательность импульсов, равномерно распределенную во времени по своим шинам ([рис. 1.4](http://www.intuit.ru/department/hardware/archhard2/4/#image.4.2)1) (*n***–**общее количество управляющих сигналов, необходимых для выполнения любой операции; *m***–** количество тактов, за которое выполняется самая длинная операция);

– блока управления операциями, осуществляющего выработку управляющих сигналов, то есть коммутацию сигналов, поступающих с ДС, в соответствующем такте на нужную управляющую шину;

– дешифратора кода операций, который дешифрирует код операции команды, присутствующей в данный момент в регистре команд, и возбуждает одну шину, соответствующую данной операции; этот сигнал используется блоком управления операциями для выработки нужной последовательности управляющих сигналов.

**ДАТЧИК СИГНАЛОВ**

***m***

**1**

**2 ………………………………………..**

**УС1**

**УС2**

**УСn**

**Регистр**

**команд**

**КОп**

**ДЕШИФРАТОР**

**КОДА ОПЕРАЦИЙ**

Рис. 3.46.  Функциональная схема схемного устройства управления

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  | … |  |  |  |  |  |  | … |  |  |  |  |
|  |  |  |  |  |  | … |  |  |  |  |  |  | … |  |  |  |  |
| 2 |  |  |  |  |  | … |  |  |  |  |  |  | … |  |  |  |  |
|  |  |  |  |  |  | … |  |  |  |  |  |  | … |  |  |  |  |
| 3 |  |  |  |  |  | … |  |  |  |  |  |  | … |  |  |  |  |
| … | … | … | … | … | … | … | … | … | … | … | … | … | … | … | … | … | … |
| *m* |  |  |  |  |  | … |  |  |  |  |  |  | … |  |  |  |  |

Рис. 3.47.  Временная диаграмма работы датчика сигналов

Датчик сигналов обычно реализуется на основе счетчика с дешифратором или на сдвиговом регистре.

**Датчик сигналов на основе счетчика с дешифратором.** Реализация датчика сигналов на основе счетчика с дешифратором представлена на [рис. 3.4](http://www.intuit.ru/department/hardware/archhard2/4/#image.4.3)8. По заднему фронту каждого тактового импульса, поступающего на устройство управления с системного генератора импульсов, счетчик увеличивает свое состояние; выходы счетчика соединены со входами дешифратора, выходы которого и являются выходами датчика сигналов ([рис. 4.3](http://www.intuit.ru/department/hardware/archhard2/4/#image.4.4)).

0

1

2

DC

0

1

2

3

4

5

6

7

**ТИ**

СТ

0

1

2

**+1**

Рис. 3.48.  Схема датчика сигналов на основе счетчика с дешифратором

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 |  | 2 |  | 3 |  | 4 |  | 5 |  | 6 |  | 7 |  | 8 |  | 9 |  |  |  |
| ТИ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| СТ 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| DCO |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Рис. 3.49.  Временная диаграмма работы датчика сигналов на основе счетчика с дешифратором

**Датчик сигналов на сдвиговом регистре**. Проектирование датчика сигналов на сдвиговом регистре требует лишь его <<закольцовывания>>, то есть соединения выхода последнего разряда с входом, через который в регистр заносится информация при сдвиге, и первоначальной установки ([рис. 1.](http://www.intuit.ru/department/hardware/archhard2/4/2.html#image.4.5)50). В начальном состоянии регистр содержит "1" только в разряде 0. Входы параллельной загрузки регистра для его начальной установки и соответствующий этой операции управляющий вход регистра на схеме не показаны.

0

1

2

3

4

5

6

7

*D0*

*RG*

ТИ

*C*

Рис. 3.50.  Схема датчика сигналов на основе регистра сдвига

Временная диаграмма работы этой схемы приведена на [рис. 3.](http://www.intuit.ru/department/hardware/archhard2/4/2.html#image.4.6)51.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 |  | 2 |  | 3 |  | 4 |  | 5 |  | 6 |  | 7 |  | 8 |  | 9 |  | 10 |
| ТИ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| RG0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Рис. 3.51.  Временная диаграмма работы датчика сигналов на основе регистра сдвига

Наиболее сложной частью схемного устройства управленияявляется блок управления операциями. Он представляет собой нерегулярную схему, структура которой определяется системой команд и составом оборудования процессора. Такое УУ может быть реализовано в виде специализированной интегральной схемы.

3.4.2. Структурная схема микропрограммного устройства управления.

Микропрограммное устройство управления представлено на [рис. 1.4](http://www.intuit.ru/department/hardware/archhard2/4/2.html#image.4.7)5. Преобразователь адреса микрокоманды преобразует код операции команды, присутствующей в данный момент в регистре команд, в начальный адрес микропрограммы, реализующей данную операцию, а также определяет адрес следующей микрокоманды, выполняемой микропрограммы по значению адресной части текущей микрокоманды.

Операционная часть МК

D

0

1

**.**

**.**

**.**

m

**.**

**.**

**.**

.

Память микропрограмм

A

*m*+1

Регистр адреса

Адресная часть МК

Преобразователь адреса микрокоманды

Сигнал записи новой команды в РК

Регистр команд

КОП

Рис. 3.52.  Функциональная схема микропрограммного устройства управления (УСi - управляющие сигналы, вырабатываемые устройством управления)

В[табл.](http://www.intuit.ru/department/hardware/archhard2/4/2.html#table.4.1)1. 8 приведен пример микропрограммы для выполнения операции умножения чисел в дополнительном коде. Предполагается, что начальный адрес микропрограммы равен 300, количество разрядов множителя равно 2, а адресная часть микрокоманды содержит адрес микрокоманды, которая должна быть выбрана в следующем такте. В последней микрокоманде в регистр команд загрузится очередная команда, код операции которой определит начальный адрес очередной микропрограммы. В реальных микропрограммных устройствах управления формирование адреса следующей микрокоманды проводится более сложным образом, учитывающим возможности ветвлений и циклического повторения отдельных фрагментов микропрограмм.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 3.8  **Микропрограмма выполнения операции умножения**   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Адрес МК | УС1 | УС2 | УС3 | УС4 | УС5 | УС6 | Сигнал записи в РК | Адрес следующей МК | | 300 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 301 | | 301 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 302 | | 302 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 303 | | 303 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 304 | | 304 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | Х | |

Из анализа структуры и принципов работы схемного и микропрограммного   устройств управления видно, что УУ первого типа имеют сложную нерегулярную структуру, которая требует специальной разработки для каждой системы команд и должна практически полностью перерабатываться при любых модификациях системы команд. В то же время оно имеет достаточно высокое быстродействие, определяемое быстродействием используемого элементного базиса.

Устройство управления, реализованное по микропрограммному принципу, может легко настраиваться на возможные изменения в операционной части ЭВМ. При этом настройка во многом сводится лишь к замене микропрограммной памяти. Однако УУ этого типа обладают худшими временными показателями по сравнению с устройствами управления на жесткой логике.

3.5. Запоминающие устройства

Памятью ЭВМ называется совокупность устройств, служащих для запоминания, хранения и выдачи информации.

Отдельные устройства, входящие в эту совокупность, называются запоминающими устройствами (ЗУ) того или иного типа.

Термин <<запоминающее устройство>> обычно используется, когда речь идет о принципе построения некоторого устройства памяти (например, полупроводниковое ЗУ, ЗУ на жестком магнитном диске и т.п.), а термин <<память>>*–* когда хотят подчеркнуть выполняемую устройством памяти логическую функцию или место расположения в составе оборудования ЭВМ (например, оперативная память *–* ОП, внешняя память и т.п.). В тех вопросах, где эти отличия не имеют принципиального значения, термины <<память>> и <<запоминающее устройство>> мы будем использовать как синонимы.

Запоминающие устройства играют важную роль в общей структуре ЭВМ. По некоторым оценкам производительность компьютера на разных классах задач на 40*–* 50% определяется характеристиками ЗУ различных типов, входящих в его состав.

К основным параметрам, характеризующим запоминающие устройства, относятся емкость и быстродействие.

Емкость памяти*–*это максимальное количество данных, которое в ней может храниться.

Емкость   запоминающего устройства измеряется количеством адресуемых элементов (ячеек) ЗУ и длиной ячейки в битах. В настоящее время практически все запоминающие устройства в качестве минимально адресуемого элемента используют 1 байт (1 байт = 8 двоичных разрядов (бит)). Поэтому емкость памяти обычно определяется в байтах, килобайтах (1Кбайт=210 байт), мегабайтах (1Мбайт = 220 байт), гигабайтах (1Гбайт = 230 байт) и т.д.

За одно обращение к запоминающему устройству производится считывание или запись некоторой единицы данных, называемой словом, различной для устройств разного типа. Это определяет разную организацию памяти. Например, память объемом 1 мегабайт может быть организована как 1Мслов по 1 байту, или 512Кслов по 2 байта каждое, или 256Кслов по 4 байта и т.д.

В то же время, в каждой ЭВМ используется свое понятие машинного слова, которое применяется при определении архитектуры компьютера, в частности при его программировании, и не зависит от размерности слова памяти, используемой для построения данной ЭВМ. Например, компьютеры с архитектурой IBM PC имеют машинное слово длиной 2 байта.

Быстродействие памяти определяется продолжительностью операции обращения, то есть временем, затрачиваемым на поиск нужной информации в памяти и на ее считывание, или временем на поиск места в памяти, предназначаемого для хранения данной информации, и на ее запись:

*t*обр = max(*t*обрсч*, t*обрзп);

где*t*обрсч*–*быстродействие   ЗУ при считывании информации; *t*обрзп*–* быстродействие   ЗУ при записи.

3.5.1. Классификация запоминающих устройств. Запоминающие устройства можно классифицировать по целому ряду параметров и признаков. На [рис.](http://www.intuit.ru/department/hardware/archhard2/5/#image.5.1)3.53 представлена классификация по типу обращения и организации доступа к ячейкам ЗУ.

Запоминающие устройства

По типу

обращения

По типу

обращения

С прямым(циклическим) доступом

С произвольным

доступом

С последовательным

доступом

На запись

и чтение

Только

чтение

Рис. 3.53. Классификация запоминающих устройств

По типу обращения ЗУ делятся на устройства, допускающие как чтение, так и запись информации, и постоянные запоминающие устройства (ПЗУ), предназначенные только для чтения записанных в них данных (ROM *–* readonlymemory). ЗУ первого типа используются в процессе работы процессора для хранения выполняемых программ, исходных данных, промежуточных и окончательных результатов. В ПЗУ, как правило, хранятся системные программы, необходимые для запуска компьютера в работу, а также константы. В некоторых ЭВМ, предназначенных, например, для работы в системах управления по одним и тем же неизменяемым алгоритмам, все программное обеспечение может храниться в ПЗУ.

В ЗУ с произвольным доступом (RAM *–* randomaccessmemory ) время доступа не зависит от места расположения участка памяти (например, ОЗУ).

В ЗУ с прямым (циклическим) доступом благодаря непрерывному вращению носителя информации (например, магнитный диск *–* МД) возможность обращения к некоторому участку носителя циклически повторяется. Время доступа здесь зависит от взаимного расположения этого участка и головок чтения/записи и во многом определяется скоростью вращения носителя.

В ЗУ с последовательным доступом производится последовательный просмотр участков носителя информации, пока нужный участок не займет некоторое нужное положение напротив головок чтения/записи (например, магнитные ленты *–* МЛ).

Как отмечалось выше, основные характеристики запоминающих устройств *–* это емкость и быстродействие. Идеальное запоминающее устройство должно обладать бесконечно большой емкостью и иметь бесконечно малое время обращения. На практике эти параметры находятся в противоречии друг другу: в рамках одного типа ЗУ улучшение одного из них ведет к ухудшению значения другого. К тому же следует иметь в виду и экономическую целесообразность построения запоминающего устройства с теми или иными характеристиками при данном уровне развития технологии. Поэтому в настоящее время запоминающие устройства компьютера, как это и предполагал Нейман, строятся по иерархическому принципу ([рис.](http://www.intuit.ru/department/hardware/archhard2/5/#image.5.2) 1.47).

*Vxy*

Внешняя память

Оперативная

память

РП

Внутренняя кэш-память

Внешняя кэш-память

*tобр*

Рис. 3.54.  Иерархическая организация памяти в современных ЭВМ

Иерархическая структура памяти позволяет экономически эффективно сочетать хранение больших объемов информации с быстрым доступом к информации в процессе ее обработки.

На нижнем уровне иерархии находится регистровая память *–* набор регистров, входящих непосредственно в состав микропроцессора (центрального процессора *–* CPU). Регистры CPU программно доступны и хранят информацию, наиболее часто используемую при выполнении программы: промежуточные результаты, составные части адресов, счетчики циклов и т.д. Регистровая память имеет относительно небольшой объем (до нескольких десятков машинных слов). РП работает на частоте процессора, поэтому время доступа к ней минимально. Например, при частоте работы процессора 2 ГГц время обращения к его регистрам составит всего 0,5 нс.

Оперативная память *–* устройство, которое служит для хранения информации (программ, исходных данных, промежуточных и конечных результатов обработки), непосредственно используемой в ходе выполнения программы в процессоре. В настоящее время объем ОП персональных компьютеров составляет несколько сотен мегабайт. Оперативная память работает на частоте системной шины и требует 6*–* 8 циклов синхронизации шины для обращения к ней. Так, при частоте работы системной шины 100 МГц (при этом период равен 10 нс) время обращения к оперативной памяти составит несколько десятков наносекунд.

Для заполнения пробела между РП и ОП по объему и времени обращения в настоящее время используется кэш-память, которая организована как более быстродействующая (и, следовательно, более дорогая) статическая оперативная память со специальным механизмом записи и считывания информации и предназначена для хранения информации, наиболее часто используемой при работе программы. Как правило, часть кэш-памяти располагается непосредственно на кристалле микропроцессора (внутренний кэш), а часть *–* вне его (внешняя кэш-память). Кэш-памятьпрограммно недоступна. Для обращения к ней используются аппаратные средства процессора и компьютера.

Внешняя память организуется, как правило, на магнитных и оптических дисках, магнитных лентах. Емкость дисковой памяти достигает десятков гигабайт при времени обращения менее 1 мкс. Магнитные ленты вследствие своего малого быстродействия и большой емкости используются в настоящее время в основном только как устройства резервного копирования данных, обращение к которым происходит редко, а может быть и никогда. Время обращения для них может достигать нескольких десятков секунд.

Следует отметить, что электронная вычислительная техника развивается чрезвычайно быстрыми темпами. Так, согласно эмпирическому <<закону Мура>>, производительность компьютера удваивается приблизительно каждые 18 месяцев. Поэтому все приводимые в данном издании количественные характеристики служат по большей части только для отражения основных соотношений и тенденций в развитии тех или иных компонентов и устройств компьютеров.

3.5.2. Построение ЗУ с заданной организацией.В современных ЭВМ минимальной адресуемой единицей памяти является, как правило, 1 байт. В связи с этим обмен с памятью организуется блоками, кратными этой величине: байтами, словами, двойными словами, учетверенными словами, в зависимости от выполняемой процессором команды и разрядности внешней шины данных. Такой обмен проходит под управлением специальных сигналов, поступающих по системной шине. Преобразование информации из формата ее представления на шине данных в формат, учитывающий организацию конкретных схем памяти, осуществляется специальными интерфейсными схемами. Большие интегральные схемы (БИС), на которых строятся модули памяти, являются изделиями электронной промышленности и могут иметь различную организацию. Разработчики средств вычислительной техники должны учитывать имеющуюся у них номенклатуру БИС памяти, чтобы построить запоминающее устройство необходимой емкости и организации. Для этой цели может проводиться объединение нескольких БИС либо с целью увеличения количества слов в модуле памяти, либо для наращивания разрядности каждого слова, либо с той и другой целью одновременно.

Рассмотрим варианты построения блока памяти необходимой организации при наличии заданных БИС памяти.

Построить ОЗУ с организацией 8К\*8 разрядов на БИС с организацией 1К\*8 разрядов ([рис.](http://www.intuit.ru/department/hardware/archhard2/5/2.html#image.5.3)3.55).

...

...

…

…

*А*

0

…

12

*DI*

0

…

7

*RD*

*WR*

*CS*

…

…

*А*

0

…

9

*DI*

0

…

7

*RD*

*WR*

*CS*

*MR*

M*R*

*DO*

◊

0

1

...

7

*DO*

◊

0

1

...

7

*а б*

Рис. 3.55.  Условно-графические обозначения запоминающих устройств с различной организацией:

*а–* 1К\*8 разрядов; *б–* 8К\*8 разрядов

В данном случае требуется построить модуль памяти, имеющий большее число слов, чем в составляющих его БИС. Модуль памяти будет состоять из восьми БИС. Для обращения к модулю памяти используется 13-разрядный адрес (А12*–* А0), поступающий по шине адреса (ША). Три старших разряда (А12*–* А10) определяют ту схему, которая в данный момент включается в работу, а каждая ячейка внутри любой БИС определяется 10-ю младшими разрядами адреса (А9*–*А0) ([рис.](http://www.intuit.ru/department/hardware/archhard2/5/2.html#image.5.4) 3.56).

|  |  |  |
| --- | --- | --- |
| Разряды адреса | | Выбранная БИС |
| 12 11 10  выбор БИС | 9…0  выбор ячейки в БИС |
| 1 1 1 | 1…1  …  0…0 | БИС 7 |
| 1 1 0 | 1…1  …  0…0 | БИС 6 |
| … | | |
| 0 0 1 | 1…1  …  0...0 | БИС 1 |
| 0 0 0 | 1…1  …  0…0 | БИС 0 |

Рис. 3.56.  Организация модуля памяти

При единичном значении сигнала на входе выбора кристалла БИС (CS=1) выходные разряды данных находятся в третьем состоянии, то есть как бы отключены от шины (DO=Z). Таким образом, при любом значении кода на шине адреса всегда в работе находится одна и только одна из восьми БИС ([рис.](http://www.intuit.ru/department/hardware/archhard2/5/2.html#image.5.5) 1.57).

Шина данных чтения

ША

Шина управления

MW MR

Шина данных записи

A MRDO

0 ◊

… 0

91

DI

0 …

1

…

7

RD

WR 7

CS0

0

0

1

9

…..

0

.

.

.

1

7

…..

7

0

A MRDO

0 ◊

… 0

91

DI

0 …

1

…

7

RD

WR 7

CS1

0

…..

1

9

0

.

.

.

1

7

……..

7

A MRDO

0 ◊

… 0

91

DI

0 …

1

…

7

RD

WR 7

CS7

0

0

9

1

…..

0

.

.

.

1

7

……..

7

A DC

0

01

1 ….

2

7

10

11

12

Рис. 1.57.  Запоминающее устройство объемом 8К\*8 разрядов на БИС с организацией 1К\*8 разрядов

В реальных микросхемах шины данных записи и чтения (DI и DO) обычно представляют собой общую двунаправленную шину.

Сигналы на шине управления означают: MW – сигнал записи в память, MR –сигнал чтения из памяти.

Построить ОЗУ с организацией 1К\*8 разрядов на БИС с организацией 1К\*1 разряд ([рис. 1.5](http://www.intuit.ru/department/hardware/archhard2/5/2.html#image.5.6)7).

…

A MRDO

0 ◊

…

9

DI

0 0

RD

WR

CS

Рис. 1.58.  Условно-графическое обозначение БИС с организацией 1К\*1 разряд

В данном случае требуется увеличить разрядность слова памяти. Так как все разряды одного слова должны записываться и считываться одновременно, то все БИС должны работать параллельно. Модуль памяти будет состоять из восьми БИС. Если разрабатываемый блок является частью модуля памяти, имеющего объем больше, чем 1К слов (например, 8К), то необходим специальный дешифратор, который будет дешифрировать старшие разряды адреса аналогично тому, как показано на [рис.](http://www.intuit.ru/department/hardware/archhard2/5/2.html#image.5.5) 1.59, и включать в работу данный блок.

1

…..

9

0

7

A MRDO

0 ◊

…

9

DI

0 0

RD

WR

CS0

0

A MRDO

0 ◊

…

9

DI

0 0

RD

WR

CS1

12

11

10

A DC

0

0 1

1i

2

7

1

…..

9

0

0

ША

Шина данных чтения

Шина данных записи

Шина управления

MW MR

A MRDO

0 ◊

…

9

DI

0 0

RD

WR

CS0

…..

0

9

Рис. 1.59.  Запоминающее устройство объемом 1К\*8 разрядов на БИС с организацией 1К\*1 разряд

* 1. Элементы теории цифровых автоматов
     1. **Основные определения.**

Цифровой автомат представляет собой объект, определяемый следующими характеристиками:

W, Z, δ**,**λ, ,

где A = {, a2,…, an} – множество состояний цифрового автомата;

W = { – множество выходных сигналов цифрового автомата;

Z = (z1, z2, ...,zn} – множество входных сигналов цифрового автомата;

δ - функция перехода цифрового автомата в новое состояние a(t + 1) в зависимости от его текущего состояния a(t) и действующего на его входе сигнала z(t): a(t + 1) = δ (a(t), z(t));

λ - функция выработки выходного сигнала ϖ(t + 1) в зависимости от те­кущего состояния a(t) цифрового автомата и действующего на его входе сигнала z(t):ϖ(t + 1) = λ (a(t), z(t));

∈A - начальное состояние цифрового автомата.

Если рассматривать множество входных сигналов Z как множество букв входного алфавита, а множество выходных сигналов W - как множество букв выходного алфавита, то цифровой автомат можно рассматривать как преобразователь слов, т. е. входному слову, состоящему из S букв входного алфавита, он ставит в соответствие выходное слово, состоящее из S букв вы­ходного алфавита.В этом случае эквивалентность двух цифровых автоматов можно определить следующим образом: два цифровых автомата являются эквивалентными, если они, имея одинаковые множества входных и выход­ных сигналов, являются одинаковыми преобразователями слов, т. е. при лю­бом начальном состоянии цифровые автоматы для любого входного слова формируют одинаковые выходные слова.

Цифровой автомат называется конечным, если используемые для его зада­ния множества конечны.

Цифровой автомат является полностью определенным, если каждой паре *a*(*t*),*z*(*t*) ставится в соответствие пара a(t + 1), ϖ(t + 1). В противном случае циф­ровой автомат называется частично определенным или просто частичным.

Практически все накапливающие узлы ЭВМ можно рассматривать как цифровые автоматы. Поэтому хорошо разработанная теория цифровых автома­тов находит широкое применение для синтеза и анализа сложных схем в вы­числительной технике.

Существуют два типа цифровых автоматов: автомат Мили и автомат Мура.

Автомат Мили определяют как автомат общего типа, т. е. его выходной сиг­нал и новое состояние являются функцией текущего состояния и действующего на входе сигнала.

Что же касается автомата Мура, то его выходной сигнал прямо не зави­сит от входного сигнала и определяется состоянием автомата, т. е. работа циф­рового автомата типа Мура задается в виде уравнений

а(t + 1) = δ(a(t), z(t));

ϖ(t + 1) = λ(а(t)).

Зависимость выходного сигнала от входного в автомате Мура проявляет­ся косвенно, т. е. через зависимость состояния от входного сигнала.

Цифровой автомат может быть задан с помощью таблиц или посредством графа.

**Задание цифрового автомата с помощью таблицы.** Автомат Милизадается с помощью двух таблиц:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *a*1 | *a*2 | *a*3 | *a*4 |
| *z*1 | *a*4 | *a*1 | *a*2 | *a*4 |
| *z*2 | *a*4 | *a*3 | *a*1 | *a*1 |
| *z*3 | *a*4 | *a*4 | *a*4 | *a*2 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *a*1 | *a*2 | *a*3 | *a*4 |
| *z*1 | *w*1 | *w*1 | *w*3 | *w*4 |
| *z*2 | *w*3 | *w*4 | *w*5 | *w*2 |
| *z*3 | *w*5 | *w*2 | *w*2 | *w*1 |

а б

Рисунок 3.60.

* таблицы переходов (рис. 3.60, а), определяющей правило формирования нового состояния на основании текущего состояния и действующего входного сигнала;
* таблицы выходов (рис. 3.60, б), определяющей правило формирования выходного сигнала на основании текущего состояния и действующего входного сигнала.

Приведенный цифровой автомат имеет множество входных сигналов Z = {z1, z2, z3}, множество состоянийА = {a1,a2, a3, a4} и множество выходных сигналов W = {ω1, ω2, ω3, ω4, ω5}.

Таблицы переходов и выходных сигналов имеют одинаковую размерность и одинаковую разметку столбцов и строк, что позволяет объединять их в одну таб­лицу. На рис. 1.26, а приведена объединенная таблица, соответствующая таблицам на рис. 3.60.

Таблицы переходов и выходов задают следующую дисциплину переходов и формирования выходного сигнала: если автомат находится в текущий момент в состоянии и получает по входу сигнал , то он переходит в новое состояние, которое записано в клетке таблицы переходов, расположенной на пересечении *j*-го столбца и *i*-й строки, и вырабатывает выходной сигнал, который определен значением в этой клетке.

Например, если автомат находится в состоянии , а на вход поступает сиг­нал z2, то он переходит в состояние a1 (см. рис. 3.60, а) и вырабатывает выходной сигнал ω2 (см. рис. 3.60, б).

Работа данного автомата как преобразователя слов иллюстрируется сле­дующим образом.

Предположим, что на вход автомата, находящегося в начальном состоянии aн = *a*3, поступает входное слово в виде последовательности букв *z*1, *z*1, *z*3, *z*2, *z*2, *z*1, тогда цифровой автомат будет переходить в последовательность состояний (см. рис. 1.25, а) и вырабатывать последовательность выходных сигналов (см. рис. 1.25, б), имеющих вид

a(t): a3, a2, a1, a4, a1, a4;

z(t):

*a*(t + 1): a2, a1, a4, a1, a4, a4;

ω(t + 1):

В приведенных последовательностях приняты следующие обозначения:

* *a*(*t*) – текущее состояние цифрового автомата;
* *z*(*t*) – входной сигнал, действующий на текущем такте работы цифрового автомата;
* a(t + 1), ω(t + 1) - новое состояние цифрового автомата и вырабатываемая им буква выходного алфавита соответственно.

Отработка входного слова из шести букв осуществляется за шесть тактов входного автомата. Новое состояние на такте является одновременно текущим со­стоянием на следующем такте.

Из приведенных последовательностей видно, что в ответ на поступившее входное слово z1, z1, z3, z2, z2, z1, цифровой автомат, находившийся в исходном со­стоянии a3, выдает выходное слово ω3, ωi, ω5, ω2, ω3, ω4, содержащее столько же букв, что и входное слово, и в конце преобразования оказывается в состоянии a4. Очевидно, что при другом исходном состоянии, отличным от , цифровой авто­мат выработает выходное слово, отличное от ω3, ωi, ω5, ω2, ω3, ω4. В этом можно убедиться, взяв другое начального состояние, и для того же входного слова найти выходное слово, которое сформирует цифровой автомат.

Автомат Мура задается с помощью одной таблицы, в которой опреде­ляются правило формирования нового состояния на основании текущего со­стояния и действующего входного сигнала и связь выходного сигнала с состоя­нием цифрового автомата. На рис. 1 .26, б приведен пример табличного задания автомата Мура.

Таблица переходов, так же как и в случае автомата Мили, задает следую­щую дисциплину переходов: если автомат находится в текущий момент в со­стоянии и получает по входу сигнал zi, то он переходит в новое состояние, которое записано в клетке таблицы переходов, расположенной на пересечении j-го столбца и *i*-й строки, и вырабатывает выходной сигнал, который определя­ется новым состоянием цифрового автомата. На рис. 1 .26, б в самой верхнейстроке таблицы перечислены выходные сигналы в их взаимосвязи с состояния­ми, перечисленными во второй строке. Например, если автомат находится в со­стоянии *a*4, а на вход поступает сигнал *z*2, то он переходит в состояние *a*3, кото­рому соответствует выходной сигнал ϖ1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *a*1 | *a*2 | *a*3 | *a*4 |
| *z*1 | *a*3  *w*1 | *a*1  *w*1 | *a*2  *w*3 | *a*4  *w*4 |
| *z*2 | *a*4  *w*3 | *a*3  *w*4 | *a*1  *w*5 | *a*1  *w*2 |
| *z*3 | *a*4  *w*5 | *a*4  *w*2 | *a*4  *w*2 | *a*4  *w*1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *w*2 | *w*3 | *w*1 | *w*1 |
|  | *a*1 | *a*2 | *a*3 | *a*4 |
| *z*1 | *a*2 | *a*1 | *a*1 | *a*4 |
| *z*2 | *a*3 | *a*4 | *a*1 | *a*3 |
| *z*3 | *a*4 | *a*3 | *a*4 | *a*2 |

Рисунок 3.61

А- автомат Мили; б – автомат Мура

Работа данного автомата как преобразователя слов иллюстрируется сле­дующим образом.

Предположим, что на вход автомата, находящегося в начальном состоянии aн = , поступает входное слово в виде последовательности букв z1, z3, z3, z2, z3, z1, тогда цифровой автомат будет переходить в последовательность состояний и выра­батывать последовательность выходных сигналов, имеющих вид (см. рис. 3.61, б):

a(t): a2, a1, a4, a2, a4, a2;

z(t):

*a*(t + 1): a1, a4, a2, a4, a2, a1;

ω(t + 1):

В приведенных последовательностях приняты следующие обозначения:

a(t) – текущее состояние цифрового автомата;

z(t) – входной сигнал, действующий на текущем такте работы цифрового автомата;

a(t + 1), ω(t + 1) – новое состояние цифрового автомата и вырабатываемый выходной сигнал соответственно.

Отработка входного слова из шести букв осуществляется за шесть тактов работы входного автомата. Новое состояние на текущем такте является одновре­менно текущим состоянием на следующем такте.

Из приведенных последовательностей видно, что в ответ на поступившее входное слово z1, z3, z3, z2, z3, z1 цифровой автомат, находившийся в исходном со­стоянии a2, выдает выходное слово ω2, ω1, ω3, ω1, ω3, ω2, содержащее столько же букв, что и входное слово, и в конце преобразования оказывается в состоянии a1. Очевидно, что при другом исходном состоянии цифровой автомат в общем случае выработает выходное слово, отличное от ω2, ω1, ω3, ω1, ω3, ω2.

Задание цифрового автомата с помощью графа

Цифровой автомат может быть задан графом, количество вершин которого равно количеству состояний цифрового автомата. Вершины графа соединены дуга­ми, указывающими возможный переход из одного состояния в другое, помеченны­ми входными сигналами, при которых имеет место представляемый дугой переход.

В автоматах Мили дуги также помечаются выходными сигналами, которые выра­батываются цифровым автоматом при переходе. В автоматах Мура выходными сигналами помечаются вершины графа. На рис. 3.62 представлен граф, соответст­вующий автомату Мили, заданному таблицей (см. рис. 3.61, а), а на рис. 3.63 приве­ден граф автомата Мура, заданного таблицей (см. рис. 3.61, б).

*a*1

*a*2

*a*4

*a*3

*z*1*w*1

*z*3*w*2

*z*2*w*4

*z*1*w*3

*z*3*w*2

*z*1*w*4

*z*1*w*1

*z*3*w*2

*z*2*w*5

*z*2*w*2

*z*3*w*5 +

+ *z*2*w*3

Рисунок 3.62

Стрелки на дугах показывают, в какое состояние осуществляется переход. В случае если из одной вершины в другую имеют место переходы по нескольким входным сигналам, то такой переход обозначается одной дугой, на которой через знак логической суммы перечисляются все входные сигналы.

На графе автомата Мили переход из вершины а1 в вершину а4 имеет место при входном сигнале z3 с выработкой выходного сигнала ш5 и при входном сигна­ле z2 с выработкой выходного сигналаϖ3. Поэтому стрелка перехода из вершины a1 в вершину а4 помечена логическим условием z3 ω5 + z2 ω3.

* + 1. **Переход от одной формы задания автомата к другой.**

Любой цифровой автомат можно представить в виде автомата Мура или в виде автомата Мили.

Переход от автомата Мура к автомату Мили

Процедуру рассматриваемого перехода иллюстрирует следующий пример.

Имеется цифровой автомат Мура, заданный в виде таблицы, приведенной на рис. 3.63, а.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *w*1 | *w*2 | *w*2 | *w*4 | *w*3 |
|  | *a*1 | *a*2 | *a*3 | *a*4 | *a*5 |
| *z*1 | *a*3 | *a*3 | *a*2 | *a*2 | *a*5 |
| *z*2 | *a*5 | *a*5 | *a*5 | *a*5 | *a*4 |
| *z*3 | *a*1 | *a*1 | *a*4 | *a*4 | *a*2 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | *c*1 | *c*2 | *c*3 |
| *z*1 | *c*2  *w*2 | *c*1  *w*2 | *c*3  *w*2 |
| *z*2 | *c*3  *w*2 | *c*3  *w*2 | *c*2  *w*2 |
| *z*3 | *c*1  *w*2 | *c*2  *w*2 | *c*1  *w*2 |

а б

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *w*2 | *w*2 | *w*3 | *w*4 | *w*1 |
|  | *b*1 | *b*2 | *b*3 | *b*4 | *b*5 |
| *z*1 | *b*2 | *b*1 | *b*3 | *b*2 | *b*1 |
| *z*2 | *b*3 | *b*3 | *b*4 | *b*3 | *b*3 |
| *z*3 | *b*4 | *b*5 | *b*2 | *b*4 | *b*5 |

в

Рисунок 3.63

*a*1

*a*2

*a*4

*a*3

*z*1

*z*1

*z*2

*w*1

*z*3

*z*1

*z*2

*z*1 + *z*2

*z*3

*w*2

*z*3

*z*2

*z*2

*w*1

*w*3

Найти задание автомата Мили, эквивалентного заданному автомату Мура.

Решение

Из множества состояний автомата Мура составим подмножества, каждое из которых включает состояния, имеющие одинаковые переходы. Каждому по­лученному подмножеству поставим в соответствие состояние Ci искомого ав­томата Мили:

{A1, А2} – C1;

{*А*3, *А*4} – *C*2;

{*A*5} – *С*3.

Таким образом, искомый автомат Мили будет иметь три состояния, что по­зволяет разметить столбцы и строки таблицы задания искомого автомата Мили. Таблица имеет вид, приведенный на рис. 1.29, б.

Заполнение клеток таблицы выполняется следующим образом.

Для того чтобы найти переход из *С*1 при поступлении *Z*1, рассмотрим перехо­ды, которые имеют место для любого из состояний автомата Мура, объединенных в подмножество, обозначенное *С*1. Из элементов {*А*1, *А*2} возьмем, например, *А*2.

В таблице автомата Мура для этого состояния при входном сигнале *Z*1имеет место переход в состояние *А*3, которое входит в подмножество, обозна­ченное как С2. Кроме того, состоянию *А*3, как видно из той же таблицы, соответ­ствует выходной сигнал ϖ2. Из этого следует, что в клетку, соответствующую пе­реходу из С1 по входному сигналу *Z*1, нужно поставить С2 и указать его в качестве вырабатываемого выходного сигнала ϖ2.

Рассмотрим заполнение клетки перехода из состояния С3 при входном сиг­нале Z2. Из подмножества {А5,}, соответствующего С3, берем состояние автомата МураА5. В таблице автомата Мура для этого состояния при входном сигнале Z2имеет место переход в состояние А4, которое входит в подмножество, обозначен­ное как С2. Кроме того, состоянию А4, как видно из той же таблице, соответствует выходной сигнал ϖ4. Из этого следует, что в клетку, соответствующую переходу из *С*3 по входному сигналу *Z*2, нужно поставить *С*2 и указать его в качестве выра­батываемого выходного сигнала v4. Аналогичным образом заполняются все клет­ки приведенной таблицы.

Переход от автомата Мили к автомату Мура

Процедуру данного перехода иллюстрирует следующий пример.

Имеется цифровой автомат Мили, заданный в виде таблицы (см. рис. 1.29, б). Найти задание автомата Мили, эквивалентного заданному автомату Мура.

Решение

Составим множество неповторяющихся пар Сiϖj. Каждому элементу этого множества поставим в соответствие одно из состояний искомого автомата Мура:

C2 ω2 – B1;

C1 ω2 – B2;

C3ω3 –B3;

C2ω4 –B4;

C1ω1 –B5.

Для пяти состояний и трех входов таблица задания искомого автомата Мура будет иметь вид, приведенный на рис. 1.29, в. В столбцах таблицы отмечено пять найденных состояний автомата и выходные сигналы (см. пары Сi ωj ).

Заполнение клеток переходов таблицы осуществлено следующим образом.

Переход из В1 по входному сигналу *Z*1 будет соответствовать переходу из *С*2по сигналу *Z*1. Из таблицы на рис. 1.29, б, видно, что при текущем состоянии *С*2 по сигналу Z1 осуществляется переход в состояние С1и вырабатывается выходной сигнал ω2, что соответствует состоянию *В*2. Поэтому в клетку, указывающую на переход из *В*1 по входному сигналу *Z*1, записывается *В*2.

Аналогично заполняются все остальные клетки таблицы.

Полученный автомат Мили должен быть эквивалентен автомату Мили, за­данному в таблице на рис. 1.29, б, и автомату Мура, заданному в таблице на рис. 1.29, а. Для проверки эквивалентности полученных автоматов возьмем в каче­стве начального состояния для автомата Мура, заданного в таблице на рис. 1.29, а, *А*н = *А*з, а в качестве входного слова –*Z*2*Z*1*Z*1*Z*3*Z*2.

Основываясь на таблице, приведенной на рис. 1.29, б, составим для выбран­ного входного слова последовательность состояний и формируемых выходных сигналов. Эта последовательность будет иметь вид:

*A*(*t*)– *A*3 *A*5 *A*5*A*2;

X(t) – Z2 Z1Z1 Z3 Z2;

A(t + 1) – *A*5*A*5*A*5*A*2*A*5;

ω(t + 1) – ω3 ω3ω3ω2ω3.

Таким образом, рассмотренный автомат на заданное входное слово вырабо­тал выходное слово ω3ω3ω3 ω2 ω3 и оказался в состоянии *A*5. Найдем реакцию на то же входное слово автомата Мили, представленного в таблице на рис. 1.29, б.

На основании данной таблицы составим для заданного входного слова по­следовательность состояний и формируемых выходных сигналов. В качестве ис­ходного состояния возьмем С2, так как оно отражает подмножество состояний {*A*3, *A*3}, в которое входит начальное состояние ранее рассмотренного цифрового автомата Мура.

Эта последовательность будет иметь вид:

*C*(*t*) – *С*2*С*3*С*з*С*з*С*1;

X(*t*) – *Z*2*Z*1*Z*1*Z*3*Z*2;

C(t + 1) – *С*з*С*з*С*з*С*1*С*з;

ω(*t* + 1) – ωз ω3ω3 ω2ωз.

Таким образом, рассмотренный автомат на заданное входное слово вырабо­тал выходное слово ω3ω3ω3 ω2 ω3 и оказался в состоянии *С*1.

На основании таблицы автомата Мура, приведенной на рис. 1.29, в, соста­вим для заданного входного слова последовательность состояний и формируемых выходных сигналов. В качестве исходного состояния используем В1, которое от­ражает пару *С*2 ω2, где *С*2 используется в качестве исходного состояния ранее рас­смотренного цифрового автомата Мили (с таким же успехом можно взять в каче­стве начального состояние *B*4).

Эта последовательность будет иметь вид:

B(t) – *B*1*B*3*B*3*B*3*B*2;

X(t) – *Z*2*Z*1*Z*1*Z*3*Z*2;

C(t + 1) – *B*3*B*3*B*3*B*2*B*3;

ω(t + 1) – ω3ω3ω3 ω2 ω3.

Таким образом, рассмотренный автомат на заданное входное слово вырабо­тал выходное слово ω3ω3ω3ω2ω3 и оказался в состоянии *С*2. Следовательно, все три рассмотренных автомата при соответствующих начальных состояниях одина­ково преобразовали входное слово *Z*2*Z*1*Z*1*Z*3*Z*2 в выходное слово ω3ω3ω3ω2ω3. Отметим, что после преобразования заданного входного слова все три автомата перешли в состояния A5, С3, B3 соответственно.

1. **Синтез цифрового автомата.**С точки зрения синтеза цифровой автомат удобно представить в виде струк­турной схемы, приведенной на рис. 1.30.

Входной сигнал

Сигнал о состоянии

Память цифрового автомата

2

Логика цифрового автомата

1

Управление памятью

Выходной сигнал

Рис. 3.64

Цифровой автомат в общем случае состоит из памяти и логической части. Память хранит информацию о предыстории цифрового автомата, вырабатывая сигнал, характеризующий текущее состояние автомата. Логическая часть на осно­вании входного и поступающего из памяти сигналов вырабатывает выходной сиг­нал. Кроме того, используя входной сигнал и сигнал состояния, логическая часть вырабатывает сигнал управления памятью, обеспечивающий переход из текущего в новое состояние.

Память реализуется на триггерах, количество которых должно позво­лять запоминать любое состояние из множества состояний цифрового авто­мата, а сигналы ее управления должны соответствовать оговоренным при за­дании цифрового автомата переходам и типу используемого триггера для по­строения памяти.

Построение логической части предполагает представление входных и вы­ходных сигналов цифрового автомата на языке алгебры логики, т. е. в виде логи­ческих переменных и функций.

Учитывая вышеизложенное, можно выполнить синтез цифрового автомата следующим образом.

1. Кодирование входных сигналов в виде набора логических переменных.
2. Кодирование выходных сигналов в виде набора логических функций.
3. Кодирование состояний цифрового автомата.
4. Формирование кодированной таблицы переходов и выходов.
5. Выбор типа запоминающего элемента.
6. Составление логических выражений для логических функций, использо­ванных для кодировки выходных сигналов.
7. Составление логических выражений для сигналов управления памятью.
8. Синтез логических схем для сформированных логических выражений.
9. Формирование выходных сигналов цифрового автомата на основании ко­дирующих их функций.

Рассмотрим реализацию перечисленных действий на конкретном примере.

Пример

Синтезировать цифровой автомат, заданный в виде таблиц, приведенных на рис. 1.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | А1 | А2 | А3 | А4 |
| z1 | А3 | А3 | А4 | - |
| z2 | А3 | - | А2 | А1 |
| z3 | А2 | А1 | А1 | А3 |

а

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | А1 | А2 | А3 | А4 |
| z1 | Ѡ3 | Ѡ2 | Ѡ1 | - |
| z2 | Ѡ1 | - | Ѡ3 | Ѡ2 |
| z3 | Ѡ4 | Ѡ1 | Ѡ4 | Ѡ3 |

б

Память автомата построить на *T*-триггере.

При синтезе логических выражений использовать базис И, ИЛИ, НЕ.

Решение

Кодирование состояний выполним через набор логических переменных *Q*. Множество состояний включает четыре элемента. Поэтому для представления каждой из них достаточно использовать комбинации из двух переменных *Q*1 и Q2. Кодировка состояний представлена таблицей на рис. 1.32, а.

Кодирование входных сигналов выполним через набор логических пере­менных х. Множество входных сигналов включает три элемента. Поэтому для представления каждого из них достаточно использовать комбинации из двух пе­ременных х1 и *х*2. Кодировка входных переменных представлена таблицей на рис. 1.32, б.

Кодирование выходных сигналов выполним через набор логических функций *у.*Множество выходных сигналов включает четыре элемента. Поэтому для представ­ления каждой из них достаточно использовать комбинации из двух переменных *у*1 и *у*2. Кодировка выходных переменных представлена таблицей на рис. 1.32, в.

|  |  |  |
| --- | --- | --- |
|  | Q1 | Q2 |
| A1 | 0 | 1 |
| A2 | 1 | 0 |
| A3 | 1 | 1 |
| A4 | 0 | 0 |

а

|  |  |  |
| --- | --- | --- |
|  | x1 | x2 |
| z1 | 0 | 1 |
| z2 | 1 | 0 |
| z3 | 1 | 1 |
|  | | |

б

|  |  |  |
| --- | --- | --- |
|  | y1 | y2 |
| Ѡ1 | 0 | 1 |
| Ѡ2 | 1 | 0 |
| Ѡ3 | 1 | 1 |
| Ѡ4 | 0 | 0 |

Таблицы переходов (см. рис. 1.31, а) и выходных сигналов (см. рис. 1.31, б) после замены переменных исходного задания цифрового автомата на их кодиро­ванные значения будут иметь вид, представленный на рис. 1.33, а и рис. 1.33, б со­ответственно.

В таблице переходов на рис. 1.33, а клетки заполнены двухразрядным ко­дом, первый разряд которого отображает значение переменной Q1, а второй - переменной *Q*2 (1 - переменная имеет прямое значение, 0 - переменная имеет обрат­ное значение).

В таблице выходов на рис. 1.33, б клетки заполнены двухразрядным кодом, первый разряд которого отображает значение переменной у1, а второй - перемен­ной *у*2(1 - переменная имеет прямое значение, 0 - переменная имеет обратное значение).

Логические выражения для разрядов кода выходных сигналов составляются на основе таблицы на рис. 1.33, б. Эти выражения имеют следующий вид:

Приведенные выражения составляются следующим образом.

При формировании *у*1 в кодированной таблице выходных сигналов на рис. 1.33, б выбираются все случаи, когда *у*1 имеет единичное значение, и для ка­ждого из них формируется конъюнкция, отражающая начальное состояние, и входные сигналы, при которых функция *у*1 имеет единичное значение.

Например, первая конъюнкция в выражении для *y*1 имеет вид 1Q21x2 и соответствует слу­чаю, когда начальное состояние имеет значения 1*Q*2, а на вход поступает сигнал 1*x*2. В этой клетке переменная *у*1 (первый разряд двухразрядного кода выходного сигнала) имеет единичное значение.

Пятая конъюнкция выражения для *у*1 имеет вид 1 2x1x2 и соответствует

клетке, расположенной на пересечении столбца 12 и строки *x*1*x*2, где функ­ция *у*1, согласно таблице на рис. 1.33, б, имеет единичное значение.

Выражение для у2 формируется аналогично, но рассматриваются значения второго разряда двухразрядного кода, соответствующего выходному сигналу цифрового автомата.

Логические выражения для функций управления памятью составляются на основе таблицы на рис. 1.33, а с учетом особенностей используемого эле­мента памяти. В рассматриваемом случае элементом памяти является *T*-триггер, основная особенность которого заключается в том, что по каждому входному сигналу триггер изменяет свое состояние на противоположное. По­этому сигнал на вход *T*-триггера надо подавать только в тех случаях, когда но­вое его состояние отличается от текущего. Логические выражения для сигналов управления памятью в рассматриваемом случае имеют вид

Приведенные выражения составляются следующим образом.

В выражении для функции сигнала T-входа первого разряда qT1 в общей дизъюнктивной форме используются конъюнкции, отражающие все случаи, ко­гда значение первого разряда кода нового состояния противоположно его на­чальному состоянию. Например, первая конъюнкция в выражении для qT1имеет вид 1*Q*2 1 *x*2 и соответствует случаю, когда начальное состояние имеет значе­ния 1*Q*2 (01), и на вход поступает сигнал 1*x*2, а код нового состояния *Q*1*Q*2имеет в первом разряде значение, отличное от значения первого разряда кода начального состояния.

Пятая конъюнкция выражения для qT1 имеет вид *Q*1*Q*21*x*2 и соответствует клетке, расположенной на пересечении столбца *Q*1*Q*2 и строки 1*x*2 где первый разряд кода нового состояния автомата, согласно таблице на рис. 1.33, а, имеет значение 1, противоположное значению, которое имеет место в первом разряде кода начального состояния цифрового автомата.

Выражение для qT2 формируется аналогично, но рассматриваются измене­ния значения второго разряда двухразрядного кода состояния после учета воздей­ствия входного сигнала.

Для более компактного представления полученных логических выраже­ний и обозначений на формируемой схеме цифрового автомата введем деся­тичную кодировку конъюнкций, используемых в полученных логических вы­ражениях. Каждая конъюнкция представляет набор одних и тех же переменных *Q*1, *Q*2, *x*1, *x*2, поэтому эти конъюнкций можно рассматривать как четырехраз­рядный двоичный код и кодировать десятичными эквивалентами этого двоич­ного кода. Следовательно, ранее полученные выражения можно представить в следующей компактной форме:

&

9

&

14

&

2

&

3

&

6

&

11

&

13

&

15

&

7

&

1

2

3

4

с

СИ

С

1

2

&

5

1

1

1

1

T

C

TT

1

T

C

TT

2

1

2

C

DС

1

2

3

4

&

*z*1

*z*2

*z*3

*x*1

*x*2

*Q*1

*Q*2

*q*T1

*q*T2

*q*T2

*q*T1

*y*2

*y*1

СИ

*w*1

*w*2

*w*3

*w*4

y1 = 5 + 9 + 14 + 2 + 3; у2 = 5 + 6 + 11 + 13 + 14 + 3;

*qT*1 = 5 + 6 + 7 + 11 + 13 + 15 + 3; *qT*2 = 7 + 9 + 11 + 13 + 14 + 2 + 3.

Таким образом, множество неповторяющихся конъюнкций в кодированной форме, которые используются во всех сформированных логических выражениях, имеет вид {5, 9, 14, 2, 3, 6, 11, 13, 7,15}.

Логическая схема, реализующая сформированные логические выражения для выходных сигналов и сигналов управления памятью цифрового автомата и осуществляющая кодировку входных и декодировку выходных сигналов, имеет вид (рис. 1.34)

Память цифрового автомата реализована на двух *T*-триггерах, формирую­щих парафазные выходные сигналы Q*i*, *i*, используемые для кодировки состоя­ний A1,*A*2, A3, A4. Кодер формирует сигналы *x*1,1; *x*2, 2, кодирующие входные сигналы *z*1, *z*2, *z*3. Схемы «И» обозначены десятичными числами, соответствую­щими кодам конъюнкций, которые они формируют. Выходы логических схем «ИЛИ» обозначены логическими функциями, формируемыми этими логическими схемами. Выходные сигналы цифрового автомата формируются с помо­щью декодера.

1. АРХИТЕКТУРА МИКРОПРОЦЕССОРНЫХ СИСТЕМ

4.1. Организация компьютерных систем

4.1.1. Основные понятия

**Электронная система**– в данном случае это любой электронный узел, блок, прибор или комплекс, производящий обработку информации.

**Задача**–это набор функций, выполнение которых требуется от электронной системы.

**Быстродействие**– это показатель скорости выполнения электронной системой ее функций.

**Гибкость**– это способность системы подстраиваться под различные задачи.

**Избыточность**– это показатель степени соответствия возможностей системы решаемой данной системой задаче.

**Интерфейс**– соглашение об обмене информацией, правила обмена информацией, подразумевающие электрическую, логическую и конструктивную совместимость устройств, участвующих в обмене. Другое название –**сопряжение**.

**Микропроцессорная система** может рассматриваться как частный случай электронной системы, предназначенной для обработки входных сигналов и выдачи выходных сигналов ([рис. 4.1](http://www.intuit.ru/department/hardware/mpbasics/1/#image.1.1)). В качестве входных и выходных сигналов при этом могут использоваться аналоговые сигналы, одиночные цифровые сигналы, цифровые коды, последовательности цифровых кодов. Внутри системы может производиться хранение, накопление сигналов (или информации), но суть от этого не меняется. Если система цифровая (а *микропроцессорные системы* относятся к разряду цифровых), то входные аналоговые сигналы преобразуются в последовательности кодов выборок с помощью *АЦП*, а выходные аналоговые сигналы формируются из последовательности кодов выборок с помощью *ЦАП*. Обработка и хранение информации производятся в цифровом виде.

Характерная особенность традиционной цифровой системы состоит в том, что алгоритмы обработки и хранения информации в ней *жестко связаны* со схемотехникой системы. То есть изменение этих алгоритмов возможно только путем изменения структуры системы, замены электронных узлов, входящих в систему, и/или связей между ними. Например, если нам нужна дополнительная операция суммирования, то необходимо добавить в структуру системы лишний *сумматор*. Или если нужна дополнительная функция хранения кода в течение одного такта, то мы должны добавить в структуру еще один регистр. Естественно, это практически невозможно сделать в процессе эксплуатации, обязательно нужен новый производственный цикл проектирования, изготовления, отладки всей системы. Именно поэтому традиционная цифровая система часто называется системой на «жесткой логике».

Аналоговые

сигналы

Цифровые коды

Цифровые

сигналы

Электронная система (обработка, хранение)

Аналоговые

сигналы

Цифровые

сигналы

Цифровые коды

Рис. 4.1.  Электронная система.

Любая система на «жесткой логике» обязательно представляет собой специализированную систему, настроенную исключительно на одну задачу или (реже) на несколько близких, заранее известных задач. Это имеет свои бесспорные преимущества.

Во-первых, специализированная система (в отличие от универсальной) никогда не имеет аппаратурной избыточности, то есть каждый ее элемент обязательно работает в полную силу (конечно, если эта система грамотно спроектирована).

Во-вторых, именно специализированная система может обеспечить максимально высокое быстродействие, так как скорость выполнения алгоритмов обработки информации определяется в ней только быстродействием отдельных логических элементов и выбранной схемой путей прохождения информации. А именно логические элементы всегда обладают максимальным на данный момент быстродействием.

Но в то же время большим недостатком цифровой системы на "жесткой логике" является то, что для каждой новой задачи ее надо проектировать и изготавливать заново. Это процесс длительный, дорогостоящий, требующий высокой квалификации исполнителей. А если решаемая задача вдруг изменяется, то вся аппаратура должна быть полностью заменена. В нашем быстро меняющемся мире это довольно расточительно.

Путь преодоления этого недостатка довольно очевиден: надо построить такую систему, которая могла бы легко адаптироваться под любую задачу, перестраиваться с одного алгоритма работы на другой без изменения аппаратуры. И задавать тот или иной алгоритм мы тогда могли бы путем ввода в систему некой дополнительной управляющей информации, **программы** работы системы ([рис. 4.2](http://www.intuit.ru/department/hardware/mpbasics/1/#image.1.2)). Тогда система станет универсальной, или **программируемой**, не жесткой, а гибкой. Именно это и обеспечивает *микропроцессорная система*.

Аналоговые

сигналы

Цифровые коды

Цифровые

сигналы

Аналоговые

сигналы

Цифровые

сигналы

Цифровые коды

Управляющая

информация

(программа)

Рис. 4.2.  Программируемая (она же универсальная) электронная система

Но любая универсальность обязательно приводит к избыточности. Ведь решение максимально трудной задачи требует гораздо больше средств, чем решение максимально простой задачи. Поэтому сложность универсальной системы должна быть такой, чтобы обеспечивать решение самой трудной задачи, а при решении простой задачи система будет работать далеко не в полную силу, будет использовать не все свои ресурсы. И чем проще решаемая задача, тем больше избыточность, и тем менее оправданной становится универсальность. Избыточность ведет к увеличению стоимости системы, снижению ее надежности, увеличению потребляемой мощности и т.д.

Кроме того, универсальность, как правило, приводит к существенному снижению быстродействия. Оптимизировать универсальную систему так, чтобы каждая новая задача решалась максимально быстро, попросту невозможно. Общее правило таково: чем больше универсальность, гибкость, тем меньше быстродействие. Более того, для универсальных систем не существует таких задач (пусть даже и самых простых), которые бы они решали с максимально возможным быстродействием. За все приходится платить.

Таким образом, можно сделать следующий вывод. Системы на «жесткой логике» хороши там, где решаемая задача не меняется длительное время, где требуется самое высокое быстродействие, где алгоритмы обработки информации предельно просты. А универсальные, программируемые системы хороши там, где часто меняются решаемые задачи, где высокое быстродействие не слишком важно, где алгоритмы обработки информации сложные. То есть любая система хороша на своем месте.

Однако за последние десятилетия быстродействие универсальных (*микропроцессорных*) систем сильно выросло (на несколько порядков). К тому же большой объем выпуска микросхем для этих систем привел к резкому снижению их стоимости. В результате область применения систем на «жесткой логике» резко сузилась. Более того, высокими темпами развиваются сейчас программируемые системы, предназначенные для решения одной задачи или нескольких близких задач. Они удачно совмещают в себе как достоинства систем на «жесткой логике», так и программируемых систем, обеспечивая сочетание достаточно высокого быстродействия и необходимой гибкости. Так что *вытеснение* «жесткой логики» продолжается.

4.1.2. Что такое микропроцессор

Ядром любой *микропроцессорной системы* является *микропроцессор* или просто процессор (от англ.processor). Перевести на русский язык это слово правильнее всего как "обработчик", так как именно ***микропроцессор***– это тот узел, блок, который производит всю обработку информации внутри *микропроцессорной системы*. Остальные узлы выполняют всего лишь *вспомогательные функции*: хранение информации (в том числе и управляющей информации, то есть программы), связи с внешними устройствами, связи с пользователем и т.д. Процессор заменяет практически всю «жесткую логику», которая понадобилась бы в случае традиционной цифровой системы. Он выполняет *арифметические функции* (сложение, умножение и т.д.), логические функции (сдвиг, сравнение, маскирование кодов и т.д.), временное хранение кодов (во внутренних регистрах), пересылку кодов между узлами *микропроцессорной системы* и многое другое. Количество таких элементарных операций, выполняемых процессором, может достигать нескольких сотен. Процессор можно сравнить с мозгом системы.

Но при этом надо учитывать, что все свои операции процессор выполняет **последовательно**, то есть одну за другой, по очереди. Конечно, существуют процессоры с параллельным выполнением некоторых операций, встречаются также *микропроцессорные системы*, в которых несколько процессоров работают над одной задачей параллельно, но это редкие исключения. С одной стороны, последовательное выполнение операций – несомненное достоинство, так как позволяет с помощью всего одного процессора выполнять любые, самые сложные алгоритмы обработки информации. Но, с другой стороны, последовательное выполнение операций приводит к тому, что время выполнения алгоритма зависит от его сложности. Простые алгоритмы выполняются быстрее сложных. То есть *микропроцессорная система* способна сделать все, но работает

Процессор

Управляющая

информационная

(программа)

Рис. 4.3.  Информационные потоки в микропроцессорной системе

Итак, *микропроцессор* способен выполнять множество операций. Но откуда он узнает, какую операцию ему надо выполнять в данный момент? Именно это определяется управляющей информацией, программой. Программа представляет собой набор **команд (инструкций)**, то есть цифровых кодов, расшифровав которые, процессор узнает, что ему надо делать. Программа от начала и до конца составляется человеком, программистом, а процессор выступает в роли послушного исполнителя этой программы, никакой инициативы он не проявляет (если, конечно, исправен). Поэтому сравнение процессора с мозгом не слишком корректно. Он всего лишь исполнитель того алгоритма, который заранее составил для него человек. Любое отклонение от этого алгоритма может быть вызвано только неисправностью процессора или каких-нибудь других узлов *микропроцессорной системы*.

Все команды, выполняемые процессором, образуют **систему команд** процессора. Структура и объем системы команд процессора определяют его быстродействие, гибкость, *удобство использования*. Всего команд у процессора может быть от нескольких десятков до нескольких сотен. Система команд может быть рассчитана на узкий круг решаемых задач (у специализированных процессоров) или на максимально широкий круг задач (у универсальных процессоров). Коды команд могут иметь различное количество разрядов (занимать от одного до нескольких байт). Каждая команда имеет свое время выполнения, поэтому время выполнения всей программы зависит не только от количества команд в программе, но и от того, какие именно команды используются.

По числу больших интегральных схем (БИС) в микропроцессорном комплекте различают микропроцессоры однокристальные, многокристальные и многокристальные секционные.

Процессоры даже самых простых ЭВМ имеют сложную функциональную структуру, содержат большое количество электронных элементов и множество разветвленных связей. Изменять структуру процессора необходимо так, чтобы полная принципиальная схема или ее части имели количество элементов и связей, совместимое с возможностями БИС. При этом микропроцессоры приобретают внутреннюю магистральную архитектуру, т. е. в них к единой внутренней информационной магистрали подключаются все основные функциональные блоки (арифметико-логический, рабочих регистров, стека, прерываний, интерфейса, управления и синхронизации и др.).

Для обоснования классификации микропроцессоров по числу БИС надо распределить все аппаратные блоки процессора между основными тремя функциональными частями: операционной, управляющей и интерфейсной. Сложность операционной и управляющей частей процессора определяется их разрядностью, системой команд и требованиями к системе прерываний; сложность интерфейсной части разрядностью и возможностями подключения других устройств ЭВМ (памяти, внешних устройств, датчиков и исполнительных механизмов и др.). Интерфейс процессора содержит несколько десятков информационных шин данных (ШД), адресов (ША) и управления (ШУ).

Однокристальные микропроцессоры получаются при реализации всех аппаратных средств процессора в виде одной БИС или СБИС (сверхбольшой интегральной схемы). По мере увеличения степени интеграции элементов в кристалле и числа выводов корпуса параметры однокристальных микропроцессоров улучшаются. Однако возможности однокристальных микропроцессоров ограничены аппаратными ресурсами кристалла и корпуса. Для получения многокристального микропроцессора необходимо провести разбиение его логической структуры на функционально законченные части и реализовать их в виде БИС (СБИС). Функциональная законченность БИС многокристального микропроцессора означает, что его части выполняют заранее определенные функции и могут работать автономно.

На рис. 4.4, *а* показано функциональное разбиение структуры процессора при создании трехкристального микропроцессора (пунктирные линии), содержащего БИС операционного (ОП), управляющего (УП) и интерфейсного (ИП) процессоров.

УП

ОП

ИП

УП

ОП

ИП

МА

МД

МУ

*а* *б*

Рис. 4.4. Функциональная структура процессора (*а*) и ее разбиение для реализации процессора в виде комплекта секционных БИС (*б*)

Операционный процессор служит для обработки данных, управляющий процессор выполняет функции выборки, декодирования и вычисления адресов операндов и также генерирует последовательности микрокоманд. Автономность работы и большое быстродействие БИС УП позволяет выбирать команды из памяти с большей скоростью, чем скорость их исполнения БИС ОП. При этом в УП образуется очередь еще не исполненных команд, а также заранее подготавливаются те данные, которые потребуются ОП в следующих циклах работы. Такая опережающая выборка команд экономит время ОП на ожидание операндов, необходимых для выполнения команд программ. Интерфейсный процессор позволяет подключить память и периферийные средства к микропроцессору; он, по существу, является сложным контроллером для устройств ввода/вывода информации. БИС ИП выполняет также функции канала прямого доступа к памяти.

Выбираемые из памяти команды распознаются и выполняются каждой частью микропроцессора автономно и поэтому может быть обеспечен режим одновременной работы всех БИС МП, т. е. конвейерный поточный режим исполнения последовательности команд программы (выполнение последовательности с небольшим временным сдвигом). Такой режим работы значительно повышает производительность микропроцессора.

Многокристальные секционные микропроцессоры получаются в том случае, когда в виде БИС реализуются части (секции) логической структуры процессора при функциональном разбиении ее вертикальными плоскостями (рис. 4.4, *б*). Для построения многоразрядных микропроцессоров при параллельном включении секций БИС в них добавляются средства «стыковки».

Для создания высокопроизводительных многоразрядных микропроцессоров требуется столь много аппаратных средств, не реализуемых в доступных БИС, что может возникнуть необходимость еще и в функциональном разбиении структуры микропроцессора горизонтальными плоскостями. В результате рассмотренного функционального разделения структуры микропроцессора на функционально и конструктивно законченные части создаются условия реализации каждой из них в виде БИС. Все они образуют комплект секционных БИС МП.

Таким образом, микропроцессорная секция – это БИС, предназначенная для обработки нескольких разрядов данных или выполнения определенных управляющих операций. Секционность БИС МП – определяет возможность «наращивания» разрядности обрабатываемых данных или усложнения устройств управления микропроцессора при «параллельном» включении большего числа БИС.

Однокристальные и трехкристальные БИС МП, как правило, изготовляют на основе микроэлектронных технологий униполярных полупроводниковых приборов, а многокристальные секционные БИС МП на основе технологии биполярных полупроводниковых приборов. Использование многокристальных микропроцессорных высокоскоростных биполярных БИС, имеющих функциональную законченность при малой физической разрядности обрабатываемых данных и монтируемых в корпус с большим числом выводов, позволяет организовать разветвление связи в процессоре, а также осуществить конвейерные принципы обработки информации для повышения его производительности.

По назначению различают универсальные и специализированные микропроцессоры.

Универсальные микропроцессоры могут быть применены для решения широкого круга разнообразных задач. При этом их эффективная производительность слабо зависит от проблемной специфики решаемых задач. Специализация МП, т. е. его проблемная ориентация на ускоренное выполнение определенных функций позволяет резко увеличить эффективную производительность при решении только определенных задач.

Среди специализированных микропроцессоров можно выделить различные микроконтроллеры, ориентированные на выполнение сложных последовательностей логических операций, математические МП, предназначенные для повышения производительности при выполнении арифметических операций за счет, например, матричных методов их выполнения, МП для обработки данных в различных областях применений и т. д. С помощью специализированных МП можно эффективно решать новые сложные задачи параллельной обработки данных. Например, конволюция позволяет осуществить более сложную математическую обработку сигналов, чем широко используемые методы корреляции. Последние в основном сводятся к сравнению и определению подобия всего двух серий данных: входных, передаваемых формой сигнала, и фиксированных опорных. Конволюция дает возможность в реальном масштабе времени находить соответствие для сигналов изменяющейся формы путем сравнения их с различными эталонными сигналами, что, например, может позволить эффективно выделить полезный сигнал на фоне шума.

Разработанные однокристальные конвольверы используются в устройствах опознавания образов в тех случаях, когда возможности сбора данных превосходят способности системы обрабатывать эти данные.

По виду обрабатываемых входных сигналов различают цифровые и аналоговые микропроцессоры. Сами микропроцессоры – цифровые устройства, однако могут иметь встроенные аналого-цифровые и цифро-аналоговые преобразователи. Поэтому входные аналоговые сигналы передаются в МП через преобразователь в цифровой форме, обрабатываются и после обратного преобразования в аналоговую форму поступают на выход. С архитектурной точки зрения такие микропроцессоры представляют собой аналоговые функциональные преобразователи сигналов и называются аналоговыми микропроцессорами. Они выполняют функции любой аналоговой схемы (например, производят генерацию колебаний, модуляцию, смещение, фильтрацию, кодирование и декодирование сигналов в реальном масштабе времени и т. д., заменяя сложные схемы, состоящие из операционных усилителей, катушек индуктивности, конденсаторов и т. д.). При этом применение аналогового микропроцессора значительно повышает точность обработки аналоговых сигналов и их воспроизводимость, а также расширяет функциональные возможности за счет программной «настройки» цифровой части микропроцессора на различные алгоритмы обработки сигналов.

Обычно в составе однокристальных аналоговых МП имеется несколько каналов аналого-цифрового и цифро-аналогового преобразования. В аналоговом микропроцессоре разрядность обрабатываемых данных достигает 24 бит и более, большое значение уделяется увеличению скорости выполнения арифметических операций.

Отличительная черта аналоговых микропроцессоров – способность к переработке большого объема числовых данных, т. е. к выполнению операций сложения и умножения с большой скоростью при необходимости даже за счет отказа от операций прерываний и переходов. Аналоговый сигнал, преобразованный в цифровую форму, обрабатывается в реальном масштабе времени и передается на выход обычно в аналоговой форме через цифро-аналоговый преобразователь. При этом согласно теореме Котельникова частота квантования аналогового сигнала должна вдвое превышать верхнюю частоту сигнала.

Сравнение цифровых микропроцессоров производится сопоставлением времени выполнения ими списков операций. Сравнение же аналоговых микропроцессоров производится по количеству эквивалентных звеньев аналого-цифровых фильтров рекурсивных фильтров второго порядка. Производительность аналогового микропроцессора определяется его способностью быстро выполнять операции умножения: чем быстрее осуществляется умножение, тем больше эквивалентное количество звеньев фильтра в аналоговом преобразователе и тем более сложный алгоритм преобразования цифровых сигналов можно задавать в микропроцессоре.

Одним из направлений дальнейшего совершенствования аналоговых микропроцессоров является повышение их универсальности и гибкости. Поэтому вместе с повышением скорости обработки большого объема цифровых данных будут развиваться средства обеспечения развитых вычислительных процессов обработки цифровой информации за счет реализации аппаратных блоков прерывания программ и программных переходов.

По характеру временной организации работы микропроцессоры делят на синхронные и асинхронные.

Синхронные микропроцессоры – микропроцессоры, в которых начало и конец выполнения операций задаются устройством управления (время выполнения операций в этом случае не зависит от вида выполняемых команд и величин операндов).

Асинхронные микропроцессоры позволяют начало выполнения каждой следующей операции определить по сигналу фактического окончания выполнения предыдущей операции. Для более эффективного использования каждого устройства микропроцессорной системы в состав асинхронно работающих устройств вводят электронные цепи, обеспечивающие автономное функционирование устройств. Закончив работу над какой-либо операцией, устройство вырабатывает сигнал запроса, означающий его готовность к выполнению следующей операции. При этом роль естественного распределителя работ принимает на себя память, которая в соответствии с заранее установленным приоритетом выполняет запросы остальных устройств по обеспечению их командной информацией и данными.

По организации структуры микропроцессорных систем различают микроЭВМ одно- и многомагистральные.

В одномагистральных микроЭВМ все устройства имеют одинаковый интерфейс и подключены к единой информационной магистрали, по которой передаются коды данных, адресов и управляющих сигналов.

В многомагистральныхмикроЭВМ устройства группами подключаются к своей информационной магистрали. Это позволяет осуществить одновременную передачу информационных сигналов по нескольким (или всем) магистралям. Такая организация систем усложняет их конструкцию, однако увеличивает производительность.

По количеству выполняемых программ различают одно- и многопрограммные микропроцессоры.

В однопрограммных микропроцессорах выполняется только одна программа. Переход к выполнению другой программы происходит после завершения текущей программы.

В много- или мультипрограммных микропроцессорах одновременно выполняется несколько (обычно несколько десятков) программ. Организация мультипрограммной работы микропроцессорных управляющих систем позволяет осуществить контроль за состоянием и управлением большим числом источников или приемников информации.

**4.1.3. Структура типового микропроцессора**

Архитектура типичной небольшой вычислительной системы на основе микроЭВМ показана на рис. 1.2. Такая микроЭВМ содержит все пять основных блоков цифровой машины: устройство ввода информации, управляющее устройство (УУ), арифметико-логическое устройство (АЛУ) (входящее в состав микропроцессора), запоминающие устройства (ЗУ) и устройство вывода информации.

Порты

Входы

Память данных ЗУПВ

Память программ ПЗУ

Микропроцессор

Выходы

Порты

От периферийных устройств

Шина

управления

Шина

адреса

16 линий

Шина

данных

(8 линий)

Рис. 4.5. Архитектура типового микропроцессора.

Микропроцессор координирует работу всех устройств цифровой системы с помощью шины управления (ШУ). Помимо ШУ имеется 16-разрядная адресная шина (ША), которая служит для выбора определенной ячейки памяти, порта ввода или порта вывода. По 8-разрядной информационной шине или шине данных (ШД) осуществляется двунаправленная пересылка данных к микропроцессору и от микропроцессора. Важно отметить, что МП может посылать информацию в память микроЭВМ или к одному из портов вывода, а также получать информацию из памяти или от одного из портов ввода.

Постоянное запоминающее устройство (ПЗУ) в микроЭВМ содержит некоторую программу (на практике программу инициализации ЭВМ). Программы могут быть загружены в запоминающее устройство с произвольной выборкой (ЗУПВ) и из внешнего запоминающего устройства (ВЗУ). Это программы пользователя.

В качестве примера, иллюстрирующего работу микроЭВМ, рассмотрим процедуру, для реализации которой нужно выполнить следующую последовательность элементарных операций:

1. Нажать клавишу с буквой «А» на клавиатуре.

2. Поместить букву «А» в память микроЭВМ.

3. Вывести букву «А» на экран дисплея.

Это типичная процедура ввода-запоминания-вывода, рассмотрение которой дает возможность пояснить принципы использования некоторых устройств, входящих в микроЭВМ.

На рис. 1.3 приведена подробная диаграмма выполнения процедуры ввода-запоминания-вывода. Обратите внимание, что команды уже загружены в первые шесть ячеек памяти. Хранимая программа содержит следующую цепочку команд:

1. Ввести данные из порта ввода 1.

2. Запомнить данные в ячейке памяти 200.

3. Переслать данные в порт вывода 10.

В данной программе всего три команды, хотя на рис. 1.2 может показаться, что в памяти программ записано шесть команд. Это связано с тем, что команда обычно разбивается на части. Первая часть команды 1 в приведенной выше программе – команда ввода данных. Во второй части команды 1 указывается, откуда нужно ввести данные (из порта 1). Первая часть команды, предписывающая конкретное действие, называется кодом операции (КОП), а вторая часть – операндом. Коды операции и операнд размещаются в отдельных ячейках памяти программ. На рис. 1.3 КОП хранится в ячейке 100, а код операнда – в ячейке 101 (порт 1); последний указывает, откуда нужно взять информацию.

ПОРТ1

ВХОД

Аккумулятор

МП

Регистр команд

Адрес

200

202

203

Содержимое

201

“А”

11

16

ВЫХОД

Порт 10

Адрес

100

101

102

103

104

105

106

Ввести данные

Из порта 1

Запомнить данные

В ячейки памяти 200

Вывести данные

В порт 10

Содержимое

1

3

6

8

12

14

16

2

4

7

9

13

15

5

16

5

Код символа “А” с

клавиатуры

Код символа “А” к монитору

Память программ

ЗУПВ

10

Рис. 4.6. Диаграмма выполнения процедуры ввода-запоминания-вывода

В МП на рис. 4.6 выделены еще два новых блока – регистры: аккумулятор и регистр команд.

Рассмотрим прохождение команд и данных внутри микроЭВМ с помощью занумерованных кружков на диаграмме. Напомним, что микропроцессор – это центральный узел, управляющий перемещением всех данных и выполнением операций.

Итак, при выполнении типичной процедуры ввода-запоминания-вывода в микроЭВМ происходит следующая последовательность действий:

1. МП выдает адрес 100 на шину адреса. По шине управления поступает сигнал, устанавливающий память программ (конкретную микросхему) в режим считывания.

2. ЗУ программ пересылает первую команду («Ввести данные») по шине данных, и МП получает это закодированное сообщение. Команда помещается в регистр команд. МП декодирует (интерпретирует) полученную команду и определяет, что для команды нужен операнд.

3. МП выдает адрес 101 на ША; ШУ используется для перевода памяти программ в режим считывания.

4. Из памяти программ на ШД пересылается операнд «Из порта 1». Этот операнд находится в программной памяти в ячейке 101. Код операнда (содержащий адрес порта 1) передается по ШД к МП и направляется в регистр команд. МП теперь декодирует полную команду («Ввести данные из порта 1»).

5. МП, используя ША и ШУ, связывающие его с устройством ввода, открывает порт 1. Цифровой код буквы «А» передается в аккумулятор внутри МП и запоминается.Важно отметить, что при обработке каждой программной команды МП действует согласно микропроцедуре выборки-декодирования-исполнения.

6. МП обращается к ячейке 102 по ША. ШУ используется для перевода памяти программ в режим считывания.

7. Код команды «Запомнить данные» подается на ШД и пересылается в МП, где помещается в регистр команд.

8. МП дешифрирует эту команду и определяет, что для нее нужен операнд. МП обращается к ячейке памяти 103 и приводит в активное состояние вход считывания микросхем памяти программ.

9. Из памяти программ на ШД пересылается код сообщения «В ячейке памяти 200». МП воспринимает этот операнд и помещает его в регистр команд. Полная команда «Запомнить данные в ячейке памяти 200» выбрана из памяти программ и декодирована.

10. Теперь начинается процесс выполнения команды. МП пересылает адрес 200 на ША и активизирует вход записи, относящийся к памяти данных.

11. МП направляет хранящуюся в аккумуляторе информацию в память данных. Код буквы «А» передается по ШД и записывается в ячейку 200 этой памяти. Выполнена вторая команда. Процесс запоминания не разрушает содержимого аккумулятора. В нем по-прежнему находится код буквы «А».

12. МП обращается к ячейке памяти 104 для выбора очередной команды и переводит память программ в режим считывания.

13. Код команды вывода данных пересылается по ШД к МП, который помещает ее в регистр команд, дешифрирует и определяет, что нужен операнд.

14. МП выдает адрес 105 на ША и устанавливает память программ в режим считывания.

15. Из памяти программ по ШД к МП поступает код операнда «В порт 10», который далее помещается в регистр команд.

16. МП дешифрирует полную команду «Вывести данные в порт 10». С помощью ША и ШУ, связывающих его с устройством вывода, МП открывает порт 10, пересылает код буквы «А» (все еще находящийся в аккумуляторе) по ШД. Буква «А» выводится через порт 10 на экран дисплея.

В большинстве микропроцессорных систем (МПС) передача информации осуществляется способом, аналогичным рассмотренному выше. Наиболее существенные различия возможны в блоках ввода и вывода информации.

Подчеркнем еще раз, что именно микропроцессор является ядром системы и осуществляет управление всеми операциями. Его работа представляет последовательную реализацию микропроцедур выборки-дешифрации-исполнения. Однако фактическая последовательность операций в МПС определяется командами, записанными в памяти программ.

Таким образом, в МПС микропроцессор выполняет следующие функции:

- выборку команд программы из основной памяти;

- дешифрацию команд;

- выполнение арифметических, логических и других операций, закодированных в командах;

- управление пересылкой информации между регистрами и основной памятью, между устройствами ввода/вывода;

- отработку сигналов от устройств ввода/вывода, в том числе реализацию прерываний с этих устройств;

- управление и координацию работы основных узлов МП.

**4.1.4. Логическая структура микропроцессора**

Логическая структура микропроцессора, т. е. конфигурация составляющих микропроцессор логических схем и связей между ними, определяется функциональным назначением. Именно структура задает состав логических блоков микропроцессора и то, как эти блоки должны быть связаны между собой, чтобы полностью отвечать архитектурным требованиям. Срабатывание электронных блоков микропроцессора в определенной последовательности приводит к выполнению заданных архитектурой микропроцессора функций, т. е. к реализации вычислительных алгоритмов. Одни и те же функции можно выполнить в микропроцессорах со структурой, отличающейся набором, количеством и порядком срабатывания логических блоков. Различные структуры микропроцессоров, как правило, обеспечивают их различные возможности, в том числе и различную скорость обработки данных. Логические блоки микропроцессора с развитой архитектурой показаны на рис. 1.3.

БС

БУВВ

БУВП

БУПК

БУФКА

БУПРПр

БУВОп

БЗП

РгСОЗУ

АЛБ

БДА1

БДАn

II

I

Рис. 4.7. Общая логическая структура микропроцессора:

*I – управляющая часть, II – операционная часть; БУПК – блок управления последовательно-стью команд; БУВОп – блок управления выполнением операций; БУФКА ­ блок управления формированием кодов адресов; БУВП – блок управления виртуальной памятью; БЗП – блок защиты памяти; БУПРПр – блок управления прерыванием работы процессора; БУВВ ­ блок управления вводом/выводом; РгСОЗУ­ регистровое сверхоперативное запоминающее устройство; АЛБ – арифметико-логический блок; БДА – блок дополнительной арифметики; БС – блок синхронизации*

При проектировании логической структуры микропроцессоров необходимо рассмотреть:

1) номенклатуру электронных блоков, необходимую и достаточную для реализации архитектурных требований;

2) способы и средства реализации связей между электронными блоками;   
3) методы отбора наиболее рациональных вариантов логических структур из возможного числа структур с отличающимся составом блоков и конфигурацией связей между ними.

При проектировании микропроцессора приводятся в соответствие внутренняя сложность кристалла и количество выводов корпуса. Относительный рост числа элементов по мере развития микроэлектронной технологии во много раз превышает относительное увеличение числа выводов корпуса, поэтому проектирование БИС в виде конечного автомата, а не в виде набора схем, реализующих некоторый набор логических переключательных функций и схем памяти, дает возможность получить функционально законченные блоки и устройства ЭВМ.

Использование микропроцессорных комплектов БИС позволяет создать микроЭВМ для широких областей применения вследствие программной адаптации микропроцессора к конкретной области применения: изменяя программу работы микропроцессора, изменяют функции информационно-управляющей системы. Поэтому за счет составления программы работы микропроцессоров в конкретных условиях работы определенной системы можно получить оптимальные характеристики последней.

Если уровень только программной «настройки» микропроцессоров не позволит получить эффективную систему, доступен следующий уровень проектирования – микропрограммный. За счет изменения содержимого ПЗУ или программируемой логической матрицы (ПЛМ) можно «настроиться» на более специфичные черты системы обработки информации. В этом случае частично за счет изменения микропрограмм затрагивается аппаратный уровень системы. Технико-экономические последствия здесь связаны лишь с ограниченным вмешательством в технологию изготовления управляющих блоков микроЭВМ.

Изменение аппаратного уровня информационно-управляющей микропроцессорной системы, включающего в себя функциональные БИС комплекта, одновременно с конкретизацией микропрограммного и программного уровней позволяет наилучшим образом удовлетворить требованиям, предъявляемым к системе.

Решение задач управления в конкретной системе чисто аппаратными средствами (аппаратная логика) дает выигрыш в быстродействии, однако приводит к сложностям при модификации системы. Микропроцессорное решение (программная логика) является более медленным, но более гибким решением, позволяющим развивать и модифицировать систему. Изменение технических требований к информационно-управляющей микропроцессорной системе ведет лишь к необходимости перепрограммирования работы микропроцессора. Именно это качество обеспечивает высокую логическую гибкость микропроцессоров, определяет возможность их широкого использования, а значит и крупносерийного производства.

**4.1.5. Типы архитектур**

Существует несколько подходов к классификации микропроцессоров по типу архитектуры. Так, выделяют МП с CISC (CompleteInstructionSetComputer) архитектурой, характеризуемой полным набором команд, и RISC (ReduceInstructionSetComputer) архитектурой, которая определяет систему с сокращенным набором команд одинакового формата, выполняемых за один такт МП.

Определяя в качестве основной характеристики МП разрядность, выделяют следующие типы МП архитектуры:

* с фиксированной разрядностью и списком команд (однокристальные);
* с наращиваемой разрядностью (секционные) и микропрограммным управлением.

Анализируя адресные пространства программ и данных, определяют МП с архитектурой фон Неймана (память программ и память данных находятся в едином пространстве и нет никаких признаков, указывающих на тип информации в ячейке памяти) и МП с архитектурой Гарвардской лаборатории (память программ и память данных разделены, имеют свои адресные пространства и способы доступа к ним).

Для выполнения команд в структуру процессора входят внутренние регистры, *арифметико-логическое устройство* (*АЛУ*, *ALU* – *ArithmeticLogicUnit*), мультиплексоры, буферы, регистры и другие узлы. Работа всех узлов синхронизируется общим внешним тактовым сигналом процессора. То есть процессор представляет собой довольно сложное цифровое устройство ([рис. 4.](http://www.intuit.ru/department/hardware/mpbasics/1/#image.1.4)8).

Микропроцессор

Арифметическо-логическое устройство(АЛУ)

Схема управления выборкой команд

Схема управления прерываниями

RG

RG

**.**

**.**

**.**

Регистр признаков

Схема управления ПДП

Логика

управления

RG

RG

Рис. 4.8.  Пример структуры простейшего процессора

Впрочем, для разработчика *микропроцессорных систем* информация о тонкостях внутренней структуры процессора не слишком важна. Разработчик должен рассматривать процессор как «*черный ящик*», который в ответ на входные и *управляющие коды* производит ту или иную операцию и выдает выходные сигналы. Разработчику необходимо знать систему команд, режимы работы процессора, а также правила взаимодействия процессора с внешним миром или, как их еще называют, протоколы обмена информацией. О внутренней структуре процессора надо знать только то, что необходимо для выбора той или иной команды, того или иного режима работы.

**4.2. Обмен данными в микропроцессорной системе**

**4.2.1. Шинная структура связей**

Для достижения максимальной универсальности и упрощения *протоколов обмена* информацией в *микропроцессорных системах* применяется так называемая шинная структура связей между отдельными устройствами, входящими в систему. Суть шинной структуры связей сводится к следующему.

При классической структуре связей ([рис. 4.9](http://www.intuit.ru/department/hardware/mpbasics/1/2.html#image.1.5)) все сигналы и коды между устройствами передаются по отдельным линиям связи. Каждое устройство, входящее в систему, передает свои сигналы и коды независимо от других устройств. При этом в системе получается очень много линий связи и разных *протоколов обмена* информацией.

Устройство 1

Устройство 3

Устройство 2

Устройство 4

Рис. 4.10.  Классическая структура связей

При шинной структуре связей ([рис. 4.10](http://www.intuit.ru/department/hardware/mpbasics/1/2.html#image.1.6)) все сигналы между устройствами передаются по одним и тем же линиям связи, но в разное время (это называется мультиплексированной передачей). Причем передача по всем линиям связи может осуществляться в обоих направлениях (так называемая двунаправленная передача). В результате количество линий связи существенно сокращается, а правила обмена (протоколы) упрощаются. Группа линий связи, по которым передаются сигналы или коды, как раз и называется ***шиной*** (англ. bus).

Устройство 1

Устройство 2

Устройство 3

Устройство 4

Рис. 4.11.  Шинная структура связей

Понятно, что при шинной структуре связей легко осуществляется пересылка всех информационных потоков в нужном направлении, например, их можно пропустить через один процессор, что очень важно для *микропроцессорной системы*. Однако при шинной структуре связей вся информация передается по линиям связи последовательно во времени, по очереди, что снижает быстродействие системы по сравнению с классической структурой связей.

Большое достоинство шинной структуры связей состоит в том, что все устройства, подключенные к *шине*, должны принимать и передавать информацию по одним и тем же правилам (протоколам обмена информацией по *шине*). Соответственно, все узлы, отвечающие за обмен с *шиной* в этих устройствах, должны быть единообразны, унифицированы.

Существенный недостаток шинной структуры связан с тем, что все устройства подключаются к каждой линии связи параллельно. Поэтому любая неисправность любого устройства может вывести из строя всю систему, если она портит линию связи. По этой же причине отладка системы с шинной структурой связей довольно сложна и обычно требует специального оборудования.

В системах с шинной структурой связей применяют все три существующие разновидности выходных каскадов цифровых микросхем:

* стандартный выход или выход с двумя состояниями (обозначается 2С, 2S, реже ТТЛ, *TTL*);
* выход с открытым коллектором (обозначается ОК, *OC*);
* выход с тремя состояниями или (что-то же самое) с возможностью отключения (обозначается 3С, 3S).

Упрощенно эти три типа выходных каскадов могут быть представлены в виде схем на [рис. 4.](http://www.intuit.ru/department/hardware/mpbasics/1/2.html#image.1.7)12.

Выход 2С

Выход 3С

Рис. 4.12.  Три типа выходов цифровых микросхем

У выхода 2С два ключа замыкаются по очереди, что соответствует уровням логической единицы (верхний ключ замкнут) и логического нуля (нижний ключ замкнут). У выхода ОК замкнутый ключ формирует уровень логического нуля, разомкнутый – логической единицы. У выхода 3С ключи могут замыкаться по очереди (как в случае 2С), а могут размыкаться одновременно, образуя третье, высокоимпедансное состояние. Переход в третье состояние (Z-состояние) управляется сигналом на специальном входе EZ.

Выходные каскады типов 3С и ОК позволяют объединять несколько выходов микросхем для получения мультиплексированных ([рис. 4.13](http://www.intuit.ru/department/hardware/mpbasics/1/2.html#image.1.8)) или двунаправленных ([рис. 4.14](http://www.intuit.ru/department/hardware/mpbasics/1/2.html#image.1.9)) линий.

◊

Вых.

Вх.

◊

Вых.

◊

Вых.

Рис. 4.13.  Мультиплексированная линия

◊

Вх. Вых.

Вх.

◊

Вых.

◊

Вх. Вых.

Рис. 4.14.  Двунаправленная линия

При этом в случае выходов 3С необходимо обеспечить, чтобы на линии всегда работал только один активный выход, а все остальные выходы находились бы в это время в третьем состоянии, иначе возможны конфликты. Объединенные выходы ОК могут работать все одновременно, без всяких конфликтов.

Типичная структура *микропроцессорной системы* приведена на [рис. 2.10](http://www.intuit.ru/department/hardware/mpbasics/1/2.html#image.1.10). Она включает в себя три основных типа устройств:

* процессор;
* *память*, включающую оперативную ***память*** (ОЗУ, RAM –*RandomAccess*Memory) и постоянную *память*( ПЗУ, ROM –ReadOnlyMemory), которая служит для хранения данных и программ;
* ***устройства ввода/вывода*** (УВВ, I/O–Input/*OutputDevices*), служащие для связи *микропроцессорной системы* с внешними устройствами, для приема (ввода, чтения, Read) входных сигналов и выдачи (вывода, записи, Write) выходных сигналов.

Выходные

сигналы

Входные

сигналы

Устройство ввода/вывода

Память

Процессор

Шина

питания

Системная шина

(магистраль)

Шина данных

Шина

управления

Шина адреса

Рис. 4.15.  Структура микропроцессорной системы

Все устройства *микропроцессорной системы* объединяются общей системной ***шиной*** (она же называется еще **системной магистралью** или **каналом**). Системная *магистраль* включает в себя четыре основные *шины* нижнего уровня:

* *шина* адреса (*AddressBus*);
* *шина* данных (*DataBus*);
* *шина* управления (*ControlBus*);
* *шина* питания (*PowerBus*).

*Шина* адреса служит для определения адреса (номера) устройства, с которым процессор обменивается информацией в данный момент. Каждому устройству (кроме процессора), каждой ячейке *памяти* в *микропроцессорной системе* присваивается собственный адрес. Когда код какого-то адреса выставляется процессором на *шине* адреса, устройство, которому этот адрес приписан, понимает, что ему предстоит обмен информацией. *Шина* адреса может быть однонаправленной или двунаправленной.

*Шина* данных – это основная *шина*, которая используется для передачи информационных кодов между всеми устройствами *микропроцессорной системы*. Обычно в пересылке информации участвует процессор, который передает код данных в какое-то устройство или в ячейку *памяти* или же принимает код данных из какого-то устройства или из ячейки *памяти*. Но возможна также и передача информации между устройствами без участия процессора. *Шина* данных всегда двунаправленная.

*Шина* управления в отличие от *шины* адреса и *шины* данных состоит из отдельных управляющих сигналов. Каждый из этих сигналов во время обмена информацией имеет свою функцию. Некоторые сигналы служат для стробирования передаваемых или принимаемых данных (то есть определяют моменты времени, когда информационный код выставлен на *шину* данных). Другие управляющие сигналы могут использоваться для подтверждения приема данных, для сброса всех устройств в исходное состояние, для тактирования всех устройств и т.д. Линии *шины* управления могут быть однонаправленными или двунаправленными.

Наконец, *шина* питания предназначена не для пересылки информационных сигналов, а для питания системы. Она состоит из линий питания и общего провода. В *микропроцессорной системе* может быть один источник питания (чаще +5 В) или несколько источников питания (обычно еще –5 В, +12 В и –12 В). Каждому напряжению питания соответствует своя линия связи. Все устройства подключены к этим линиям параллельно.

Если в *микропроцессорную систему* надо ввести входной код (или входной сигнал), то процессор по *шине* адреса обращается к нужному *устройству ввода/вывода* и принимает по *шине* данных *входную информацию*. Если из *микропроцессорной системы* надо вывести выходной код (или выходной сигнал), то процессор обращается по *шине* адреса к нужному *устройству ввода/вывода* и передает ему по *шине* данных *выходную информацию*.

Если информация должна пройти сложную многоступенчатую обработку, то процессор может хранить промежуточные результаты в системной оперативной *памяти*. Для обращения к любой ячейке *памяти* процессор выставляет ее адрес на *шину* адреса и передает в нее информационный код по *шине* данных или же принимает из нее информационный код по *шине* данных. В *памяти* (оперативной и постоянной) находятся также и *управляющие коды* (команды выполняемой процессором программы), которые процессор также читает по *шине* данных с адресацией по *шине* адреса. Постоянная *память* используется в основном для хранения программы начального пуска *микропроцессорной системы*, которая выполняется каждый раз после включения питания. Информация в нее заносится изготовителем раз и навсегда.

Таким образом, в *микропроцессорной системе* все информационные коды и коды команд передаются по *шинам* последовательно, по очереди. Это определяет сравнительно невысокое быстродействие *микропроцессорной системы*. Оно ограничено обычно даже не быстродействием процессора (которое тоже очень важно) и не скоростью обмена по системной *шине* (магистрали), а именно последовательным характером передачи информации по системной *шине* (магистрали).

Важно учитывать, что *устройства ввода/вывода* чаще всего представляют собой устройства на «жесткой логике». На них может быть возложена часть функций, выполняемых *микропроцессорной системой*. Поэтому у разработчика всегда имеется возможность перераспределять функции системы между аппаратной и программной реализациями оптимальным образом. Аппаратная реализация ускоряет выполнение функции, но имеет недостаточную гибкость. Программная реализация значительно медленнее, но обеспечивает высокую гибкость. Аппаратная реализация функций увеличивает стоимость системы и ее *энергопотребление*, программная –не увеличивает. Чаще всего применяется комбинирование аппаратных и программных функций.

Иногда *устройства ввода/вывода* имеют в своем составе процессор, то есть представляют собой небольшую специализированную *микропроцессорную систему*. Это позволяет переложить часть программных функций на *устройства ввода/вывода*, разгрузив центральный процессор системы.

**4.2.2. Режимы работы микропроцессорной системы.** Как уже отмечалось, *микропроцессорная система* обеспечивает большую гибкость работы, она способна настраиваться на любую задачу. Гибкость эта обусловлена прежде всего тем, что функции, выполняемые системой, определяются программой (программным обеспечением, software), которую выполняет процессор. Аппаратура (аппаратное обеспечение, hardware) остается неизменной при любой задаче. Записывая в *память* системы программу, можно заставить *микропроцессорную систему* выполнять любую задачу, поддерживаемую данной аппаратурой. К тому же шинная *организация связеймикропроцессорной системы* позволяет довольно легко заменять аппаратные модули, например, заменять *память* на новую большего объема или более высокого быстродействия, добавлять или модернизировать *устройства ввода/вывода*, наконец, заменять процессор на более мощный. Это также позволяет увеличить гибкость системы, продлить ее жизнь при любом изменении требований к ней.

Но гибкость *микропроцессорной системы* определяется не только этим. Настраиваться на задачу помогает еще и выбор режима работы системы, то есть режима обмена информацией по системной магистрали ( *шине* ).

Практически любая развитая *микропроцессорная система* (в том числе и компьютер) поддерживает три основных режима обмена по магистрали:

* программный обмен информацией;
* обмен с использованием прерываний (*Interrupts*);
* обмен с использованием прямого доступа к *памяти* (*ПДП*, *DMA* — *DirectMemoryAccess*).

**Программный обмен информацией** является основным в любой *микропроцессорной системе*. Он предусмотрен всегда, без него невозможны другие режимы обмена. В этом режиме процессор является единоличным хозяином (или задатчиком, Master) системной магистрали. Все операции (циклы) обмена информацией в данном случае инициируются только процессором, все они выполняются строго в порядке, предписанном исполняемой программой.

Процессор читает (выбирает) из *памяти* коды команд и исполняет их, читая данные из *памяти* или из *устройства ввода/вывода*, обрабатывая их, записывая данные в *память* или передавая их в *устройство ввода/вывода*. Путь процессора по программе может быть линейным, циклическим, может содержать переходы (прыжки), но он всегда непрерывен и полностью находится под контролем процессора. Ни на какие внешние события, не связанные с программой, процессор не реагирует (рис. 4.16). Все сигналы на магистрали в данном случае контролируются процессором.

Команда 1

Команда 2

Команда 3

●

●

●

Команда N

Команда N+1

Команда N+2

Команда N+3

Команда N+4

●

●

●

Выполнение команд

Программа

Последовательно

Цикл

Прыжок

**Рис. 4.16.**  Программный обмен информацией.

**Обмен по прерываниям** используется тогда, когда необходима реакция *микропроцессорной системы* на какое-то внешнее событие, на приход внешнего сигнала. В случае компьютера внешним событием может быть, например, нажатие на клавишу клавиатуры или приход по локальной сети пакета данных. Компьютер должен реагировать на это, соответственно, выводом символа на экран или же чтением и обработкой принятого по сети пакета.

В общем случае организовать реакцию на внешнее событие можно тремя различными путями:

* с помощью постоянного программного контроля факта наступления события (так называемый метод опроса флага или *polling*);
* с помощью прерывания, то есть насильственного перевода процессора с выполнения текущей программы на выполнение экстренно необходимой программы;
* с помощью прямого доступа к *памяти*, то есть без участия процессора при его отключении от системной магистрали.

Проиллюстрировать эти три способа можно следующим простым примером. Допустим, вы готовите себе завтрак, поставив на плиту кипятиться молоко. Естественно, на закипание молока надо реагировать, причем срочно. Как это организовать? Первый путь — постоянно следить за молоком, но тогда вы ничего другого не сможете делать. Правильнее будет регулярно поглядывать на молоко, делая одновременно что-то другое. Это программный режим с опросом флага. Второй путь — установить на кастрюлю с молоком датчик, который подаст звуковой сигнал при закипании молока, и спокойно заниматься другими делами. Услышав сигнал, вы выключите молоко. Правда, возможно, вам придется сначала закончить то, что вы начали делать, так что ваша реакция будет медленнее, чем в первом случае. Наконец, третий путь состоит в том, чтобы соединить датчик на кастрюле с управлением плитой так, чтобы при закипании молока горелка была выключена без вашего участия (правда, аналогия с *ПДП* здесь не очень точная, так как в данном случае на момент выполнения действия вас не отвлекают от работы).

Первый случай с опросом флага реализуется в *микропроцессорной системе* постоянным чтением информации процессором из *устройства ввода/вывода*, связанного с тем внешним устройством, на поведение которого необходимо срочно реагировать.

Во втором случае в режиме прерывания процессор, получив запрос прерывания от внешнего устройства (часто называемый *IRQ* — *InterruptReQuest*), заканчивает выполнение текущей команды и переходит к программе обработки прерывания. Закончив выполнение программы обработки прерывания, он возвращается к прерванной программе с той точки, где его прервали ([рис. 4.1](http://www.intuit.ru/department/hardware/mpbasics/1/3.html#image.1.12)7).

Здесь важно то, что вся работа, как и в случае программного режима, осуществляется самим процессором, внешнее событие просто временно отвлекает его. Реакция на внешнее событие по прерыванию в общем случае медленнее, чем при программном режиме. Как и в случае программного обмена, здесь все сигналы на магистрали выставляются процессором, то есть он полностью контролирует *магистраль*. Для обслуживания прерываний в систему иногда вводится специальный модуль *контроллера прерываний*, но он в обмене информацией не участвует. Его задача состоит в том, чтобы упростить работу процессора с внешними *запросами прерываний*. Этот контроллер обычно программно управляется процессором по системной магистрали.

Команда 1

Команда 2

Команда 3

●

●

●

Команда N

Команда N+1

Команда N+2

●

●

●

Команда 1

Команда 2

Команда 3

●

●

●

Команда M

Программа обработки прерывания

Запрос прерывания

Выполнение команд

Выполнение команд

Основная программа

**Рис. 4.17.**  Обслуживание прерывания.

Естественно, никакого ускорения работы системы прерывание не дает. Его применение позволяет только отказаться от постоянного опроса флага внешнего события и временно, до наступления внешнего события, занять процессор выполнением каких-то других задач.

**Прямой доступ к памяти (ПДП, DMA)** — это режим, принципиально отличающийся от двух ранее рассмотренных режимов тем, что обмен по системной *шине* идет без участия процессора. Внешнее устройство, требующее обслуживания, сигнализирует процессору, что режим *ПДП* необходим, в ответ на это процессор заканчивает выполнение текущей команды и отключается от всех *шин*, сигнализируя запросившему устройству, что обмен в режиме *ПДП* можно начинать.

Операция *ПДП* сводится к пересылке информации из *устройства ввода/вывода* в *память* или же из *памяти* в *устройство ввода/вывода*. Когда пересылка информации будет закончена, процессор вновь возвращается к прерванной программе, продолжая ее с той точки, где его прервали ([рис. 1.13](http://www.intuit.ru/department/hardware/mpbasics/1/3.html#image.1.13)). Это похоже на режим обслуживания прерываний, но в данном случае процессор не участвует в обмене. Как и в случае прерываний, реакция на внешнее событие при *ПДП* существенно медленнее, чем при программном режиме.

Понятно, что в этом случае требуется введение в систему дополнительного устройства (контроллера *ПДП*), которое будет осуществлять полноценный обмен по системной магистрали без всякого участия процессора. Причем процессор предварительно должен сообщить этому контроллеру *ПДП*, откуда ему следует брать информацию и/или куда ее следует помещать. Контроллер *ПДП* может считаться специализированным процессором, который отличается тем, что сам не участвует в обмене, не принимает в себя информацию и не выдает ее ([рис. 4.1](http://www.intuit.ru/department/hardware/mpbasics/1/3.html#image.1.14)8).

**Рис. 4.18.**  Обслуживание ПДП.

Команда 1

Команда 2

Команда 3

●

●

●

Команда N

Команда N+1

Команда N+2

Команда N+3

Команда N+4

●

●

●

Выполнение команд

Запрос ПДП

Точка остановки на время ПДП

Продолжение

выполнения команд

Программа

Процессор

Контроллер

ПДП

Память

Устройство

Ввода/выводы

Системная магистраль

**Рис. 4.19.**  Информационные потоки в режиме ПДП.

В принципе контроллер *ПДП* может входить в состав *устройства ввода/вывода*, которому необходим режим *ПДП* или даже в состав нескольких *устройств ввода/вывода*. Теоретически обмен с помощью прямого доступа к *памяти* может обеспечить более высокую скорость передачи информации, чем программный обмен, так как процессор передает данные медленнее, чем специализированный контроллер *ПДП*. Однако на практике это преимущество реализуется далеко не всегда. Скорость обмена в режиме *ПДП* обычно ограничена возможностями магистрали. К тому же необходимость программного задания режимов контроллера *ПДП* может свести на нет выигрыш от более высокой скорости пересылки данных в режиме *ПДП*. Поэтому режим *ПДП* применяется редко.

Если в системе уже имеется самостоятельный контроллер *ПДП*, то это может в ряде случаев существенно упростить аппаратуру *устройств ввода/вывода*, работающих в режиме *ПДП*. В этом, пожалуй, состоит единственное бесспорное преимущество режима *ПДП*.

**1.4. Архитектура микропроцессорных систем.** До сих пор мы рассматривали только один тип *архитектуры*   *микропроцессорных систем* — *архитектуру* с общей, единой *шиной* для данных и команд (одношинную, или **принстонскую**, **фон-неймановскую**   *архитектуру* ). Соответственно, в составе системы в этом случае присутствует одна общая *память*, как для данных, так и для команд ([рис. 4.](http://www.intuit.ru/department/hardware/mpbasics/1/4.html#image.1.15)20).

Процессор

Устройства

Ввода/вывода

Память

Команды

Данные

Общая шина данных и команд

**Рис. 4.20.**  Архитектура с общей шиной данных и команд.

Но существует также и альтернативный тип *архитектурымикропроцессорной системы* — это *архитектура* с раздельными *шинами* данных и команд (двухшинная, или гарвардская, *архитектура* ). Эта *архитектура* предполагает наличие в системе отдельной *памяти* для данных и отдельной *памяти* для команд ([рис. 1.16](http://www.intuit.ru/department/hardware/mpbasics/1/4.html#image.1.16)). Обмен процессора с каждым из двух типов *памяти* происходит по своей *шине*.

*Архитектура* с общей *шиной* распространена гораздо больше, она применяется, например, в персональных компьютерах и в сложных *микрокомпьютерах*. *Архитектура* с раздельными *шинами* применяется в основном в однокристальных *микроконтроллерах*.

Рассмотрим некоторые достоинства и недостатки обоих архитектурных решений.

*Архитектура* с общей *шиной* (принстонская, фон-неймановская) проще, она не требует от процессора одновременного обслуживания двух *шин*, контроля обмена по двум *шинам* сразу. Наличие единой *памяти* данных и команд позволяет гибко распределять ее объем между кодами данных и команд. Например, в некоторых случаях нужна большая и сложная программа, а данных в *памяти* надо хранить не слишком много. В других случаях, наоборот, программа требуется простая, но необходимы большие объемы хранимых данных. Перераспределение *памяти* не вызывает никаких проблем, главное — чтобы программа и данные вместе помещались в *памяти* системы. Как правило, в системах с такой *архитектурой память* бывает довольно большого объема (до десятков и сотен мегабайт). Это позволяет решать самые сложные задачи.

Шина команд

Память команд

Процессор

Устройства

ввода/вывода

Память данных

Шина данных

**Рис. 4.21.**  Архитектура с раздельными шинами данных и команд.

*Архитектура* с раздельными *шинами* данных и команд сложнее, она заставляет процессор работать одновременно с двумя потоками кодов, обслуживать обмен по двум *шинам* одновременно. Программа может размещаться только в *памяти* команд, данные — только в *памяти* данных. Такая узкая специализация ограничивает круг задач, решаемых системой, так как не дает возможности гибкого перераспределения *памяти*. *Память* данных и *память* команд в этом случае имеют не слишком большой объем, поэтому применение систем с данной *архитектурой* ограничивается обычно не слишком сложными задачами.

В чем же преимущество *архитектуры* с двумя *шинами* (гарвардской)? В первую очередь, в быстродействии.

Дело в том, что при единственной *шине* команд и данных процессор вынужден по одной этой *шине* принимать данные (из *памяти* или *устройства ввода/вывода* ) и передавать данные (в *память* или в *устройство ввода/вывода* ), а также читать команды из *памяти*. Естественно, одновременно эти пересылки кодов по магистрали происходить не могут, они должны производиться по очереди. Современные процессоры способны совместить во времени выполнение команд и проведение циклов обмена по системной *шине*. Использование конвейерных технологий и быстрой кэш-памяти позволяет им ускорить процесс взаимодействия со сравнительно медленной системной *памятью*. Повышение тактовой частоты и совершенствование структуры процессоров дают возможность сократить время выполнения команд. Но дальнейшее увеличение быстродействия системы возможно только при совмещении пересылки данных и чтения команд, то есть при переходе к *архитектуре* с двумя *шинами*.

В случае двухшинной *архитектуры* обмен по обеим *шинам* может быть независимым, параллельным во времени. Соответственно, структуры *шин* (количество разрядов кода адреса и кода данных, порядок и скорость обмена информацией и т.д.) могут быть выбраны оптимально для той задачи, которая решается каждой *шиной*. Поэтому при прочих равных условиях переход на двухшинную *архитектуру* ускоряет работу *микропроцессорной системы*, хотя и требует дополнительных затрат на аппаратуру, усложнения структуры процессора. *Память* данных в этом случае имеет свое распределение адресов, а *память* команд — свое.

Проще всего преимущества двухшинной *архитектуры* реализуются внутри одной микросхемы. В этом случае можно также существенно уменьшить влияние недостатков этой *архитектуры*. Поэтому основное ее применение — в *микроконтроллерах*, от которых не требуется решения слишком сложных задач, но зато необходимо максимальное быстродействие при заданной тактовой частоте.

**4. Архитектура 16-тиразрядного микропроцессора**

Мы рассмотрим более подробно основные типы архитектурных решений, выделяя связь со способами адресации памяти. 1. Регистровая архитектура определяется наличием достаточно большого регистрового файла внутри МП. Команды получают возможность обратиться к операндам, расположенным в одной из двух запоминающих сред: оперативной памяти или регистрах. Размер регистра обычно фиксирован и совпадает с размером слова, физически реализованного в оперативной памяти. К любому регистру можно обратиться непосредственно, поскольку регистры представлены в виде массива запоминающих элементов – регистрового файла. Типичным является выполнение арифметических операций только в регистре, при этом команда содержит два операнда (оба операнда в регистре или один операнд в регистре, а второй в оперативной памяти).

**4.1. Структура микропроцессора Intel8080**

Intel8080 является однокристальным микропроцессором, работающим с 8-разрядной шиной данных и 16-разрядной шиной адреса. Управляющие сигналы передаются по шине управления. Шины отделены друг от друга. Структура Intel 8080 приведена на рис. 4.22.

Буфер данных

Аккумулятор

Буфер аккумулятора

Арифметико-

Логическое устройство

Регистр

флагов

Схема инкремента

JP(16)

SP(16)

B(8) C(8)

D(8) E(8)

H(8) L(8)

W(8) Z(8)

Буфер адреса

Дешифратор выбора регистров

Устройство

управления

Intel 8080

Дешифратор

команд

Регистр

команд

16

8

Внутренняя шина данных

8

Схема десятичной коррекции

8

RC

Управление обменом информацией

TR

Рис. 4.22. Внутренняя структура микропроцессораIntel8080

Микропроцессор содержит внутреннюю шину данных, посредством которой происходит обмен информацией между внутренними регистрами, арифметико-логическим устройством, обрабатывающим 8-разрядные данные и передающим их через буфер на внешнюю шину данных. Кроме того, в состав 8080 входит устройство управления, буфер адресной шины, связанный с регистром команд, и блок регистров. В общих чертах работа микропроцессора выглядит следующим образом: в регистре, называемом программным счетчиком, хранится адрес следующей команды, которую необходимо выполнить. Устройство управления подключает этот регистр к шине адреса (конечно, через буфер) и выдает управляющие сигналы, необходимые для чтения кода команды из памяти. На этом завершается первый такт.

В следующем такте микропроцессор проверяет состояние сигналов на входе готовности и запрос останова. При их наличии микропроцессор переходит в соответствующее состояние. В противном случае, после появления на шине управления сигналов, подтверждающих выдачу кода команды на шину данных, устройство управления подключает к ней регистр команды и записывает в нее полученный код. Это требуется потому, что команда передается только в первом машинном цикле, а сохранить ее нужно на все время выполнения команды. Из регистра команды ее код поступает в дешифратор команды и затем в устройство управления, которое в зависимости от поступившей команды либо сразу переходит к ее выполнению, либо считывает данные или адрес, расположенные сразу после кода команды и необходимые для ее выполнения. На это тратится третий такт и, если это необходимо, четвертый и пятый такты. Таким образом, вся команда выполняется за 3–5 тактов. При тактовой частоте 2 МГц это составляет 1,5 – 2,5 мкс.

Перед выполнением команды проверяется состояние сигнала на входе захвата шины HLD (этот сигнал отключает микропроцессор от шины, давая внешним устройствам возможность прямого доступа в память по общей шине). При его обнаружении микропроцессор переходит в состояние захвата и вырабатывает сигнал подтверждения захвата шины. После снятия сигнала HLD процессор продолжает выполнение команды. В конце машинного цикла вновь анализируется состояние входа захвата, а затем проверяется, завершено ли выполнение команды. Если нет, то микропроцессор переходит к выполнению следующего цикла команды. Это может быть цикл работы с оперативной памятью или с внешним устройством.

После выполнения каждой команды проверяется состояние входа запроса прерывания. Если этот сигнал присутствует, то текущая программа приостанавливается и на шину данных выдается сигнал «подтверждение прерывания». Затем внешний контроллер прерываний передает по шине данных команду и адрес перехода к подпрограмме обработки прерывания. По окончании обработки прерывания происходит возврат к выполнению прерванной программы.

Сигнал готовности позволяет синхронизировать работу микропроцессора с работой более медленных устройств, и, кроме того, используется для пошагового выполнения программ.

Следующим важным узлом микропроцессора является блок регистров. Он включает в себя 16-разрядный регистр для временного хранения данных WZ, шесть 8-разрядных регистров общего назначения B, C, D, E, H, L, которые могут использоваться парами в качестве 16-разрядных – BC, DE, HL (это сделано прежде всего для удобной работы с адресами). Кроме того, блок регистров содержит 16-разрядный регистр адреса команды IP (программный счетчик), 16-разрядный регистр указателя стека SP, а также 16-разрядную схему инкремента-декремента. С помощью последней изменяется, например, состояние программного счетчика после выполнения каждой следующей команды.

Еще один важный узел – регистр результата (аккумулятор), связанный с АЛУ и используемый для хранения одного из исходных операндов или результата выполнения команды.

Последний регистр –это регистр флажков. В нем записан байт, каждый бит которого содержит информацию о результате выполнения последней команды.

**4.2. Архитектура микропроцессора 8086**

Микропроцессор Intel 8086 существенно отличается от i8080. В нем применена новая значительно более мощная и гибкая система команд, есть возможность адресации 1 Мбайта памяти, обращения к 65536 устройствам ввода и такому же количеству устройств вывода информации.

В i8086 имеется возможность изменения внутренней аппаратной конфигурации с помощью специального управляющего сигнала. В более простом режиме 8086 ориентирован на использование в простых вычислительных и управляющих устройствах. При этом микропроцессор сам вырабатывает сигналы управления шиной и обеспечивает прямой доступ к ней посредством контроллера Intel 8257. В режиме полной конфигурации обеспечивается работа с контроллером шины 8288, который декодирует три сигнала состояния процессора и в зависимости от них выдает семь сигналов управления шиной. Такой режим используется в мультипроцессорных системах и в сложных вычислительных устройствах, в частности, в компьютере IBM PC/XT.

Интересно организована память: хранение 16-разрядных слов осуществляется в виде отдельных байтов, причем байты, передающиеся по восьми младшим линиям шины данных (D7-D0), собраны в банк 0, а передаваемые по восьми старшим линиям – в банк 1. Объем каждого банка составляет 512 Кбайт. Таким образом, нечетные байты хранятся в банке 1, а четные– в банке 0. Выбор банка осуществляется с помощью младшего адреса и сигнала управления старшими разрядами шины данных.

Еще одна важная особенность –возможность обработки 256 типов прерываний (от 0 до 255), в том числе есть прерывания, определяемые пользователем, и пошаговые прерывания.

Микропроцессор Intel 8086 приспособлен для работы с несколькими процессорами в одной системе, причем возможно использование как независимых процессоров, так и сопроцессоров. Отличие заключается в том, что независимый процессор выполняет свою собственную последовательность команд, а сопроцессор следит за потоком команд центрального процессора и выделяет из него «свои» команды, расширяя набор команд основного процессора и улучшая таким образом характеристики системы. Для поддержки этих режимов используются команды ESC, LOCK и XCHG, а также специальные управляющие сигналы, позволяющие разрешать конфликты доступа к общим ресурсам.

Внешние шины адреса и данных в 8086 объединены, и поэтому наличие на шине в данный момент времени информации или адреса определяется порядковым номером такта внутри цикла. Процессор ориентирован на параллельное выполнение команды и выборки следующей команды. В целом выполнение команды происходит примерно так же, как и в 8080. Команда выбирается из памяти и принимается микропроцессором в свободный регистр очереди команд, причем в то же самое время выполняется предыдущая команда. Конвейеризация команд позволяет значительно повысить быстродействие системы. При выполнении команд проверяются состояния входов запросов прерываний и захвата шины, и при необходимости выполняются соответствующие действия.

Микропроцессор i8086 состоит из трех основных частей: устройства сопряжения шины, устройства обработки и устройства управления и синхронизации.

Устройство сопряжения шины состоит из шести 8-разрядных регистров очереди команд, четырех 16-разрядных регистров адреса команды, 16-разрядного регистра команды и 16-разрядного сумматора адреса (см. рис. 2.12). Оно выполняет следующие функции: выбирает команды из памяти и записывает их в регистр очереди команд, вычисляет и формирует физический адрес, читает операнды из памяти или из регистров и записывает результат выполнения команд в память или в регистры.

А0/D0-A15/D15

16

А16-А19

4

Устройство управления и синхронизации

Буфер шины адреса/данных

Внутренняя шина команд

Внутренняя шина

Регистры очереди команд (8-разрядные)

1

2

3

4

5

6

16

Сумматор адреса

CS

DS

SS

ES

IP

Регистр обмена

Внутренняя шина данных

16

Микропрограммное устройство управления

Арифметико-

логическое устройство

Регистр флагов

AH AL

BH BL

CH CL

DH DL

SP

BP

SI

DI

A

B

C

D

Рис. 4.23. Внутренняя структура микропроцессора 8086

Устройство обработки преобразует данные. Команда из очереди команд по запросу устройства обработки поступает на внутреннюю шину команд, а с нее на микропрограммное устройство управления, декодирующее ее и генерирующее соответствующие последовательности микрокоманд, необходимые для выполнения текущей операции. В отличие от первых микропроцессоров, устройство обработки в 8086 не связано с внешней шиной, а обменивается с ней информацией через регистр обмена устройства сопряжения шины.

Устройство обработки содержит 16-разрядное арифметико-логическое устройство, восемь 16-разрядных регистров общего назначения и 16-разрядный регистр флагов. Регистры могут использоваться как 16-разрядные или как пары 8-разрядных (при этом их количество удваивается).

2.6. Архитектура микропроцессора Intel80286

Кристалл 80286 представляет для читателя интерес прежде всего потому, что является, пожалуй, наиболее распространенным микропроцессором из применяющихся в персональных компьютерах. Как и его предшественник – 8086 – он имеет 16-разрядные шины данных и адреса и самым характерным его отличием можно считать, помимо большей тактовой частоты, возможность работы в режиме виртуальной адресации (адресация памяти объемом более 1 Мбайта), речь о котором пойдет ниже.

**Регистры**

Как и любой процессор, Intel-80286 содержит некоторое количество ячеек памяти быстрого доступа, называемых регистрами. В состав i802286 входят три набора по четыре регистра и один специальный регистр – указатель команды.

***Регистры общего назначения***

Первый набор включает в себя регистры общего назначения или РОН, необходимые для временного хранения тех операндов и результатов вычислений, доступ к которым постоянно повторяется в процессе выполнения программы. Использование РОН в подобных случаях существенно ускоряет работу системы за счет сокращения времени чтения/записи и пересылки данных из ОЗУ. Всего регистров общего назначения четыре, они, разумеется, 16-разрядные, но могут использоваться и как 8-разрядные (однобайтные), при этом их количество удваивается.

Функции всех РОН, в основном, идентичны, но в некоторых случаях архитектура предполагает их строгую специализацию. Например, при выполнении команд обработки строк и циклов, в одном из регистров должно храниться число, равное количеству итераций. Этот регистр выполняет роль счетчика (counter) и носит название CX. Остальные регистры выполняют функции аккумулятора (AX), базы (BX) и ячейки временного хранения данных (DX). Как мы уже знаем, каждый регистр из числа РОН может быть разделен на два однобайтных, один из которых (0-7) называется младшим (Low), а другой (7-15) – старшим (High). В соответствии с этим, каждый 8-разрядный регистр получил свое название: младшие именуются AL, BL, CL, DL, а старшие – AН, BН, CН и DH (рис. 2.13).

AH AL

BH BL

CH CL

DH DL

AX

BX

CX

DX

Аккумулятор

База

Счетчик

Данные

7

7

0

0

Рис. 4.24 Регистры общего назначения

Перед тем как познакомиться с назначением и функциями остальных наборов регистров, разберемся, каким образом процессору с 16-разрядной шиной адреса удается работать с памятью объемом в 1 Мбайт.

***Режим реального адреса***. Адресная шина процессора 80286 имеет ширину 16 бит, к тому же известно, что максимальное двоичное число длиной в два байта равно 216 или 64 Кбайт и, если адрес задается таким числом, то, вроде бы, пространство ОЗУ, с которым может работать процессор, не должно превышать 64 Кбайт. С другой стороны, 1 Мбайт памяти можно адресовать с помощью двоичного числа длиной 20 бит (220). Как быть?

Можно, например, воспользоваться двойным адресом, ведь в повседневной жизни нам приходится постоянно сталкиваться с многоступенчатыми адресами: мы пишем на почтовом конверте сначала название города, потом – улицы, дома и т.д. (Предположим на мгновенье, что все квартиры в СССР пронумерованы последовательно, каково придется почте в такой ситуации?) Разработчики 80286 решили проблему подобным же образом: полный адрес ячейки памяти состоит из комбинации двух 16-разрядных чисел, причем одно из них предназначили для адресации внутри некоторой области ОЗУ размером 64 Кбайта, а второе – для локализации этой области во всем пространстве ОЗУ. Область, внутри которой происходит адресация, называется сегментом, а адрес внутри сегмента – внутрисегментным смещением. Адрес, локализующий положение сегмента в оперативной памяти, содержится в одном из специальных сегментных регистров процессора, но он тоже 16-разрядный. Для того, чтобы при помощи этого адреса можно было перекрыть все пространство ОЗУ, со стороны младшего байта его дополняют четырьмя нулями. Например, если содержимое сегментного регистра: 0001.1101.1000.1111 (или 1D8F16), то адрес начала соответствующего сегмента будет равен: 0001.1101.1000.1111.0000 (или 1D8F016). Таким образом можно искусственно разделить всю память на сегменты, начинающиеся по адресам, кратным 1610. Предположим, что внутрисегментное смещение нашей ячейки задано числом 1001.1011.0010.0101 или 9В2516, в этом случае ее реальный адрес будет равен сумме адреса сегмента и внутрисегментного смещения: 1D8F016+9B2516 = 2701516. Схема такого формированияреального адреса будет рассмотрена в следующей лекции.

Выполняемая программа может обращаться к любому из четырех сегментов, именуемых: текущий сегмент кода (то есть программы), текущий сегмент данных, текущий сегмент стека и текущий дополнительный сегмент.

Теперь вернемся к регистрам.

***Указательные и индексные регистры***

Второй набор состоит из четырех 16-разрядных регистров, которые, в основном, используются для хранения внутрисегментных смещений. При выполнении многих команд функции каждого из данных регистров строго определены.

Регистры SI (индекс источника) и DI (индекс приемника) называются указательными и содержат смещения в текущем сегменте данных. Регистры SP (указатель стека) и ВР (указатель базы) называются базовыми и содержат смещения в текущем сегменте стека. Для тех, кто забыл или не знает, следует напомнить, что стек – это способ организации работы с оперативной памятью по принципу LIFO (LastIn-FirstOut), что в переводе означает: «последним вошел – первым вышел», то есть слово данных, помещенное в стек последним, будет извлечено оттуда в первую очередь. Само слово стек произошло от английского stack–скирда. И действительно, область памяти, организованную в виде стека, можно сравнить со скирдой сена: последние, уложенные сверху снопы, будут использованы первыми – никому не придет в голову выгребать сено из середины. Для работы со стеком необходимо знать две величины: адрес дна и адрес вершины стека. Если адрес дна – число фиксированное, то адрес вершины зависит от того, сколько байтов данных содержится в стеке. В нашем случае адрес вершины находится в регистре SP.

Регистры SP и BP содержат указатели стека и базы соответственно, регистры SI и DI – смещения, соответственно, источника и приемника при выполнении команд обработки строк MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW и команды LOOP.

***Сегментные регистры***

Если мы вспомним, что программа в любой момент может обратиться к одному из четырех сегментов: к текущему сегменту кода, данных, стека или к дополнительному (сегменту данных), то нас вряд ли удивит, что в состав процессора входят четыре 16-разрядных регистра, являющихся указателями адресов текущих Сегментов. Их функции строго дифференцированы, а потому каждый регистр имеет свою «профессию»: CS определяет сегмент кода, DS – сегмент данных, SS – сегмент стека и ES – дополнительный сегмент.

Теперь для того, чтобы, к примеру, произвести выборку слова данных из стека, программе достаточно обратиться к регистрам SS и SP, сложить находящиеся в них числа по уже известному нам правилу и в качестве результата получить реальный адрес вершины стека.

***Флажки***

Совокупность значений флажков и определяет состояние процессора во время его работы. В самом общем случае слово состояния – это двоичное число, каждый бит которого отражает строго определенный параметр состояния устройства. Что касается 80286, то здесь биты слова состояния называются флажками, всего их девять, причем шесть из них регистрируют состояние процессора, а три применяются для управления его работой.

C

T

S

NT

IOPL

O

D

I

Z

A

P

IP

Флажки

Указатель команды

15

0

Рис.4.25 Указатель команды и флажки

Верхний регистр IP является указателем команды, нижний – регистром флажков. К флажкам состояния относятся: флажок переноса CF (имеет значение равное 1 при переносе из старшего бита), флажок вспомогательного переноса AF (индицирует перенос из младших 4-х бит), флажок переполнения OF (устанавливается равным единице при выходе знакового результата за границу диапазона), флажок нуля ZF (фиксирует нулевой результат выполнения команды), флажок знака SF (фиксирует отрицательный результат выполнения команды) флажок четности PF (фиксирует четное число единиц в последнем байте, полученном в результате выполнения команды).

К флажкам управления относятся: флажок направления DF (указывает направление прохождения строк в строковых командах), флажок разрешения прерывания IF (разрешает или запрещает прерывание по входу INTR), флажок трассировки TF (переводит процессор в пошаговый режим).

Размещение флажков в разрядах регистра слова состояния будет также рассмотрено в следующей лекции.

5. УРОВЕНЬ АРХИТЕКТУРЫ НАБОРА КОМАНД

5.1. Система команд

5.1.1. Классификация команд

Проектирование системы команд оказывает влияние на структуру ЭВМ. Оптимальную систему команд иногда определяют как совокупность команд, которая удовлетворяет требованиям проблемно-ориентированных применений таким образом, что избыточность аппаратных и аппаратно-программных средств на реализацию редко используемых команд оказывается минимальной. В различных программах ЭВМ частота появления команд различна; например, по данным фирмы DEC в программах для ЭВМ семейства PDP-11 наиболее часто встречается команда передачи MOV(B), на ее долю приходится приблизительно 32% всех команд в типичных программах. Систему команд следует выбирать таким образом, чтобы затраты на редко используемые команды были минимальными.

При наличии статистических данных можно разработать (выбрать) ЭВМ с эффективной системой команд. Одним из подходов к достижению данной цели является разработка команд длиной в одно слово и кодирование их таким образом, чтобы разряды таких коротких команд использовать оптимально, что позволит сократить время реализации программы и ее длину.

Другим подходом к оптимизации системы команд является использование микроинструкций. В этом случае отдельные биты или группы бит команды используются для кодирования нескольких элементарных операций, которые выполняются в одном командном цикле. Эти элементарные операции не требуют обращения к памяти, а последовательность их реализации определяется аппаратной логикой.

Сокращение времени выполнения программ и емкости памяти достигается за счет увеличения сложности логики управления.

Важной характеристикой команды является ее формат, определяющий структурные элементы команды, каждый из которых интерпретируется определенным образом при ее выполнении. Среди таких элементов (полей) команды выделяют следующие: код операции, определяющий выполняемое действие; адрес ячейки памяти, регистра процессора, внешнего устройства; режим адресации; операнд при использовании непосредственной адресации; код анализируемых признаков для команд условного перехода.

Классификация команд по основным признакам представлена на рис. 2.1. Важнейшим структурным элементом формата любой команды является код операции (КОП), определяющийдействие, которое должно быть выполнено. Большое число КОП в процессоре очень важно, так как аппаратная реализация команд экономит память и время. Но при выборе ЭВМ необходимо концентрировать внимание на полноте операций с конкретными типами данных, а не только на числе команд, на доступных режимах адресации. Число бит, отводимое подКОП, является функцией полного набора реализуемых команд.

Команды

По функциональному числу

По числу

адресов

По способу кодирования

команд

По длине команды

По способу

адресации

Передача данных

Обработка данных

Передача управления

Дополнительные

Безадресные

(нуль адресные)

Одноадресные

Двухадресные

С фиксированным полем КОП

С расширяющимся полем КОП

Длиной в одно слово(байт)

Длиной в два слова(байта)

Длиной в три слова(байта)

Рис. 5.1. Классификация команд.

При использовании фиксированного числа бит под КОП для кодирования всех *m* команд необходимо в поле КОП выделитьn двоичных разрядов, чтобы 2*n*=*m*. Однако, учитывая ограниченную длину слова мини- и микроЭВМ, различное функциональное назначение команд, источники и приемники результатов операций, а также то, что не все команды содержат адресную часть для обращения к памяти и периферийным устройствам, в малых ЭВМ для кодирования команд широко используется принцип кодирования с переменным числом бит под поле КОП для различных групп команд.

В некоторых командах необходим только один операнд и они называются однооперандными (или одноадресными) командами в отличие от двухоперандных (или двухадресных), в которых требуются два операнда. При наличии двух операндов командой обычно изменяется только один из них. Так как информация берется только из одной ячейки, эту ячейку называют источником; ячейку, содержимое которой изменяется, называют приемником.

По форматам команд можно судить о возможностях ЭВМ.

5.1.2. Режимы адресации

Для взаимодействия с различными модулями в ЭВМ должны быть средства идентификации ячеек внешней памяти, ячеек внутренней памяти, регистров МП и регистров устройств ввода/вывода. Поэтому каждой из запоминающих ячеек присваивается адрес, т.е. однозначная комбинация бит. Количество бит определяет число идентифицируемых ячеек. Обычно ЭВМ имеет различные адресные пространства памяти и регистров МП, а иногда –отдельные адресные пространства регистров устройств ввода/вывода и внутренней памяти. Кроме того, память хранит как данные, так и команды. Поэтому для ЭВМ разработано множество способов обращения к памяти, называемых режимами адресации.

Режим адресации памяти – это процедура или схема преобразования адресной информации об операнде в его исполнительный адрес.

Все способы адресации памяти можно разделить на:

1) прямой, когда исполнительный адрес берется непосредственно из команды или вычисляется с использованием значения, указанного в команде, и содержимого какого-либо регистра (прямая адресация, регистровая, базовая, индексная и т.д.);   
2) косвенный, который предполагает, что в команде содержится значение косвенного адреса, т.е. адреса ячейки памяти, в которой находится окончательный исполнительный адрес (косвенная адресация).

5.1.3. Адресация операндов

Большая часть команд процессора работает с кодами данных (операндами). Одни команды требуют входных операндов (одного или двух), другие выдают выходные операнды (чаще один операнд). Входные операнды называются еще операндами-источниками, а выходные называются операндами-приемниками. Все эти коды операндов (входные и выходные) должны где-то располагаться. Они могут находиться во внутренних регистрах процессора (наиболее удобный и быстрый вариант). Они могут располагаться в системной памяти (самый распространенный вариант). Наконец, они могут находиться в устройствах ввода/вывода (наиболее редкий случай). Определение места положения операндов производится кодом команды. Причем существуют разные методы, с помощью которых код команды может определить, откуда брать входной операнд и куда помещать выходной операнд. Эти методы называются **методами адресации**. Эффективность выбранных *методов адресации* во многом определяет эффективность работы всего процессора в целом.

Методы адресации

Количество *методов адресации* в различных процессорах может быть от 4 до 16. Рассмотрим несколько типичных *методов адресации операндов*, используемых сейчас в большинстве микропроцессоров.

***Непосредственная адресация*** (рис. 5.2.) предполагает, что *операнд* (входной) находится в памяти непосредственно за кодом команды. *Операнд* обычно представляет собой константу, которую надо куда-то переслать, к чему-то прибавить и т.д. Например, команда может состоять в том, чтобы прибавить число 6 к содержимому какого-то внутреннего регистра процессора. Это число 6 будет располагаться в памяти, внутри программы в адресе, следующем за кодом данной команды сложения.

Программа

Команда

Операнд

Память

Рис.5.2. Непосредственная адресация

***Прямая (она же абсолютная) адресация*** (рис. 5.3.) предполагает, что *операнд* (входной или выходной) находится в памяти по адресу, код которого находится внутри программы сразу же за кодом команды. Например, команда может состоять в том, чтобы очистить (сделать нулевым) содержимое ячейки памяти с адресом 1000000. Код этого адреса 1000000 будет располагаться в памяти, внутри программы в следующем адресе за кодом данной команды очистки.

Память

Команда

Операнд

Программа

Операнд

Данные

Рис. 5.3.  Прямая адресация.

***Регистровая адресация*** (рис. 5.4) предполагает, что *операнд* (входной или выходной) находится во внутреннем регистре процессора. Например, команда может состоять в том, чтобы переслать число из нулевого регистра в первый. Номера обоих регистров (0 и 1) будут определяться кодом *команды пересылки*.

Косвенно-регистровая (она же косвенная) адресация предполагает, что во внутреннем регистре процессора находится не сам *операнд*, а его адрес в памяти (рис. 5.5). Например, команда может состоять в том, чтобы очистить ячейку памяти с адресом, находящимся в нулевом регистре. Номер этого регистра (0) будет определяться кодом команды очистки.

Память

Регистр

Команда

Программа

Операнд

Рис. 5.4.  Регистровая адресация.

Память

Регистр

Команда

Операнд

Программа

Операнд

Данные

Рис. 5.5.  Косвенная адресация.

Реже встречаются еще два *метода адресации*.

***Автоинкрементная адресация*** очень близка к косвенной адресации, но отличается от нее тем, что после выполнения команды содержимое используемого регистра увеличивается на единицу или на два. Этот *метод адресации* очень удобен, например, при последовательной обработке кодов из массива данных, находящегося в памяти. После обработки какого-то кода адрес в регистре будет указывать уже на следующий код из массива. При использовании косвенной адресации в данном случае пришлось бы увеличивать содержимое этого регистра отдельной командой.

***Автодекрементная адресация*** работает похоже на автоинкрементную, но только содержимое выбранного регистра уменьшается на единицу или на два перед выполнением команды. Эта адресация также удобна при обработке массивов данных. Совместное использование автоинкрементной и автодекрементной адресаций позволяет организовать память стекового типа.

Из других распространенных *методов адресации* можно упомянуть об индексных методах, которые предполагают для вычисления адреса *операнда* прибавление к содержимому регистра заданной константы (индекса). Код этой константы располагается в памяти непосредственно за кодом команды.

Отметим, что выбор того или иного *метода адресации* в значительной степени определяет время выполнения команды. Самая быстрая адресация – это регистровая, так как она не требует дополнительных циклов обмена по магистрали. Если же адресация требует обращения к памяти, то время выполнения команды будет увеличиваться за счет длительности необходимых циклов обращения к памяти. Понятно, что чем больше внутренних регистров у процессора, тем чаще и свободнее можно применять регистровую адресацию, и тем быстрее будет работать система в целом.

Сегментирование памяти.

Говоря об адресации, нельзя обойти вопрос о *сегментировании* памяти, применяемой в некоторых процессорах, например в процессорах IBM PC-совместимых персональных компьютеров.

В процессоре Intel 8086 *сегментирование* памяти организовано следующим образом.

Вся память системы представляется не в виде непрерывного пространства, а в виде нескольких кусков – сегментов заданного размера (по 64 Кбайта), положение которых в пространстве памяти можно изменять программным путем.

Для хранения кодов адресов памяти используются не отдельные регистры, а пары регистров:

* сегментный регистр определяет адрес начала сегмента (то есть положение сегмента в памяти);
* регистр указателя (регистр смещения) определяет положение рабочего адреса внутри сегмента.

При этом физический 20-разрядный адрес памяти, выставляемый на внешнюю *шину адреса*, образуется так, как показано на рис. 3.6, то есть путем сложения смещения и адреса сегмента со сдвигом на 4 бита. Положение этого адреса в памяти показано на рис. 3.7.

Сегмент может начинаться только на 16-байтной границе памяти (так как адрес начала сегмента, по сути, имеет четыре младших нулевых разряда, как видно из рис. 5.6), то есть с адреса, кратного 16.

Эти допустимые границы сегментов называются границами параграфов.

Отметим, что введение *сегментирования*, прежде всего, связано с тем, что внутренние регистры процессора 16-разрядные, а физический адрес памяти 20-разрядный (16-разрядный адрес позволяет использовать память только в 64 Кбайт, что явно недостаточно). В появившемся в то же время процессоре MC68000 фирмы Motorola внутренние регистры 32-разрядные, поэтому там проблемы *сегментирования* памяти не возникает.

16 бит

20 бит

4 бита

16 бит

Исполнительный адрес

(смещение)

Адрес сегмента

Физический адрес памяти

Рис. 5.6.  Формирование физического адреса памяти из адреса сегмента и смещения

15АВЕ

Смещение (ЗАВЕ)

12000

.

.

.

.

.

.

Память

Адрес сегмента

Размер сегмента

(10000 байт = 64 Кбайт)

Физический адрес

Адреса 00000

21FFF

22000

Рис. 5.7.  Физический адрес в сегменте (все коды – шестнадцатеричные)

Применяются и более сложные методы *сегментирования* памяти. Например, в процессоре *Intel 80286* в так называемом защищенном режиме адрес памяти вычисляется в соответствии с [рис. 3.](http://www.intuit.ru/department/hardware/mpbasics/4/#image.3.7)8.

В *сегментном регистре* в данном случае хранится не базовый (начальный) адрес сегментов, а коды селекторов, определяющие адреса в памяти, по которым хранятся дескрипторы (то есть описатели) сегментов. Область памяти с дескрипторами называется таблицей дескрипторов. Каждый дескриптор сегмента содержит базовый адрес сегмента, размер сегмента (от 1 до 64 Кбайт) и его атрибуты. Базовый адрес сегмента имеет разрядность 24 бит, что обеспечивает адресацию 16 Мбайт физической памяти.

Процессор

Сегментный регистр

15

0

Селектор

15

0

Смещение

Сумматор физического адреса

23

0

Базовый адрес сегмента

Память

Операнд

Операнд

Сегмент

Дескриптор

сегмента

Таблица

дескрипторов

сегмента

Рис. 5.8.  Адресация памяти в защищенном режиме процессора Intel 80286

Таким образом, на *сумматор*, вычисляющий физический адрес памяти, подается не содержимое сегментного регистра, как в предыдущем случае, а базовый адрес сегмента из таблицы дескрипторов.

Еще более сложный *метод адресации* памяти с *сегментированием* использован в процессоре *Intel 80386* и в более поздних моделях процессоров фирмы Intel. Этот метод иллюстрируется рис. 3.9.

Индекс

дескриптора

32

Эффективный

адрес

Вычисление эффективного адреса

INDEX

SCALE (1,2,4,8)

BASE

DISP

32

Линейный

адрес

32

Физический

адрес

Память

Блок страничной переадресации

Блок сегментации

Сегментный регистр

Рис. 5.9.  Формирование физического адреса памяти процессора 80386 в защищенном режиме.

Адрес памяти (физический адрес) вычисляется в три этапа. Сначала вычисляется так называемый **эффективный адрес** (32-разрядный) путем суммирования трех компонентов: базы, индекса и смещения (Base, Index, *Displacement*), причем возможно умножение индекса на масштаб (Scale). Эти компоненты имеют следующий смысл:

* смещение– это 8-, 16- или 32-разрядное число, включенное в команду;
* база – это содержимое базового регистра процессора. Обычно оно используется для указания на начало некоторого массива;
* индекс – это содержимое индексного регистра процессора. Обычно оно используется для выбора одного из элементов массива;
* масштаб – это множитель (он может быть равен 1, 2, 4 или 8), указанный в коде команды, на который перед суммированием с другими компонентами умножается индекс. Он используется для указания размера элемента массива.,

Затем специальный блок *сегментации* вычисляет 32-разрядный линейный адрес, который представляет собой сумму базового адреса сегмента из сегментного регистра с эффективным адресом. Наконец, физический 32-битный адрес памяти образуется путем преобразования линейного адреса блоком страничной переадресации, который осуществляет перевод линейного адреса в*физический страницами* по

4 Кбайта.

В любом случае *сегментирование* позволяет выделить в памяти один или несколько сегментов для данных и один или несколько сегментов для программ. Переход от одного сегмента к другому сводится всего лишь к изменению содержимого сегментного регистра. Иногда это бывает очень удобно. Но для программиста работать с сегментированной памятью обычно сложнее, чем с непрерывной, несегментированной памятью, так как приходится следить за границами сегментов, за их описанием, переключением и т.д.

Адресация байтов и слов.

Многие процессоры, имеющие разрядность 16 или 32, способны адресовать не только целое слово в памяти (16-разрядное или 32-разрядное), но и отдельные байты. Каждому байту в каждом слове при этом отводится свой адрес.

Так, в случае 16-разрядных процессоров все слова в памяти (16-разрядные) имеют четные адреса. А байты, входящие в эти слова, могут иметь как четные адреса, так и нечетные.

Например, пусть 16-разрядная ячейка памяти имеет адрес 23420, и в ней хранится код 2А5Е (рис. 3.10).

.

.

16 бит

Память

.

.

2A 5E

48 7F

Старший байт

(адрес 23421)

Старший байт

(адрес 23423)

Младший байт

(адрес 23420)

Младший байт

(адрес 23422)

Слово

(адрес 23422)

Слово

(адрес 23420)

Рис. 5.10. Адресация слов и байтов.

При обращении к целому слову (с содержимым 2А5Е) процессор выставляет адрес 23420. При обращении к младшему байту этой ячейки (с содержимым 5Е) процессор выставляет тот же самый адрес 23420, но использует команду, адресующую байт, а не слово. При обращении к старшему байту этой же ячейки (с содержимым 2А) процессор выставляет адрес 23421 и использует команду, адресующую байт. Следующая по порядку 16-разрядная ячейка памяти с содержимым 487F будет иметь адрес 23422, то есть опять же четный. Ее байты будут иметь адреса 23422 и 23423.

Для различия байтовых и словных циклов обмена на магистрали в шине управления предусматривается специальный сигнал байтового обмена. Для работы с байтами в систему команд процессора вводятся специальные команды или предусматриваются методы байтовой адресации.

5.1.4 Регистры процессора

Как уже упоминалось, внутренние регистры процессора представляют собой сверхоперативную память небольшого размера, которая предназначена для временного хранения служебной информации или данных. Количество регистров в разных процессорах может быть от 6—8 до нескольких десятков. Регистры могут быть универсальными и специализированными. Специализированные регистры, которые присутствуют в большинстве процессоров, – это *регистр*-*счетчик команд, регистр состояния* (**PSW**), *регистр указателястека*. Остальные регистры процессора могут быть как универсальными, так и специализированными.

Например, в 16-разрядном процессоре Т-11 фирмы DEC было 8 регистров общего назначения (РОН) и один *регистр состояния*. Все регистры имели по 16 разрядов. Из регистров общего назначения один отводился под *счетчик команд*, другой – под указатель стека. Все остальные регистры общего назначения полностью взаимозаменяемы, то есть имеют универсальное назначение, могут хранить как данные, так и адреса (указатели), индексы и т.д. Максимально допустимый объем памяти для данного процессора составлял 64 Кбайт (адрес памяти 16-разрядный).

В 16-разрядном процессоре MC68000 фирмы Motorola было 19 регистров: 16-разрядный *регистр состояния*, 32-разрядный регистр *счетчика команд*, 9 регистров адреса (32-разрядных) и 8 регистров данных (32-разрядных). Два регистра адреса отведены под указатели стека. Максимально допустимый объем адресуемой памяти – 16 Мбайт (внешняя шина адреса 24-разрядная). Все 8 регистров данных взаимозаменяемы, 7 регистров адреса – тоже взаимозаменяемы.

В 16-разрядном процессоре Intel 8086, который стал базовым в линии процессоров, используемых в персональных компьютерах, реализован принципиально другой подход. Каждый регистр этого процессора имеет свое особое назначение, и заменять друг друга регистры могут только частично или же не могут вообще. Остановимся на особенностях этого процессора подробнее.

Процессор 8086 имеет 14 регистров разрядностью по 16 бит. Из них четыре регистра (AX, BX, CX, DX) – это регистры данных, каждый из которых помимо хранения операндов и результатов операций имеет еще и свое специфическое назначение:

• регистр AX – умножение, деление, обмен с устройствами ввода/вывода (команды ввода и вывода);

• регистр BX – базовый регистр в вычислениях адреса;

• регистр CX – счетчик циклов;

• регистр DX –определение адреса ввода/вывода.

Для регистров данных существует возможность раздельного использования обоих байтов (например, для регистра AX они имеют обозначения AL – младший байт и AH — старший байт).

Следующие четыре внутренних регистра процессора — это сегментные регистры, каждый из которых определяет положение одного из рабочих сегментов (рис. 3.11):

• CS (CodeSegment) соответствует сегменту команд, исполняемых в данный момент;

• DS (DataSegment) соответствует сегменту данных, с которыми работает процессор;

• ES (ExtraSegment) соответствует дополнительному сегменту данных;

• SS (StackSegment) соответствует сегменту стека.

**.**

**.**

**.**

(CS)

Сегмент команд

**.**

**.**

**.**

(DS)

Сегмент данных

**.**

**.**

**.**

(ES)

Дополнительный сегмент данных

**.**

**.**

**.**

(SS)

Стек

**.**

**.**

**.**

Рис. 5.11. Сегменты команд, данных и стека в памяти.

В принципе, все эти сегменты могут и перекрываться для оптимального использования пространства памяти. Например, если программа занимает только часть сегмента, то сегмент данных может начинаться сразу после завершения работы программы (с точностью 16 байт), а не после окончания всего сегмента программы.

Следующие пять регистров процессора (SP –StackPointer, BP –BasePointer, SI –SourceIndex, DI –DestinationIndex, IP –InstructionPointer) служат указателями (то есть определяют смещение в пределах сегмента). Например, *счетчик команд* процессора образуется парой регистров CS и IP, а указатель стека – парой регистров SP и SS. Регистры SI, DI используются в строковых операциях, то есть при последовательной обработке нескольких ячеек памяти одной командой.

Последний регистр FLAGS – это регистр состояния процессора (PSW). Из его 16 разрядов используются только девять (рис. 3.12): CF (CarryFlag) – флаг пере-носа при арифметических операциях, PF (ParityFlag) – флаг четности результа-та, AF (AuxiliaryFlag) – флаг дополнительного переноса, ZF (ZeroFlag) — флаг нулевого результата, SF (SignFlag) – флаг знака (совпадает со старшим битом результата), TF (TrapFlag) – флаг пошагового режима (используется при отла-ке), IF (Interrupt-enableFlag) – флаг разрешения аппаратных прерываний, DF (DirectionFlag) – флаг направления при строковых операциях, OF (OverflowFlag) – флаг переполнения.

CF

1514 13 12 11 109 8 7 6 5 4 3 2 1 0

OF DF IF TF SF ZF AF PF CFCFАCF

Рис. 5.12. Регистр состояния процессора 8086.

Биты регистра состояния устанавливаются или очищаются в зависимости от результата исполнения предыдущей команды и используются некоторыми командами процессора. Биты регистра состояния могут также устанавливаться и очищаться специальными командами процессора (о системе команд процессора будет рассказано в следующем разделе).

Во многих процессорах выделяется специальный регистр, называемый **аккумулятором** (то есть накопителем). При этом, как правило, только этот регистр-аккумулятор может участвовать во всех операциях, только через него может производиться взаимодействие с устройствами ввода/вывода. Иногда в него же помещается результат любой выполненной команды (в этом случае говорят даже об «аккумуляторной» архитектуре процессора). Например, в процессоре 8086 регистр данных АХ можно считать своеобразным аккумулятором, так как именно он обязательно участвует в командах умножения и деления, а также только через него можно пересылать данные в устройство ввода/вывода и из устройства ввода/вывода. Выделение специального регистра-аккумулятора упрощает структуру процессора и ускоряет пересылки кодов внутри процессора, но в некоторых случаях замедляет работу системы в целом, так как весь поток информации должен пройти через один регистр-аккумулятор. В случае, когда несколько регистров процессора полностью взаимозаменяемы, таких проблем не возникает.

К данному типу архитектуры относится микропроцессор фирмы Zilog. Процессор Z80 – детище фирмы Zilog помимо расширенной системы команд, одного номинала питания и способности исполнять программы, написанные для i8080, имел архитектурные «изюминки».

Шина адреса (16 бит)

Шина данных (8бит)

Управление (13 линий)

Регистр команды

8 бит

Индексный регистр 1Х

Индексный регистр 1Y

Указатель стека SP

Программный

счетчик PC

16 бит

Основной набор регистров

Аккумулятор A

B

D

H

S

Z

H

PV

N

C

C

E

L

Аккумулятор A

B`

D`

H`

S

Z

H

PV

N

C

C`

E`

L`

АЛУ

8 бит

8 бит

8 бит

8 бит

PSW

Дополнительный набор регистров

Управление шиной

Вектор прерывания 1

Регенерация памяти - R

8 бит

8 бит

Рис. 5.13. Микропроцессор Z80 фирмы Zilog

В дополнение к основному набору РОН, в кристалле был реализован второй комплект аналогичных регистров. Это значительно упрощало работу при вызове подпрограмм или процедур обслуживания прерываний, поскольку программист мог использовать для них альтернативный набор регистров, избегая сохранения в стеке содержимого РОНов для основной программы с помощью операций PUSH. Кроме того, в систему команд был включен ряд специальных инструкций, ориентированных на обработку отдельных битов, а для поддержки регенерации динамической памяти в схему процессора введены соответствующие аппаратные средства. Z80 применялся в машинахSinclairZX, SinclairSpectrum, TandyTRS80.

Предельный вариант – архитектура с адресацией посредством аккумуляторов (меньший набор команд).

МП фирмы Motorola имел ряд существенных преимуществ. Прежде всего, кристалл МС6800 требовал для работы одного номинала питания, а система команд оказалась весьма прозрачной для программиста. Архитектура МП также имела ряд особенностей.

Адрес(16 бит)

Данные (8бит)

Управление (9 линий)

8 бит

АЛУ

Управление шиной

16 бит

16 бит

16 бит

Программный счетчик

Указатель стека

Индексный регистр

H

I

N

Z

V

C

Регистр команды

Состояние процессора

А (8 бит)

А (8 бит)

Аккумуляторы

Рис 5.14. Микропроцессор МС6800 фирмы Motorola.

Микропроцессор МС 6800 содержал два аккумулятора, и результат операции АЛУ мог быть помещен в любой из них. Но самым ценным качеством структуры МС 6800 было автоматическое сохранение в стеке содержимого всех регистров процессора при обработке прерываний (Z80 требовалось для этого несколько команд PUSH). Процедура восстановления РОН из стека тоже выполнялась аппаратно.

1. Стековая архитектура дает возможность создать поле памяти с упорядоченной последовательностью записи и выборки информации.
2. В общем случае команды неявно адресуются к элементу стека, расположенному на его вершине, или к двум верхним элементам стека.
3. Архитектура МП, ориентированная на оперативную память (типа «память-память»), обеспечивает высокую скорость работы и большую информационную емкость рабочих регистров и стека при их организации в оперативной памяти.

Архитектура этого типа не предполагает явного определения аккумулятора, регистров общего назначения или стека; все операнды команд адресуются к области основной памяти.

С точки зрения важности для пользователя-программиста под архитектурой в общем случае понимают совокупность следующих компонентов и характеристик:

* разрядности адресов и данных;
* состава, имен и назначения программно-доступных регистров;
* форматов и системы команд;
* режимов адресации памяти;
* способов машинного представления данных разного типа;
* структуры адресного пространства;
* способа адресации внешних устройств и средств выполнения операций ввода/вывода;
* классов прерываний, особенностей инициирования и обработки прерываний.

5.1.5. Система команд процессора

В общем случае система команд процессора включает в себя следующие четыре основные группы команд:

• команды пересылки данных;

• арифметические;

• логические;

• команды переходов.

**Команды пересылки** данных не требуют выполнения никаких операций над операндами. Операнды просто пересылаются (точнее, копируются) из источника (Source) в приемник (Destination). Источником и приемником могут быть внутренние регистры процессора, ячейки памяти или устройства ввода/вывода. АЛУ в данном случае не используется.

**Арифметические команды** выполняют операции сложения, вычитания, умножения, деления, увеличения на единицу (инкрементирования), уменьшения на единицу (декрементирования) и т.д. Этим командам требуется один или два входных операнда. Формируют команды один выходной операнд.

**Логические команды** производят над операндами логические операции, например, логическое И, *логическое* ИЛИ, исключающее ИЛИ, очистку, инверсию, разнообразные сдвиги (вправо, влево, арифметический сдвиг, циклический сдвиг). Этим командам, как и *арифметическим*, требуется один или два входных операнда, и формируют они один выходной операнд.

Наконец, **команды переходов** предназначены для изменения обычного порядка последовательного выполнения команд. С их помощью организуются переходы на подпрограммы и возвраты из них, всевозможные циклы, ветвления программ, пропуски фрагментов программ и т.д. Команды переходов всегда меняют содержимое счетчика команд. Переходы могут быть условными и безусловными. Именно эти команды позволяют строить сложные алгоритмы обработки информации.

В соответствии с результатом каждой выполненной команды устанавливаются или очищаются биты регистра состояния процессора (PSW). Но надо помнить, что не все команды изменяют все имеющиеся в PSW флаги. Это определяется особенностями каждого конкретного процессора.

У разных процессоров системы команд существенно различаются, но в основе своей они очень похожи. Количество команд у процессоров также различно. Например, у упоминавшегося уже процессора МС68000 всего 61 команда, а у процессора 8086 – 133 команды. У современных мощных процессоров количество команд достигает нескольких сотен. В то же время существуют процессоры с *сокращенным набором команд* (так называемые RISC-процессоры), в которых за счет максимального сокращения количества команд достигается увеличение эффективности и скорости их выполнения.

Рассмотрим теперь особенности четырех выделенных групп команд процессора более подробно.

Команды пересылки данных.

*Команды пересылки* данных занимают очень важное место в системе команд любого процессора. Они выполняют следующие важнейшие функции:

• загрузка (запись) содержимого во внутренние регистры процессора;

• сохранение в памяти содержимого внутренних регистров процессора;

• копирование содержимого из одной области памяти в другую;

• запись в устройства ввода/вывода и чтение из устройств ввода/вывода.

В некоторых процессорах (например, Т-11) все эти функции выполняются одной единственной командой MOV (для байтовых пересылок – MOVB), но с различными *методами адресации* операндов.

В других процессорах помимо команды MOV имеется еще несколько команд для выполнения перечисленных функций. Например, для загрузки регистров могут использоваться команды загрузки, причем для разных регистров – разные команды (их обозначения обычно строятся с использованием слова LOAD – загрузка). Часто выделяются специальные команды для сохранения в стеке и для извлечения из стека (PUSH – сохранить в стеке, POP – извлечь из стека). Эти команды выполняют пересылку с автоинкрементной и с автодекрементной адресацией (даже если эти режимы адресации не предусмотрены в процессоре в явном виде).

Иногда в систему команд вводится специальная команда MOVS для строчной (или цепочечной) пересылки данных (например, в процессоре 8086). Эта команда пересылает не одно слово или байт, а заданное количество слов или байтов (MOVSB), то есть инициирует не один цикл обмена по магистрали, а несколько. При этом адрес памяти, с которым происходит взаимодействие, увеличивается на 1 или на 2 после каждого обращения или же уменьшается на 1 или на 2 после каждого обращения. То есть в неявном виде применяется автоинкрементная или автодекрементная адресация.

В некоторых процессорах (например, в процессоре 8086) специально выделяются функции обмена с устройствами ввода/вывода. Команда IN используется для ввода (чтения) информации из устройства ввода/вывода, а команда OUT используется для вывода (записи) в устройство ввода/вывода. Обмен информацией в этом случае производится между регистром-аккумулятором и устройством ввода/вывода. В более продвинутых процессорах этого же семейства (начиная с процессора 80286) добавлены команды строчного (цепочечного) ввода (команда INS) и строчного вывода (команда OUTS). Эти команды позволяют пересылать целый массив (строку) данных из памяти в устройство ввода/вывода (OUTS) или из устройства ввода/вывода в память (INS). Адрес памяти после каждого обращения увеличивается или уменьшается (как и в случае с командой MOVS).

Также к *командам пересылки* данных относятся команды обмена информацией (их обозначение строится на основе слова Exchange). Может быть предусмотрен обмен информацией между внутренними регистрами, между двумя половинами одного регистра (SWAP) или между регистром и ячейкой памяти.

Арифметические команды

*Арифметические команды* рассматривают коды операндов как числовые двоичные или двоично-десятичные коды. Эти команды могут быть разделены на пять основных групп:

• команды операций с *фиксированной запятой* (сложение, вычитание, умножение, деление);

• операций с *плавающей запятой* (сложение, вычитание, умножение, деление);

• очистки;

• инкремента и декремента;

• сравнения.

Команды операций с *фиксированной запятой* работают с кодами в регистрах процессора или в памяти как с обычными двоичными кодами. Команда сложения (ADD) вычисляет сумму двух кодов. Команда вычитания (SUB) вычисляет разность двух кодов. Команда умножения (MUL) вычисляет произведение двух кодов (разрядность результата вдвое больше разрядности сомножителей). Команда деления (DIV) вычисляет частное от деления одного кода на другой. Причем все эти команды могут работать как с числами со знаком, так и с числами без знака.

Команды операций с *плавающей запятой* (точкой) используют формат представления чисел с порядком и *мантиссой* (обычно эти числа занимают две последовательные ячейки памяти). В современных мощных процессорах набор команд с *плавающей запятой* не ограничивается только четырьмя арифметическими действиями, а содержит и множество других более сложных команд, например, вычисление тригонометрических функций, логарифмических функций, а также сложных функций, необходимых при обработке звука и изображения.

Команды очистки (CLR) предназначены для записи нулевого кода в регистр или ячейку памяти. Эти команды могут быть заменены *командами пересылки* нулевого кода, но специальные команды очистки обычно выполняются быстрее, чем *команды пересылки*. Команды очистки иногда относят к группе *логических команд*, но суть их от этого не меняется.

Команды инкремента (увеличения на единицу, INC) и декремента (уменьшения на единицу, DEC) также бывают очень удобны. Их можно в принципе заменить командами суммирования с единицей или вычитания единицы, но инкремент и декремент выполняются быстрее, чем суммирование и вычитание. Эти команды требуют одного входного операнда, который одновременно является и выходным операндом.

Наконец, команда сравнения (обозначается CMP) предназначена для сравнения двух входных операндов. По сути, она вычисляет разность этих двух операндов, но выходного операнда не формирует, а всего лишь изменяет биты в регистре состояния процессора (PSW) по результату этого вычитания. Следующая за командой сравнения команда (обычно это *команда перехода*) будет анализировать биты в регистре состояния процессора и выполнять действия в зависимости от их значений (о *командах перехода* речь идет в разделе 3.3.4). В некоторых процессорах предусмотрены команды цепочечного сравнения двух последовательностей операндов, находящихся в памяти.

Логические команды.

*Логические команды* выполняют над операндами логические (побитовые) операции, то есть они рассматривают коды операндов не как единое число, а как набор отдельных битов. Этим они отличаются от *арифметических команд*. *Логические команды* выполняют следующие основные операции:

• логическое И, логическое ИЛИ, сложение по модулю 2 (исключающее ИЛИ);

• логические, арифметические и циклические сдвиги;

• проверка битов и операндов;

• установка и очистка битов (флагов) регистра состояния процессора (PSW).

Команды логических операций позволяют побитно вычислять основные логические функции от двух входных операндов. Кроме того, операция И (AND) используется для принудительной очистки заданных битов (в качестве одного из операндов при этом используется код маски, в котором разряды, требующие очистки, установлены в нуль). Операция ИЛИ (OR) применяется для принудительной установки заданных битов (в качестве одного из операндов при этом используется код маски, в котором разряды, требующие установки в единицу, равны единице). Операция «исключающее ИЛИ» (XOR) используется для инверсии заданных битов (в качестве одного из операндов при этом применяется код маски, в котором биты, подлежащие инверсии, установлены в единицу). Команды требуют двух входных операндов и формируют один выходной операнд.

Команды сдвигов позволяют побитно сдвигать код операнда вправо (в сторону младших разрядов) или влево (в сторону старших разрядов). Тип сдвига (логический, арифметический или циклический) определяет, каково будет новое значение старшего бита (при сдвиге вправо) или младшего бита (при сдвиге влево), а также определяет, будет ли где-то сохранено прежнее значение старшего бита (при сдвиге влево) или младшего бита (при сдвиге вправо). Например, при логическом сдвиге вправо в старшем разряде кода операнда устанавливается нуль, а младший разряд записывается в качестве флага переноса в регистр состояния процессора. А при арифметическом сдвиге вправо значение старшего разряда сохраняется прежним (нулем или единицей), младший разряд также записывается в качестве флага переноса.

Циклические сдвиги позволяют сдвигать биты кода операнда по кругу (по часовой стрелке при сдвиге вправо или против часовой стрелки при сдвиге влево). При этом в кольцо сдвига может входить или не входить флаг переноса. В бит флага переноса (если он используется) записывается значение старшего бита при циклическом сдвиге влево и младшего бита при циклическом сдвиге вправо. Соответственно, значение бита флага переноса будет переписываться в младший разряд при циклическом сдвиге влево и в старший разряд при циклическом сдвиге вправо.

Для примера на рис. 5.13 показаны действия, выполняемые командами сдвигов вправо.

Команды проверки битов и операндов предназначены для установки или очистки битов *регистра состояния* процессора в зависимости от значения выбранных битов или всего операнда в целом. Выходного операнда команды не формируют. Команда проверки операнда (TST) проверяет весь код операнда в целом на равенство нулю и на знак (на значение старшего бита), она требует только одного входного операнда. Команда проверки бита (BIT) проверяет только отдельные биты, для выбора которых в качестве второго операнда используется код маски. В коде маски проверяемым битам основного операнда должны соответствовать единичные разряды.

Циклический сдвиг вправо

через перенос

Циклический сдвиг вправо

Арифметический сдвиг вправо

Логический сдвиг вправо

Флаг переноса

Мл.

Ст.

Операнд

Флаг переноса

Мл.

Ст.

Операнд

Флаг переноса

Мл.

Ст.

Операнд

Флаг переноса

Мл.

Ст.

Операнд

Рис. 5.13. Команды сдвигов вправо.

Наконец, команды установки и очистки битов *регистра состояния* процессора (то есть флагов) позволяют установить или очистить любой флаг, что бывает очень удобно. Каждому флагу обычно соответствуют две команды, одна из которых устанавливает его в единицу, а другая сбрасывает в нуль. Например, флагу переноса C (от Carry) будут соответствовать команды CLC (очистка) и SEC или STC (установка).

Команды переходов.

*Команды переходов* предназначены для организации всевозможных циклов, *ветвлений*, вызовов *подпрограмм* и т.д., то есть они нарушают последовательный ход выполнения программы. Эти команды записывают в регистр-*счетчик команд* новое значение и тем самым вызывают переход процессора не к следующей по порядку команде, а к любой другой команде в памяти программ. Некоторые *команды переходов* предусматривают в дальнейшем возврат назад, в точку, из которой был сделан переход, другие не предусматривают этого. Если возврат предусмотрен, то текущие параметры процессора сохраняются в стеке. Если возврат не предусмотрен, то текущие параметры процессора не сохраняются.

*Команды переходов* без возврата делятся на две группы:

• команды *безусловных переходов*;

• команды *условных переходов.*

В обозначениях этих команд используются слова Branch (ветвление) и Jump (прыжок).

Команды безусловных переходов вызывают переход в новый адрес независимо ни от чего. Они могут вызывать переход на указанную величину смещения (вперед или назад) или же на указанный адрес памяти. Величина смещения или новое значение адреса указываются в качестве входного операнда.

Команды условных переходов вызывают переход не всегда, а только при выполнении заданных условий. В качестве таких условий обычно выступают значения флагов в регистре состояния процессора (PSW). То есть условием перехода является результат предыдущей операции, меняющей значения флагов. Всего таких условий перехода может быть от 4 до 16. Несколько примеров команд условных переходов:

• переход, если равно нулю;

• если не равно нулю;

• если есть переполнение;

• если нет переполнения;

• если больше нуля;

• если меньше или равно нулю.

Если условие перехода выполняется, то производится загрузка в регистр-счетчик команд нового значения. Если же условие перехода не выполняется, счетчик команд просто наращивается, и процессор выбирает и выполняет следующую по порядку команду.

Специально для проверки условий перехода применяется команда сравнения (CMP), предшествующая команде условного перехода (или даже нескольким командам условных переходов). Но флаги могут устанавливаться и любой другой командой, например командой пересылки данных, любой арифметической или логической командой. Отметим, что сами команды переходов флаги не меняют, что как раз и позволяет ставить несколько команд переходов одну за другой.

Совместное использование нескольких команд условных и безусловных переходов позволяет процессору выполнять разветвленные алгоритмы любой сложности. Для примера на рис. 3.14 показано разветвление программы на две ветки с последующим соединением, а на рис. 3.15– разветвление на три ветки с последующим соединением.

Команды переходов с дальнейшим возвратом в точку, из которой был произведен переход, применяются для выполнения подпрограмм, то есть вспомогательных программ. Эти команды называются также командами вызова подпрограмм (распространенное название – CALL). Использование подпрограмм позволяет упростить структуру основной программы, сделать ее более логичной, гибкой, легкой для написания и отладки. В то же время надо учитывать, что широкое использование подпрограмм, как правило, увеличивает время выполнения программы.

нет

да

Программа 1

А + В А

Программа 3

Программа 2

А>0

Память

Сложение

Переход если больше 0

Начало программы 1

.

.

Безусловный переход

Начало программы 2

.

.

Конец программы 2

Начало программы 3

.

.

Рис. 5.14. Реализация разветвления на две ветки.

Память

Сложение

Переход, если равно 0

Переход если больше 0

Начало программы 3

.

.

Безусловный переход

Начало программы 1

.

.

Безусловный переход

Начало программы 2

.

.

Конец программы 2

Начало программы 4

нет

да

нет

Программа 1

А + В А

Программа 3

Программа 2

А=0

А>0

Программа 4

Рис. 5.15. Реализация разветвления на три ветки.

Все команды переходов с возвратом предполагают безусловный переход (они не проверяют никаких флагов). При этом они требуют одного входного операнда, который может указывать как абсолютное значение нового адреса, так и смещение, складываемое с текущим значением адреса. Текущее значение счетчика команд (текущий адрес) сохраняется перед выполнением перехода в стеке.

Для обратного возврата в точку вызова подпрограммы (точку перехода) используется специальная команда возврата (RET или RTS). Эта команда извлекает из стека значение адреса команды перехода и записывает его в регистр-счетчик команд.

Особое место среди команд перехода с возвратом занимают команды прерываний (распространенное название – INT). Эти команды в качестве входного операнда требуют номер прерывания (адрес вектора). Обслуживание таких переходов осуществляется точно так же, как и аппаратных прерываний. То есть для выполнения данного перехода процессор обращается к таблице векторов прерываний и получает из нее по номеру прерывания адрес памяти, в который ему необходимо перейти. Адрес вызова прерывания и содержимое регистра состояния процессора (PSW) сохраняются в стеке. Сохранение PSW – важное отличие команд прерывания от команд переходов с возвратом.

Команды прерываний во многих случаях оказываются удобнее, чем обычные команды переходов с возвратом. Сформировать таблицу векторов прерываний можно один раз, а потом уже обращаться к ней по мере необходимости. Номер прерывания соответствует номеру подпрограммы, то есть номеру функции, выполняемой подпрограммой. Поэтому команды прерывания гораздо чаще включаются в системы команд процессоров, чем обычные команды переходов с возвратом.

Для возврата из подпрограммы, вызванной командой прерывания, используется команда возврата из прерывания (IRET или RTI). Эта команда извлекает из стека сохраненное там значение счетчика команд и регистра состояния процессора (PSW).

Отметим, что у некоторых процессоров предусмотрены также команды условных прерываний, например, команда прерывания при переполнении.

Конечно, в данном разделе мы рассмотрели только основные команды, наиболее часто встречающиеся в процессорах. У конкретных процессоров могут быть и многие другие команды, не относящиеся к перечисленным группам команд. Но изучать их надо уже после того, как выбран тип процессора, подходящий для задачи, решаемой данной микропроцессорной системой.

5.2. Система команд микропроцессора Intel 8086

5.2.1. Способы адресации и форматы команд микропроцессора I 8086

Микропроцессор Intel8086(К1810ВМ80) имеет двухадресную систему команд. Ее особенностью является отсутствие команд, использующих оба операнда из оперативной памяти. Исключение составляют лишь команды пересылки и сравнения цепочек байт или слов, которые в данном пособии рассматриваться не будут. Таким образом, в командах допустимы следующие сочетания операндов: RR, RS, RI, SI. Здесь R обозначает операнд, находящийся в одном из регистров регистровой памяти микропроцессора, S– операнд, находящийся в оперативной памяти, адрес которого формируется по одному из допустимых способов адресации, I– непосредственный операнд, закодированный в адресном поле самой команды. Формат команды во многом определяется способом адресации операнда, находящего в оперативной памяти, длиной используемого непосредственного операнда, а также наличием и длиной смещения, используемого при относительных режимах адресации.

Микропроцессор имеет все режимы адресации, общая схема которых была рассмотрена выше. Естественно, они имеют определенные особенности, присущие данному процессору.

Непосредственная адресация предполагает, что операнд занимает одно из полей команды и, следовательно, выбирается из оперативной памяти одновременно с ней. В зависимости от форматов обрабатываемых процессором данных непосредственный операнд может иметь длину 8 или 16 бит, что в дальнейшем будем обозначать data8 и data16 соответственно.

Механизмы адресации операндов, находящихся в регистровой памяти и в оперативной памяти, существенно различаются. К регистровой памяти допускается лишь прямая регистровая адресация. При этом в команде указывается номер регистра, содержащего операнд. 16-разрядный операнд может находиться в регистрах AX, BX, CX, DX, DI, SI, SP, BP, а 8-разрядный – в регистрах AL, AH, BL, BH, CL, CH, DL, DH.

Адресация оперативной памяти имеет свои особенности, связанные с ее разбиением на сегменты и использованием сегментной группы регистров для указания начального адреса сегмента. 16-разрядный адрес, получаемый в блоке формирования адреса операнда на основе указанного режима адресации, называется эффективным адресом (ЭА). Иногда эффективный адрес обозначается как ЕА (effectiveaddress). 20-разрядный адрес, который получается сложением эффективного адреса и увеличенного в 16 раз значения соответствующего сегментного регистра, называется физическим адресом (ФА).

Именно физический адрес передается из микропроцессора по 20-ти адресным линиям, входящим в состав системной шины, в оперативную память и используется при обращении к ее ячейке на физическом уровне. При получении эффективного адреса могут использоваться все основные режимы адресации, рассмотренные выше, а также некоторые их комбинации.

Прямая адресация предполагает, что эффективный адрес является частью команды. Так как ЭА состоит из 16 разрядов, то и соответствующее поле команды должно иметь такую же длину.

При регистровой косвенной адресации эффективный адрес операнда находится в базовом регистре BX или одном из индексных регистров DI либо SI:

ЭА=

[BX]

[DI]

[SI]

**.**

Обозначение имени регистра в квадратных скобках указывает на содержимое соответствующего регистра. Фигурные скобки – символ выбора одной из нескольких возможных альтернатив.

При регистровой относительной адресации эффективный адрес равен сумме содержимого базового или индексного регистра и смещения:

ЭА=

[BX]

[BP]

[DI]

[SI]

+

disp8

disp16

**.**

Обозначения disp8 и disp16 здесь и далее указывают на 8- или 16-разрядное смещение соответственно.

Эффективный адрес при базово-индексной адресации равен сумме содержимого базового и индексного регистров, определяемых командой:

ЭА=

[BX]

[BP]

+

[DI]

[*SI]*

**.**

Наиболее сложен механизм относительной базово-индексной адресации. Эффективный адрес в этом случае равен сумме 8- или 16-разрядного смещения и базово-индексного адреса:

**.**

+

+

ЭА=

*[*BX]

[BP*]*

[DI]

[SI]

disp8

disp16

Форматы двухоперандных команд представлены на рис. 3.16. Пунктиром показаны поля, которые в зависимости от режима адресации могут отсутствовать в команде.

а) формат командтипа RR і RS

1 байт

1 байт

1 байт

1 байт

КАП *d w*

*mdreg r/m*

*disp L*

*disp H*

1 байт

1 байт

1 байт

1 байт

1 байт

1 байт

КАП *s w*

*md* КАП*r/m*

*disp L*

*disp H*

*data L*

*data H*

б) формат командснепосредственнымоперандом

Рис. 5.16. Форматы двухоперандных команд микропроцессора I808:

Поле КОП содержит код выполняемой операции. Признак w указывает на длину операндов. При *w* = 1 операция проводится над словами, а при *w* = 0 – над байтами. Признак *d* указывает положение приемника результата. Признак *d* = 1, если результат записывается на место операнда, закодированного в поле reg, и *d* = 0, если результат записывается по адресу, закодированному полями (md, r/m).

Второй байт команды, называемый постбайтом, определяет операнды, участвующие в операции. Поле reg указывает регистр регистровой памяти согласно табл. 5.1

|  |  |  |
| --- | --- | --- |
| Таблица 5.1 | | |
| **reg** | **Регистр** | |
| **w=1** | **w=0** |
| 000 | AX | AL |
| 001 | CX | CL |
| 010 | DX | DL |
| 011 | BX | BL |
| 100 | SP | AH |
| 101 | BP | CH |
| 110 | SI | DH |
| 111 | DI | BH |

Поля md и r/m задают режим адресации второго операнда согласно табл. 5.2.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 5.2 | | | | | | | |
| **r/m** | ***md*** | | | | | | |
| 00 | | 01 | 10 | | 11 | |
| ***w* =1** | ***w*=0** |
| 000 | (BX)+(SI)  (DS) | | (BX)+(SI)+disp 8  (DS) | (BX)+(SI)+disp 16  (DS) | | AX | AL |
| 001 | (BX)+(DI)  (DS) | | (BX)+(DI)+disp 8  (DS) | (BX)+(DI)+disp 16  (DS) | | CX | CL |
| 010 | (BP)+(SI)  (SS) | | (BP)+(SI)+disp 8  (SS) | (BP)+(SI)+disp 16  (SS) | | DX | DL |
| 011 | (BP)+(DI)  (SS) | | (BP)+(DI)+disp 8  (SS) | (BP)+(DI)+disp 16  (SS) | | BX | BL |
| 100 | (SI)  (DS) | | (SI)+disp 8  (DS) | (SI)+disp 16  (DS) | | SP | AH |
| 101 | (DI)  (DS) | | (DI)+disp 8  (DS) | (DI)+disp 16  (DS) | | BP | CH |
| 110 | disp16  (DS) | | (BP)+disp 8  (SS) | (BP)+disp 16  (SS) | | SI | DH |
| 111 | | (BX)  (DS) | (BX)+disp 8  (DS) | (BX)+disp 16  (DS) | DI | | BH |

В этой таблице помимо определения режима адресации оперативной памяти указан также сегментный регистр, используемый по умолчанию для получения физического адреса. Использование другого сегментного регистра возможно введением специального префикса (дополнительного байта, который записывается перед командой).

В командах, использующих непосредственный операнд, признак *s* вместе с признаком *w* определяет разрядность непосредственного операнда, записываемого в команде, и разрядность выполняемой операции согласно [табл. 5.3](http://www.intuit.ru/department/hardware/archhard2/6/2.html#table.6.3).

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 5.3  **Использование признака *s*** | | | |
| ***w*** | ***s*** | **Операция** | **Непосредственный операнд** |
| 0 | 0 | 8-разрядная | 8-разрядный |
| 0 | 1 | не используется | |
| 1 | 0 | 16-разрядная | 16-разрядный |
| 1 | 1 | 8-разрядный, расширяемый знаком до 16-ти разрядов при выполнении операции |

Изменение естественного порядка выполнения команд программы осуществляется с помощью команд передачи управления. К ним относятся команды переходов, циклов, вызова подпрограммы и возврата из нее, а также некоторые другие. Мы рассмотрим лишь первые две группы команд.

Классификация команд переходов в персональной ЭВМ представлена на рис. 5.17.

Команды переходов

Условные

Прямые

Косвенные

Внутрисегментные

Безусловные

Прямые

Косвенные

Межсегментные

Рис. 5.17. Классификация команд переходов IBM PC

Физический адрес выполняемой команды определяется содержимым указателя команд IP и сегментного регистра команд CS. Команды, меняющие значение обоих этих регистров, называются командами межсегментных переходов, а меняющие только значение IP, – командами внутрисегментных переходов.

Команды безусловных переходов производят модификацию регистра IP или регистров IP и CS без предварительного анализа ка-ких-либо условий. Существует пять команд безусловных переходов. Все они имеют одинаковую мнемонику JMP и содержат один операнд. Конкретный формат команды определяется соответствующим префиксом и приведен в общей таблице машинного представления команд (табл. 5.4).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 5.4  **Машинные коды некоторых команд** | | | | |
| **Команда** | **Байты кода команды** | | | **Схема операции** |
| **байт 1** | **байт 2** | **байты 3…6** |
| ADD | 000000dw | *md*reg r/m | (disp8/16) | r(r/m) = r + r/m |
| 100000sw | *md* 000 r/m | (disp8/disp16)d8/16 | r/m = r/m + d8/16 |
|  | 0000010w | data L | (data H) | ac = ac + d8/16 |
|  | | | | |
| SUB | 001010dw | *md*reg r/m | (disp8/16) | r(r/m)=r(r/m) - (r/m)r |
| 100000sw | *md* 101 r/m | (disp8/16)d8/16 | r/m = r/m - d8/16 |
|  | 0010110w | data L | (data H) | ac = ac - d8/16 |
| AND | 001000dw | *md*reg r/m | (disp8/16) | r(r/m) = r & r/m |
|  | 100000sw | *md* 100 r/m | (disp8/16)d8/16 | r/m = r/m & d8/16 |
|  | 0010010w | data L | (data H) | ac = ac& d8/16 |
| OR | 000010dw | *md*reg r/m | (disp8/16) | r(r/m) = r V r/m |
|  | 100000sw | *md* 001 r/m | (disp8/16)d8/16 | r/m = r/m V d8/16 |
|  | 0000110w | data L | (data H) | ac = ac V d8/16 |
| XOR | 001100dw | *md*reg r/m | (disp8/16) | r(r/m) = r + r/m |
|  | 100000sw | *md* 110 r/m | (disp8/16)d8/16 | r/m = r/m + d8/16 |
|  | 0011010w | data L | (data H) | ac = ac + d8/16 |
| MOV | 100010dw | *md*reg r/m | (disp8/16) | r = r/m, r/m = r |
| 1100011w | *md* 000 r/m | (disp8/16)d8/16 | r/m = d8/16 |
|  | 1011wreg | data L | (data H) | reg = d8/16 |
|  | 1010000w | disp L | disp H | аc=m; прямой адрес |
|  | 1010001w | disp L | disp H | m=ac; прямой адрес |
| CMP | 0011101w | *md*reg r/m | (disp8/16) | r - r/m |
|  | 0011100w | *md*reg r/m | (disp8/16) | r/m - r |
|  | 100000sw | *md* 111 r/m | (disp8/16)d8/16 | r/m - d8/16 |
|  | 0011110w | data L | (data H) | ac - d8/16 |
| INC | 1111111w | *md* 000 r/m | (disp8/16) | r/m = r/m+1 |
|  | 01000reg |  |  | reg = reg+1 |
| DEC | 1111111w | *md* 001 r/m | (disp8/16) | r/m = r/m-1 |
|  | 01001reg |  |  | reg = reg-1 |
| TEST | 1000010w | *md*regr/m | (disp8/16) | r & r/m |
|  | 1111011w | *md* 000 r/m | (disp8/16)d8/16 | r/m & d8/16 |
|  | 1010100w | data L | (data H) | ac& d8/16 |
| XCHG | 10010reg |  |  | reg↔AX |
|  | 1000011w | *md*reg r/m | (disp8/16) | reg↔r/m |
| JMP short | 11101011 | disp L |  | IP=IP+dispL |
| nearptr | 11101001 | disp L | disp H | IP=IP+dispH,L |
| wordptr | 11111111 | *md* 100 r/m | (disp8/16) | IP=(*EA*) |
| farptr | 11101010 | IP-L | IP-H,CS-L,CS-H | IP=IPH,L, CS=*CSH*,L |
| dwordptr | 11111111 | *md* 101 r/m | (disp8/16) | IP=(*EA*), CS=(*EA*+2) |
| *Условный переход* | \_\_\_\_\_\_ | \_\_\_\_\_\_ | «нет данных» | IP=IP+dispL, если условие выполнено, иначе к следующей команде |
| JZ (JE) | 01110100 | disp L | «нет данных» | ноль (равно) |
| JNZ (JNE) | 01110101 | disp L | «нет данных» | не ноль (не равно) |
| JS | 01111000 | disp L | «нет данных» | минус |
| JNS | 01111001 | disp L | «нет данных» | плюс |
| JO | 01110000 | disp L | «нет данных» | переполнение |
| JNO | 01110001 | disp L | «нет данных» | нет переполнения |
| JL (JNGE) | 01111100 | disp L | «нет данных» | меньше для чисел |
| JNL (JGE) | 01111101 | disp L | «нет данных» | не меньше для чисел |
| JG (JNLE) | 01111100 | disp L | «нет данных» | больше для чисел |
| JNG (JLE) | 01111101 | disp L | «нет данных» | не больше для чисел |
| JB (JNAE,JC) | 01110010 | disp L | «нет данных» | меньше для кодов |
| JNB (*JAE*, JNC) | 01110011 | disp L | «нет данных» | не меньше для кодов |
| JA (JNBE) | 01110010 | disp L | «нет данных» | больше для кодов |
| JNA (JBE) | 01110011 | disp L | «нет данных» | не больше для кодов |
| JP (JPE) | 01111010 | disp L | «нет данных» | четное число "1" |
| JNP (JPO) | 01111011 | disp L | «нет данных» | нечетное число "1" |

*Примечание:* в столбце «Схема операции»ac означает регистр-аккумулятор, в качестве которого используется регистр AX при *w*=1 и регистр AL при *w*=0.

При безусловном прямом внутрисегментном переходе новое значение указателя команд IP равно сумме смещения, закодированного в соответствующем поле команды, и текущего значения IP, в качестве которого используется адрес команды, записанной вслед за командой перехода. Команды прямых межсегментных переходов содержат в сeбе помимо нового значения IP и новое значение сегментного регистра CS.

Команды косвенных переходов (внутрисегментных и межсегментных) передают управление на команду, адрес которой определяется содержимым регистра или ячеек оперативной памяти, на которые указывает закодированный в команде перехода постбайт.

Команды условных переходов являются только внутрисегментными. По своему формату и способу формирования нового значения IP они полностью аналогичны команде внутрисегментного прямого перехода с 8-разрядным смещением. Отличие их заключается в том, что в командах условного перехода механизм формирования нового значения IP включается лишь при выполнении определенных условий, а именно, при определенном состоянии регистра флагов. При невыполнении проверяемого условия в IP остается его текущее значение, то есть адрес команды, следующей за командой условного перехода.

Ниже приведены примеры команд переходов различных типов.

1) Команды условного перехода

Формат:

КОП

disp L

IP = IP + 2, если условие не выполнено;

IP = IP +2 + disp L, если условие выполнено;

Пример:

JZ MARK; переход на метку MARK, если ZF = 1.

2) Команды прямого внутрисегментного перехода

Формат:

КОП

disp L

disp

IP=IP+∆+disp, где ∆ - длина команды перехода (2 или 3 в зависимости от длины смещения).

Примеры:

JMP shortptr MARK; переход на метку MARK, с использованием 8-разрядного смещения;

JMP nearptr MARK; переход на метку MARK, с использованием 16-разрядного смещения.

3) Команды прямого межсегментного перехода

Формат:

КОП

IP\_L

IP\_H

CS\_H

CS\_H

IP = IP\_H, IP\_L,

CS = CS\_H, CS\_L.

Пример:

JMP farptr MARK; переход на метку MARK к команде, находящейся в другом сегменте.

4) Команды косвенного внутрисегментного перехода.

Формат:

КОП

md

disp

disp

IP = [*EA* + 1, *EA*]; или

IP = <регистр>, если в постбайте задано обращение к регистровой памяти;

Пример:

JMP wordptr [BX + SI]; новое значение IP берется из двух последовательных байт памяти, эффективный адрес первого из которых определяется суммой регистров BX и SI.

5) Команды косвенного межсегментного перехода

Формат:

КОП

md

dispL

disp

IP = [*EA* + 1, *EA*],

CS = [*EA* + 3, *EA* + 2].

Пример:

JMP dwordptr [BX + SI]; сумма регистров BX и SI определяет эффективный адрес области памяти, первые два байта которой содержат новое значение IP, а следующие два байта–новое значение CS.

Команды циклов идентичны по формату и очень близки по вы-полняемым действиям командам условных переходов. Однако по сравнению с последними они имеют ряд особенностей, позволяющих эффективно использовать их при программировании циклических участков алгоритмов.

Один из наиболее распространенных видов циклического участка программы представлен на рис. 5.18.

да

нет

СЧ=0

СЧ=<число повторений>

тело цикла

СЧ=СЧ-1

Рис. 5.18. Структура счетного цикла с постпроверкой.

Команды циклов предназначены для упрощения действий декремента (уменьшения на 1) счетчика цикла, проверки условия выхода из цикла и перехода.

Некоторые команды цикла реализуют выход из цикла не только по значению счетчика, но и при выполнении некоторых других условий.

Описание команд цикла сведено в табл. 5.5. За исключением команды JCXZ, которая не изменяет значения регистра CX, при выполнении команд циклов производятся следующие действия:

CX=(CX)–1. Затем, если проверяемое условие выполнено, то IP=(IP)+disp8 с расширением смещения знаком до 16 разрядов, в противном случае IP не изменяется, и программа продолжает выполнение в естественном порядке.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 5.5  **Команды циклов** | | | | |
| **Название** | **Мнемоника** | **Альтернативная мнемоника** | **КОП** | **Проверяемое условие** |
| Зациклить | LOOP |  | 11100010 | (CX)=0 |
| Зациклить пока ноль (равно) | LOOPZ | LOOPE | 11100001 | (ZF=1)&(CX)=0) |
| Зациклить пока не ноль (неравно) | LOOPNZ | LOOPNE | 11100000 | (ZF=0)&((CX)=0) |
| Перейти по (CX) | JCXZ |  | 11100011 | (CX)=0 |

**6. Архитектура параллельных компьютерных систем**

Развитие элементно-конструкторской базы, основанное на физических принципах электроники, всегда (и это диалектически правильно) отстает от требований конкретно решаемых задач. Быстродействия одного компьютера недостаточно для, например, массового обслуживания группы перехватчиков, стартующих против налета баллистических целей. Своевременно пришло понимание того, что только *структурными* методами можно добиться необходимой производительности вычислительных средств. Таким основным структурным методом является *распараллеливание вычислений*.

Сначала стало ясно, что вместо одной ЭВМ в систему управления необходимо встраивать несколько, решающих единый набор функционально связанных задач. Такой коллектив ЭВМ получил название вычислительного комплекса (ВК). Информационная взаимосвязанность задач потребовала применения средств оперативного обмена данными внутри ВК и средств синхронизации. Однако временные затраты на организацию взаимодействия ЭВМ в ВК (накладные расходы) оказались весьма высоки. Возникла идея использования взамен *разделенной* оперативной памяти (ОП), реализованной в ВК, *общей (разделяемой)* ОП, которая связывает несколько центральных процессоров. Проблема затрат времени на обмен оказалась решенной. Так зародилось понятие *параллельной вычислительной системы* (ВС).

Впоследствии принципы распараллеливания стали применяться на всех уровнях разработки ВС: и отдельных устройств, и процессоров. Отметим, что такой способ распараллеливания, как конвейерный, использовался на раннем этапе создания ЭВМ и используется сейчас.

Приоритет в создании ВС на общей памяти принадлежит С. Крею, признанному «отцу супер-ЭВМ», главному конструктору системы CDC 6600, разработанной в 1963 году. Однако условия соперничества и сокрытия информации определили независимость отечественных разработок.

Необходимо отметить тех выдающихся ученых, чьими усилиями осуществлялась не только разработка, но и обоснование эффективности и целесообразности пути создания многопроцессорных ВС, ибо своевременно принятая техническая политика определяет эффективное развитие государства. (К сожалению, известна и негативная политика в области вычислительной техники, отвлекшая значительные технологические ресурсы.) Такими учеными являются: В. М. Бахарев, настойчиво пропагандировавший, продвигавший и руководивший исследованиями в области ВС на общей памяти; В. С. Бурцев и Б. А. Бабаян — конструкторы и ведущие разработчики семейства «Эльбрус», продолжающие борьбу за честь Отечества и поныне; М. А. Карцев — теоретик ВС и разработчик семейства, по его словам, «универсальных спецпроцессоров», объединивших в одном вычислительном устройстве скалярные и векторные операции; Б. А. Головкин, исследовавший эффективность внедрения ВС в сложные управляющие системы.

Какова же цель распараллеливания?

Универсальным критерием, используемым при решении задач распараллеливания, является минимум времени выполнения совокупности работ, распределяемых между процессорами.

Если несколько процессоров составляют ВС, то важной характеристикой ее эффективности (основные составляющие эффективности — производительность, надежность, стоимость) при специализированном использовании (например, в составе автоматизированной системы управления, АСУ) является коэффициент полезной загрузки *k*3. Для его определения находят коэффициенты загрузки процессоров:

**,

где *Ti*, *i* = 1, ..., *n*, — суммарное время занятости каждого процессора решением задачи на всем отрезке полного решения задачи, длиной *T*реш([рис.В.1](http://www.intuit.ru/department/hardware/paralltech/0/#image.B.1)). Тогда



Если *P0* — производительность одного процессора, то реальная производительность *PBC* ВС, состоящей из n процессоров, при решении данной задачи (!) находится:



где *P*0 определяется классом решаемых задач.

Т3

Т2

Т1

**6.1 Два уровня распараллеливания**

Развитие вычислительной техники характеризуется тем, что на каждом этапе новых разработок требования к производительности значительно превышают возможности элементной базы.

Это обусловлено задачами сложных систем управления в реальном времени, централизованным решением задач в сетях, имитационным моделированием сложных процессов (например, в ядерной физике), оперативным планированием и управлением и решением других задач исследования операций, преодолевающих «проклятие размерности». Такие задачи требуют концентрации вычислительных мощностей, постоянно поддерживая высокую актуальность проблемы создания супер-ЭВМ.

Уже давно стало ясно, что только структурными методами можно уравнять возможности вычислительных средств и требуемые скорости решения на них задач. Под структурными понимают методы распараллеливания работ. К распараллеливанию прибегают при проектировании отдельных устройств ЭВМ — устройств управления, буферов команд, каналов обращения к памяти и модулей памяти, многофункциональных арифметическо-логических устройств (АЛУ), повсеместно применяемых конвейеров и т. д. Но к распараллеливанию же прибегают и в проектировании совместной работы многих процессоров при параллельной или распределенной обработке информации, вводя в обращение термин "вычислительная система (ВС)".

Технический прогресс, несомненно, сказывается на росте частоты работы элементной (элементно-конструкторской) базы, на повышении степени интеграции, но благодаря ему появляются все новые задачи, требующие еще более значительного роста производительности вычислительных средств. Это можно считать законом, приводящим к новым уловкам при совмещении работы устройств ВС, при увеличении их количества в системе, при увеличении их эффективности в процессе решения задач.

Под ***эффективностью работы устройства*** в составе ВС понимают степень его участия в общей работе ВС при решении конкретной задачи — коэффициент загрузки устройства. Распараллеливание работ оправдано, если приводит к существенному росту усредненного по всем устройствам коэффициента загрузки оборудования при решении задач. Это непосредственно сказывается на времени решения. Сегодня говорят не о специальном классе задач, а о задачах, ориентирующих ВС на универсальность, что обусловлено современными областями применения.

Важным революционизирующим моментом стал переход на микропроцессорную элементно-конструкторскую базу, обусловившую построение мультимикропроцессорных ВС.

Сложилось представление о двух основных уровнях, на которых в ВС применяются практические методы распараллеливания:

* на уровне программ, процессов, процедур (первый уровень распараллеливания);
* на уровне команд и операций (второй уровень распараллеливания).

Эти уровни обусловили уровни структуризации ВС на пути превращения ее в супер-ЭВМ. Современным практическим воплощением первого уровня структуризации являются однородные многопроцессорные ВС на общей (разделяемой) оперативной памяти. Они получили название симметричных ВС за обеспечение «равноправия» составляющих модулей. Окончательное признание симметричных ВС положило конец поиску «экзотических» архитектур, эффективных лишь при решении определенных классов задач. Универсальность симметричных ВС, возможность реализации на них любых вычислительных процессов с высокой эффективностью оборудования иллюстрируются многими применениями и анализируются ниже.

Уровень команд и операций наиболее ярко представлен многофункциональными АЛУ и их обобщением — решающими полями, представляющими разделяемый вычислительный ресурс многих процессоров. Некоторые современные проекты ВС в разной степени предполагают такой ресурс. Здесь основная проблема — полная загрузка отдельных исполнительных устройств (ИУ) в процессе выполнения программы.

Различают два способа реализации такой загрузки: динамический и статический.

Динамическая загрузка осуществляется аппаратурой в процессе выполнения программы. Она использует упрощенные алгоритмы распараллеливания.

Статическая загрузка предусматривается при трансляции программы. Транслятор оптимизирует использование оборудования, также решая задачи распараллеливания. Это выражается в формировании «длинных» командных слов, задающих работы устройствам АЛУ в каждом машинном такте.

Основная сложность распараллеливания заключается в соблюдении частичной упорядоченности распределяемых работ. Поэтому решение задач синхронизации параллельного вычислительного процесса сопровождает все усилия по организации совместной работы устройств. Это сказывается и при решении всех задач эффективного использования расслоенной многоуровневой памяти ВС.

Отечественный опыт создания семейства МВК (многопроцессорных вычислительных комплексов) «Эльбрус», модели которого относятся к симметричным ВС, и последующее проектирование позволили проанализировать, разработать и применить ряд существенно новых, важных и перспективных решений в распараллеливании как самого вычислительного процесса, так и работы отдельных устройств. Разработка пронизана такими решениями, они ложатся в основу проектирования развития семейства, являются основой обобщений и дальнейшего исследования возможности применения.

**6.2 Классификация параллельных ВС.**

**6.2.1 Потоки команд и потоки данных**

Общепринятую удачную классификацию ВС, которую предложил в 1970 г. Г. Флин (США). Основным определяющим архитектурным параметром он выбрал взаимодействие *потока команд* и *потока данных* (операндов и результатов).

В ЭВМ классической архитектуры ведется последовательная обработка команд и данных. Команды поступают одна за другой (за исключением точек ветвления программы), и для них из ОЗУ или регистров так же последовательно поступают операнды. Одной команде (операции) соответствует один необходимый ей набор операндов (как правило, два для бинарных операций). Этот тип архитектуры – «***один поток команд – один поток данных», ОКОД (SISD  – "SingleInstruction, SingleData")*** условно изображен на [рис. 6.1](http://www.intuit.ru/department/hardware/paralltech/1/#image.1.1).

Память

Поток данных после обработки

Поток команд

Поток данных для обработки

Процессор

Рис. x.1.  ВС типа ОКОД (SISD)

Тип ***ОКМД – «один поток команд – много потоков данных» (SIMD – "SingleInstruction – MultiplеData")*** охватывает ВС, в которых одной командой обрабатывается набор данных, множество данных, вектор, и вырабатывается множество результатов. Это векторные и матричные системы, в которых по одной команде выполняется одна и та же операция над всеми элементами массива – вектора или матрицы, распределенными между *процессорными (обрабатывающими) элементами ПЭ или процессорами.* Принцип обработки показан на [рис. 6.2](http://www.intuit.ru/department/hardware/paralltech/1/#image.1.2).

ПЭ1

ПЭ2

ПЭ*n*

Память данных

Память программ

Устройство управления

Поток данных для обработки

Поток данных после обработки

Поток команд

……………

Рис. 6.2.  ВС типа ОКМД (SIMD)

Отечественные векторные ВС – ПС-2000, ПС-2100. Допускают организацию матричной обработки. Классический пример матричной архитектуры – ILLIAC-1V (США).

К типу ***МКОД – «много потоков команд — один поток данных» (MISD – "MultipleInstruction – SingleData")*** принято относить векторный конвейер (обычно в составе ВС, чтобы подчеркнуть основной используемый принцип вычислений), например, в составе ВС *Crey*-1, «Электроника ССБИС». На векторном конвейере производится последовательная обработка одного потока данных многими обрабатывающими устройствами (ступенями, станциями) конвейера.

К такому же типу относится ВС, реализующая *макроконвейер* (ВС «Украина»). В ней задача, решаемая циклически, «разрезается» на последовательные этапы, закрепляемые за отдельными процессорами. Запускается конвейер многократного выполнения цикла, составляющего задачу.

Принцип обработки показан на [рис. x.3](http://www.intuit.ru/department/hardware/paralltech/1/#image.1.3)*.*



Память

Потоки команд

Поток данных для обработки

Поток данных после обработки

УУ

1

2

*n*

……………

Рис.6.3.  ВС типа МКОД (MISD)

Тип ***МКМД – «много потоков команд – много потоков данных» (MIMD – "MultipleInstruction – MultipleData")***cоответствует более полному и независимому распараллеливанию. К этому типу относятся, например, все многопроцессорные вычислительные комплексы (МВК) семейства «Эльбрус».

**6.1.2** «**Фон-Неймановские» и «не-Фон-Неймановские» архитектуры**

**Немного истории.** Первую ЭВМ создал в 1939 г. в США профессор Джон Атанасов, болгарин, со своим аспирантом К. Берри. Две малые ЭВМ, созданные ими в период 1937 – 1942 гг., были прототипами большой ЭВМ АВС для решения систем линейных уравнений, которая в 1942 г. доводилась по устройствам ввода-вывода и должна была войти в строй в 1943 г., но призыв Атанасова в армию в 1942 г. воспрепятствовал этому. Проект электронной ЭВМ Эниак (ElectronicsNumericalIntegratorandComputer) был сделан в 1942 г. Д. Моучли и Д. Эккертом и осуществлен в 1945 г. в Муровской электротехнической лаборатории Пенсильванского университета. В 1946 г. Эниак был публично продемонстрирован в работе. В нем впервые были применены триггеры. Рождение Эниак считают началом компьютерной эры, посвящая ему научные симпозиумы и другие торжественные мероприятия. (Международный симпозиум, посвященный 50-летию первой ЭВМ, был проведен и в Москве в июне 1996 г.)

Однако еще в начале 40-х г. XX века Атанасов поделился с Моучли информацией о принципах, заложенных в ЭВМ АВС. Хотя Моучли впоследствии утверждал, что он не воспользовался этой информацией в патенте на Эниак, суд не согласился с этим. Вернувшись из армии после войны, Атанасов узнал, что более мощная ЭВМ Эниак уже создана, и потерял интерес к этой теме, не поинтересовавшись, насколько Эниак похож на его ЭВМ АВС.

Известный английский математик Алан Тьюринг был не только теоретиком по информации и теории алгоритмов, автором теоретического автомата «машины Тьюринга», но и талантливым инженером, создавшим в начале 1940-х г. первую работающую специализированную ЭВМ. Эта ЭВМ под названием «Колосс» была сконструирована и сделана им совместно с Х. А. Ньюменом в Блетчи (Англия) и начала работать в 1943 г. Сообщения о ней своевременно не публиковались, т. к. она использовалась для расшифровки секретных германских кодов во время войны.

Основные архитектурно-функциональные принципы построения ЦВМ были разработаны и опубликованы в 1946 г. венгерским математиком и физиком Джоном фон Нейманом и его коллегами Г. Голдстайном и А. Берксом в ставшем классическим отчете «Предварительное обсуждение логического конструирования электронного вычислительного устройства».Основополагающими принципами ЭВМ на основании этого отчета являются: 1) принцип программного управления выполнением программы, и 2) принцип хранимой в памяти программы. Они легли в основу понятия ***фон-Неймановской архитектуры***, широко использующей *счетчик команд*.

Вернемся к настоящему. Счетчик команд отражает «узкое горло», которое ограничивает поток команд, поступающих на исполнение, их последовательным анализом.

Альтернативной архитектурой является «***не-фон-Неймановская» архитектура***, допускающая одновременный анализ более одной команды. Поиски ее обусловлены необходимостью распараллеливания выполнения программы между несколькими исполнительными устройствами – процессорами. Счетчик команд при этом не нужен. Порядок выполнения команд определяется наличием исходной информации для выполнения каждой из них. Если несколько команд готовы к выполнению, то принципиально возможно их назначение для выполнения таким же количеством свободных процессоров. Говорят, что такие ВС управляются *потоком данных (dataflow).*

В командах проверки условия возможно альтернативное задание адреса результата (ИЛИ – ИЛИ).

Адреса результатов являются адресами ПК, т. е. результаты выполнения одних команд в качестве операндов могут поступать в текст других команд. Команда не готова к выполнению, если в ее тексте отсутствует хотя бы один операнд. Ячейка, обладающая полным набором операндов, переходит в возбужденное состояние и передает в селекторную сеть информационный пакет (токен), содержащий код операции и необходимую числовую и связную информацию. Он поступает по сети на одно из исполнительных устройств.

Там же операция выполняется, и в распределительную сеть выдается результирующий пакет, содержащий результат вычислений и адреса назначения в ПК (возможно, за счет выбора альтернативы, т. е. такой выбор – тоже результат). По этим адресам в ПК результат и поступает, создавая возможность активизации новых ячеек. После выдачи токена в селекторную сеть операнды в тексте команды уничтожаются, что обеспечивает повторное выполнение команды в цикле, если это необходимо.

Общая схема потоковых ВС представлена на [рис. 6.4](http://www.intuit.ru/department/hardware/paralltech/1/2.html#image.1.4).

ИСПОЛНИТЕЛЬНЫЕ

УСТРОЙСТВА

**. . . . . . . . . . . . .**

КОММУНИКАЦИОННАЯ

СЕТЬ

Распределительная

сеть

Селекторная сеть

ПАМЯТЬ КОМАНД (ПК)

ТОКЕНЫ

**. . . . . . . . . . . . .**

Рис. 6.4. «Идеальная» потоковая ВС

Программа или ее часть (сегмент) размещается в памяти команд ПК, состоящей из ячеек команд. Команды имеют следующую структуру.

{код операции, операнд 1, ..., операнд L,

адрес результата 1, ..., адрес результата M}

Селекторная и распределительная сети образуют ***коммуникационную сеть*** ВС.

Ожидаемая сверхвысокая производительность такой системы может быть достигнута за счет одновременной и независимой активизации большого числа готовых команд, проблематичном допущении о бесконфликтной передаче пакетов по сетям и параллельной работы многих исполнительных устройств.

Существует ряд трудностей, в силу которых «не-фон-Неймановские» архитектуры не обрели технического воплощения для массового применения в «классическом», отраженном выше, исполнении. Однако многие устройства используют данный принцип, но чаще всего взаимодействие процессоров при совместном решении общей задачи и их синхронизация при использовании общих данных основаны на анализе готовности данных для их обработки. Это дает основание многим конструкторам заявлять, что в своих моделях они реализовали принцип *dataflow*.

**6.1.3. Системы с общей и распределенной памятью**

***Системы с общей (разделяемой) оперативной памятью*** образуют современный класс ВС – многопроцессорных супер-ЭВМ. Одинаковый доступ всех процессоров к программам и данным представляет широкие возможности организации параллельного вычислительного процесса *(параллельных вычислений).* Отсутствуют потери реальной производительности на межпроцессорный (между задачами, процессами и т. д.) обмен данными ([рис. 6.5, *a*](http://www.intuit.ru/department/hardware/paralltech/1/2.html#image.1.5)).

***Системы с распределенной памятью*** образуют вычислительные комплексы (ВК) – коллективы ЭВМ с межмашинным обменом для совместного решения задач ([рис. 6.5, *б*](http://www.intuit.ru/department/hardware/paralltech/1/2.html#image.1.5)).

П

С

*n*

1

. . . . . . . .

ЛОП1

Система обмена

1

*n*

Процессоры

1

ОП1

ОП1

ОП*n*

*а*

*б*

Рис. 6.5.  ВС с общей (*а*) и распределённой (*б*) памятью

В ВК объединяются вычислительные средства систем управления, решающие специальные наборы задач, взаимосвязанных по данным. Принято говорить, что такие ВК выполняют *распределенные вычисления*, а сами ВК называют *распределенными ВК.*

Другое, противоположное воплощение принципа МИМД – *масспроцессорные* или высокопараллельные архитектуры, объединяющие сотни – тысячи – десятки тысяч процессоров.

В современных супер-ЭВМ наметилась тенденция объединения двух принципов: общей (распределяемой) и распределенной (локальной) оперативной памяти (ЛОП). Такая структура используется в проекте МВК «Эльбрус-3» и «Эльбрус-3М» ([рис. 6.6](http://www.intuit.ru/department/hardware/paralltech/1/2.html#image.1.6)).

1

*n*

**. . . . . . .**

Коммутатор

**. . . . . . . . . . . . . . .** .

Общая ОП

ЛОП*1*

ЛОП*n*

М1

М*n*

Рис. 6.6.  Схема ВС с модулями локальной памяти

**6.2. Способы межмодульного соединения (комплексирования)**

Различают два противоположных способа комплексирования: *с общей шиной (шинная архитектура) и с перекрестной (матричной) коммутацией* модулей ВС (процессоров, модулей памяти, периферии).

На [рис. х.7](http://www.intuit.ru/department/hardware/paralltech/1/2.html#image.1.7) представлена система с общей шиной. Шина состоит из линий, по которым передаются информационные и управляющие сигналы.

1

Общая шина

**. . . . . . . . . . . . . . . . . . . . .**

Периферийные устройства

Процессоры

Модули памяти

*n*

1

*m*

**Рис. 6.7.**  Схема ВС с общей шиной

Шина используется в режиме разделения времени, при котором лишь один модуль в данный момент работает на передачу. Принимать принципиально могут все модули, хотя преимущественно информация при выдаче в нее адресуется. Применяется в микро- и мини-ЭВМ при сравнительно небольшом числе модулей. Практически производится разделение шины на управляющую, адресную и шину данных.

В высокопроизводительных ВС для возможности одновременного обмена многими парами абонентов используется перекрестная или матричная коммутация.

***Матричный коммутатор*** можно представить (прямоугольной) сеткой шин. К одному концу каждой подсоединен источник-потребитель информации ([рис. 6.8](http://www.intuit.ru/department/hardware/paralltech/1/2.html#image.1.8)). Точки пересечения — *узлы* этой сетки — представляют собой *управляющие ключи*, которые соединяют или разъединяют соответствующие шины, устанавливая или прекращая связь между модулями. Реализуется связь «каждый с каждым». Одновременно могут связываться многие (до *n*/2) пары модулей.

1

2

*n*

Коммутатор

Управляющие

сигналы

.

.

Процессоры

.

. . .

. . .

*а*

1

2

…

n

1 2 *m*

Процессоры

Коммутатор

Управляющие

сигналы

Модули памяти

*б*

Рис. 6.8.  Матричные коммутаторы: *а*– перекрёстная коммутация процессоров, *б*– коммутация процессоров и модулей памяти

На [рис. 6.8, *а*](http://www.intuit.ru/department/hardware/paralltech/1/2.html#image.1.8)– перекрестная связь между процессорами в ВС с распределенной памятью, на [рис. 6.8, *б*](http://www.intuit.ru/department/hardware/paralltech/1/2.html#image.1.8)– между n процессорами и m модулями ОП.

|  |  |  |
| --- | --- | --- |
| mhtml:file://F:\INTUIT_ru%20Курс%20Архитектура%20__%20Лекция%20№2%20Микропроцессорные%20системы%20и%20способы%20распараллеливания2.mht!http://www.intuit.ru/img/empty.gif | **6.3. МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ И СПОСОБЫ РАСПАРАЛЛЕЛИВАНИЯ**   |  | | --- | | **6.3.1. Мультимикропроцессорные вычислительные системы**  В настоящее время выбор сделан в пользу многопроцессорных симметричных ВС типа *MIMD*, обеспечивающих виртуализацию вычислительных ресурсов. Основу такой ВС составляет *суперскалер*, сосредоточивший в себе все способы достижения максимального быстродействия при выполнении одиночной программы. Векторные и векторно-конвейерные процессоры и системы получили своё место. Их эффективность как самостоятельных установок могла быть достаточно высокой только при решении специальных задач и тестов. Поэтому достаточно быстро выяснилось, что эти установки могут выполнять функции интеллектуальных терминалов при решении основной задачи на другом универсальном вычислительном средстве и выполнять лишь отдельные его заявки. Сегодня стало окончательно ясно, что первые эффективны лишь в роли специализированных вычислительных устройств для решения специальных задач. Вторые твердо заняли место в составе многофункциональных арифметическо-логических устройств (АЛУ) суперскалеров, ибо без конвейеров мы не мыслим себе выполнение всех операций ВС.  Складывается и структура памяти ВС, которая может совмещать в одной установке все способы доступа: от разделяемой (общей) до распределенной оперативной памяти. Однако ограниченные возможности эффективной работы с общей памятью часто диктуют иерархическую структуру ВС, где уровни иерархии (кластеры) отличаются или способом доступа к оперативной памяти, или тем, что каждый кластер имеет свою собственную физическую память в общем адресном пространстве. При этом принцип буферизации, основанный на многоуровневой по быстродействию (и, конечно, — различной по технологии) памяти, на активном использовании Кэш-памяти, продолжает развиваться. Кэш-память, как память самого высокого уровня, претерпевает функциональное разбиение в зависимости от типа данных, для хранения которых она предназначена, либо, в зависимости от вида обработки, — программ или данных.  Все сказанное выше подтверждает перспективность структурных решений при проектировании многопроцессорного комплекса «Эльбрус-3» и его микропроцессорного развития «Эльбрус-3М», «Эльбрус-2К». Таким образом, структура «длинного командного слова» (архитектура *VLIW*, лежащая в основе *EPIC*) попадает в разряд классических.  Сейчас микропроцессор, сконцентрировавший все достижения микроэлектроники, является основной составляющей элементно-конструкторской базы ВС. Поэтому понятие «мультимикропроцессорные ВС» пришло на смену понятию «микропроцессорные ВС».  Анализ современных мультимикропроцессорных ВС позволяет выделить те развиваемые характерные решения, которые в условиях микроминиатюризации и снижения энергоемкости, «экономного» логического развития обеспечивают необходимые свойства универсального применения.  Такими решениями являются следующие.  **6.3.2. Многопроцессорные кристаллы**.   1. Воспроизведение многопроцессорной ВС на одном кристалле в значительной степени характерно для сигнальных вычислительных средств, специализирующихся на обработке двух- и трехмерных изображений, которые применяются в цифровом телевидении и радиовещании, при передаче изображений по каналам связи и др. Такие средства эффективно используются в качестве нейрокомпьютеров.   Например, на одном кристалле *MVP (MultimediaVideoProcessor)* семейства *TMS 320 C80* (фирма *TexasInstrument*) расположены 4 32-разрядных цифровых сигнальных процессора (*DSP*– DigitalSignalProcessor) с фиксированной запятой (*ADSP-0 – ADSP-3*). Их особенность — высокая степень конвейеризации и до 64 бит длина командного слова для параллельного выполнения нескольких операций. Система команд содержит команды над битовыми полями и структурами данных, несущими графическую информацию. Такая специализация обусловила понятие — *DSP-архитектура*.  Процессоры работают независимо. Т. е. ВС – типа *MIMD* – (Multiple-Instruction, Multiple-Data). Программируются отдельно на ассемблере или ЯВУ. Данными обмениваются через общую внутрикристальную память.  Каждый из *ADSP* содержит КЭШ-память команд (2 Кбайта), и через матричный коммутатор *Crossbar* получает доступ к 32 из имеющихся 50 Кбайт быстродействующей статической внутренней памяти. Память расслоенная — поделена на сегменты. Если два и более процессора в одном цикле попытаются обратиться к одному сегменту, аппаратная система управления доступом с циклическим изменением приоритета (*roundrobinprioritization*) позволит сделать это только одному процессору.  32-разрядное АЛУ *ADSP* может работать как два 16-разрядных или четыре 8-разрядных АЛУ. Этого достаточно для обработки видеоизображений. Специальные блоки ускоряют обработку графики. Блоки генерации адресов формируют кольцевые (бесконечные) буферы. Аппаратно поддержаны три вложенных цикла.  *RISC*-процессор управляет четырьмя *ADSP* с помощью диспетчера. Диспетчер и планировщик заданий тесно взаимодействуют с контроллером пересылок. Кроме того, управляющий процессор самостоятельно выполняет вычисления и обеспечивает обмен с внешними устройствами. Содержит встроенный блок плавающей арифметики и набор векторных операций с плавающей запятой, оптимизированных для обработки изображений, звука и трехмерной графики.  **6.3.3. Транспьютерная технология.**   1. Представленная выше архитектура обладает такой конструктивной законченностью, которая позволяет как встраивать ее в некоторую систему, так и организовать взаимодействие нескольких кристаллов. Это обеспечивается развитыми средствами связи и обмена данными.   Возможность комплексирования привлекла внимание еще на раннем этапе развития микропроцессоров (в середине 1980-х годов) и привела к построению *транспьютеров* — микропроцессоров, снабженных развитыми средствами комплексирования. Таким образом, создавались "кирпичики", на основе которых можно было создавать сложные структуры. Эта тенденция не только сохранилась, но является необходимым средством построения мультимикропроцессорных ВС.  Преследуя многофункциональность средств обмена, не обязательно требовать их размещения на одном кристалле с центральным процессором. Так, фирма *AnalogDevices* предлагает микропроцессоры *ADSP-21060/62 SHARC* ("АКУЛА") для цифровой обработки сигналов, специально предназначенные для комплексирования.  Средства комплексирования «АКУЛЫ»:   * + магистраль для подключения 6 «АКУЛ» и одного ХОСТ-процессора (управляющего, с привилегированным доступом к магистрали, а также к памяти каждого процессора — через специальный порт);   + сигнальные регистры в составе каждого процессора, непосредственно связанные (одной ножкой) с каждым из других процессоров — для контроля их состояния;   + ЛИНКи — каждый процессор имеет 6 выходов (ЛИНКов) для непосредственной связи «процессор – процессор».  1. ***Общее адресное пространство***комплексируемых микропроцессоров «АКУЛА» обеспечивает псевдообщую память и исключает необходимость программной организации обмена данными. Если адрес физически принадлежит ОП другого процессора, то обмен организуется автоматически, без вмешательства пользователя (т. е. программно не предусматривается). 2. ***Межпроцессорный (магистральный) обмен*** инициируется в том случае, если адрес считывания или записи принадлежит адресному пространству другого процессора (единичный обмен). Аналогично возникают групповые пересылки данных с использованием "чужого" адресного пространства.   Пользователь не составляет программу обмена, даже для контроллера обмена данных. Достаточно указать «чужие» адреса.  Процессоры обмениваются сигналами состояния. Поэтому каждый процессор знает, кто является «хозяином» магистрали, т.е. ведет обмен, и свой приоритет в очереди к магистрали. По завершении каждого обмена производится циклическая смена приоритетов процессоров, которым нужна магистраль. Процессор с максимальным приоритетом становится «хозяином». Обмен может прерываться только ХОСТ-процессором.  Микропроцессор утверждается в роли основы элементно-конструкторской базы ВС, и это поняли ведущие разработчики.  В этом смысле привлекает внимание трансформация интересов "отца суперкомпьютеров" С.Крея, который признал определяющую роль принципа *MIMD* при построении ВС *CraySuperserver 6400 System* (*CS640*), выпускаемой корпорацией *CrayResearch* в сотрудничестве с компанией *SUN Microsystems* (сотрудничество с фирмой *SUN* ныне характерно и для ведущих российских разработчиков).  Система предполагает наращиваемую конфигурацию от 4 до 64 процессоров *SuperSPARC*. Принято компромиссное решение на основе классической схемы разделения (общей) ОП при многопроцессорной обработке и распределенной памяти при параллельной обработке массивов. Чтобы работать с частично распределенной памятью в ОЗУ, ВС имеет в любой конфигурации 4 шины. Шина использует сетевую технологию «коммутации пакетов». Это позволяет находить путь обмена единицами информации в соответствии с занятостью или освобождением шин.  В целом, архитектуру следует считать шинной, хотя наличие нескольких шин делает ее промежуточной между шинной и использующей матричный коммутатор.  **6.4. Направление «мини-супер» призвано поддержать персональный компьютер**  То, что говорилось выше, «по умолчанию» соответствует разработке супер-ЭВМ, предназначенных для решения особо сложных задач в составе систем управления в реальном времени, моделирования сложнейших физических процессов, решения задач исследования операций, задач искусственного интеллекта, выполнения роли майнфреймов и серверов в локальных, корпоративных и глобальных сетях.  Супер-ЭВМ уникальна, мало тиражируема, цена ее высока.  С другой стороны, ничто уже не может остановить "победного шествия" персональных компьютеров. Область применения их стала всеобъемлющей. Они используются и там, где могут справиться с задачами, и там, где уже не справляются, несмотря на применение современныхсуперскалеров.  Тогда целесообразно поставить следующую проблему.  Введем в состав персонального компьютера (*РС*), как его внешнее устройство, мультимикропроцессорную систему (мультипроцессор), использование которого в монопольном и однозадачном режиме может обеспечить успешное решение задач повышенной сложности.  Действительно, разрешение этой проблемы позволило бы заполнить определенную нишу между супер-ЭВМ и *PC*, вывести персональный компьютер на уровень мини-супер-ЭВМ. Применение мультипроцессора *РС* в однопрограммном режиме, при жестком распределении памяти, использование (см. далее) прогрессивной технологии «одна программа — много потоков данных» позволяют существенно снизить издержки производительности на работу ОС, легко «врезать» их в современные операционные системы компьютеров. Сборка такой системы должна производиться на основе существующей микропроцессорной элементно-конструкторской базы, с минимальным использованием вновь разрабатываемых элементов.  Здесь воспроизводится упомянутая выше идея о наличии мониторной системы, на которой решается основная задача, и о наличии интеллектуального терминала, который берет на себя функции, обеспечивающие общую эффективность системы.  Общая схема такой установки показана на [рис. 6.9](http://www.intuit.ru/department/hardware/paralltech/2/2.html#image.2.1). Выбраны конкретные значения параметров.  Мониторная  система — PC  Память команд  Память команд  ЛОП  ЛОП  КЭШ команд  КЭШ команд  Процессор 2  Процессор 1  Общая шина или коммутатор  Средства синхронизации (семафоры, закрытые адреса, предикаты)  Синхронизатор  Общая разделяемая память  Модуль 1  Модуль 16  Рис. 6.9.  Схема ВС для персонального компьютера  Мультимикропроцессорную приставку к персональному компьютеру целесообразно разработать на основе исследования принципов построения локально-асинхронной архитектуры (*SPMD*-технологии). Важным достоинством архитектуры является сведение традиционных функций ОС на уровень команд. Т.е. система команд мультипроцессора такова, что позволяет реализовать функции управления параллельным процессом, не требуя запуска процедур ОС. Способствует простоте управления параллельным процессом также монопольный и однозадачный режим использования мультипроцессора. Ниже мы подробнее остановимся на принципах *SPMD*-технологии. Предполагая первоначальное знакомство с этими принципами, отметим следующее.  Известно (см. далее), что семафоры — универсальное средство синхронизации. Однако семафоры традиционно используют ОС. Чтобы этого избежать, семафоры следует реализовать с помощью *предикатного механизма*, т.е. с использованием памяти предикатов.  Семафорный механизм может быть эффективно реализован с помощью *механизма закрытия адресов* (памяти закрытых адресов).  Тогда, в общем случае применения семафоров, должны быть введены команды следующего вида.  ***Считать по семафору (Сч(С) А***). Считывание по адресу производится в случае, если указанный семафор (реализованный в памяти регистрового типа, наряду с индексными и базовыми регистрами) открыт. Если семафор закрыт, реализуется ожидание данного ПЭ без прерывания (т.е. в данном применении пользователь может быть допущен к операциям над семафорами типа «жужжать»).  ***Записать по семафору (Зап(С) А***). Запись по адресу производится аналогично предыдущей команде.  При использовании памяти закрытых адресов необходима лишь команда*Закрыть адрес*. Любое последующее считывание по этому адресу циклически возобновляется (в режиме "жужжания") до тех пор, пока по этому же адресу другой процессор не произведет запись.  В случае использования механизма предикатов адрес некоторой булевой переменной записывается в специальные разряды командного слова. Команда, для которой указанный в ней предикат имеет значение 0, выполняется, в соответствии с кодом операции, в *спекулятивном режиме* в двух модификациях:   * ожидается присвоение данному предикату значения 1 (в режиме «жужжания»); * пропускается выполнение данной команды.   ПЭ реализует идею *RISC*-архитектуры и представляет собой функционально законченное устройство, состоящее из микропроцессора, схем обрамления и локальной оперативной памяти (ЛОП). Локальная память процессора содержит область для хранения стеков вычислительного процесса, в том числе — стеков подпрограмм и вложенных циклов. В других областях этой памяти хранятся модификаторы, дескрипторы массивов и локальные величины. Здесь же находятся микропрограммы, реализующие систему команд ВС.  Общая (разделяемая) память (ОП) содержит M модулей с общим адресным пространством и реализует принцип ***интерливинга***, предполагающий, что смежные ячейки памяти находятся в разных модулях.  ***Синхронизатор*** предназначен для обеспечения одновременного пуска программ или их модулей.  Возможно использование простейших коммутаторов для обмена ПЭ с модулями памяти. | |

**6.5. Распределенный и разделяемый вычислительный ресурс второго уровня. Решающие поля**

Произведем некоторые обобщения.

Итак, второй уровень распараллеливания предполагает распределение команд, инструкций, операций, элементарных функций и других несложных процедур — для выполнения исполнительными устройствами процессоров или в общем вычислительном ресурсе симметричной ВС. Здесь существуют свои проблемы, связанные с "элементарным" характером операций, небольшим объемом содержащихся в них работ, с их еще большей критичностью по отношению к "накладным расходам" на организацию и синхронизацию. Мы предполагаем, что исполнительные устройства ВС образуют вычислительный ресурс второго уровня (распараллеливания).

Сложились традиции построения этого ресурса, где основное внимание уделяется построению многофункциональных АЛУ. Однако в ряде архитектур пока еще робко пробивает себе дорогу объединение АЛУ в единый разделяемый ресурс системы — построение решающих полей.

Проработка этой идеи проводилась неоднократно в отечественной практике разработки ВС. Она проявлялась во включении в состав ВС специализированных процессоров на правах интеллектуальных терминалов для эффективного выполнения определенных операций. Это были векторные процессоры с доступом от нескольких ЦП (ПС-3100), векторный сопроцессор в ПС-2100. Эта же идея фактически воплощена в семействе «Эльбрус», допускающем включение в свой состав спецпроцессоров — эмуляции других систем, векторно-конвейерных модулей и др.

При реализации идеи решающего поля проблема выбора и развития вычислительного ресурса неотделима от выбора вариантов архитектуры системы вообще. Единственным средством обоснования и исследования этого выбора является моделирование. Построение детерминированных имитаторов позволяет с любой детализацией выявить целесообразные технические решения и обосновать язык системы. Целью стохастических моделей является оценка эффективности различных архитектур на основе полноты полезной загрузки оборудования.

На основе традиций разработки многопроцессорных симметричных вычислительных систем можно сделать вывод о практике и тенденции развития вычислительного ресурса второго уровня.

Все модели семейства МВК «Эльбрус» предполагают наличие в составе АЛУ процессора нескольких исполнительных устройств (ИУ), специализированных по типам операций. Тогда в целом для ВС можно сказать, что вычислительный ресурс второго уровня является *распределенным и неоднородным* ([рис.6.10](http://www.intuit.ru/department/hardware/paralltech/2/3.html#image.2.2)).

УУ

ИУ1

ЦП 1

. . . . .

ЦП N

УУ

ИУ*n*

ИУ1

ИУ*n*

Рис.6.10. ВС с распределённым решающим полем

Выше говорилось, что использование ресурса второго уровня неотделимо от общих идей функционирования процессоров ВС — от их архитектуры и от архитектуры ВС в целом. Поэтому сказанного о распределенном ресурсе недостаточно, надо говорить и о способе его использования.

Так, в МВК «Эльбрус-2» применена динамическая загрузка ИУ в процессе выполнения последовательности безадресных команд программы, которую подробно рассмотрим в лекции 3. Обобщенный алгоритм такой загрузки основан на промежуточном переводе безадресных команд в трехадресные. Появление адресов аргументов и результатов в каждой команде позволяет на основе совместного анализа нескольких команд, представленных в "окне просмотра", выявлять их частичную упорядоченность и выделять независимые команды для одновременного выполнения. Это — адресный метод распараллеливания, который рассматривался в архитектуре *dataflow*. Команды проходят стадии обработки, как показано на [рис.Y.3](http://www.intuit.ru/department/hardware/paralltech/2/3.html#image.2.3).

Однако проект МВК «Эльбрус-3», породивший микропроцессорное воплощение — МВК «Эльбрус-3М», — основан на использовании идеи «длинного» командного слова и управления каждым тактом системы. Динамическое распределение работ между ИУ заменено статическим — предписанием каждому ИУ, что он должен начать делать в данном такте. В «длинном» командном слове, в соответствующих позициях, записаны инструкции каждому ИУ.

Дополнение снизу «окна просмотра» новой командой

Исключение из «окна просмотра» выполненных команд

Формирование для новой команды признака возможности выполнения на основе совместного анализа адресов команды и адресов выше находящихся команд

Выборка из «окна просмотра» команд, готовых к выполнению, и назначение их на свободные ИУ

Рис. 6.11. Схема оптимизатора-компоновщика «длинных» командных слов

Это означает, что решение проблем оптимального использования ИУ, их синхронизации при выполнении данного алгоритма возлагаются на оптимизирующий транслятор. Он фактически производит диспетчирование, оптимальное планирование параллельного вычислительного процесса на одном процессоре.

**6.6. Способы распараллеливания**

Различают два основных способа распараллеливания: *по управлению* и *по информации*.

**Первый способ** — представление алгоритма задачи в виде частично-упорядоченной последовательности выполняемых работ. Затем в результате диспетчирования реализуется оптимальный план выполнения работ в ВС при ограничениях на время выполнения всего алгоритма или за минимальное время.

Основой является представление алгоритма граф-схемой G, отражающей информационные связи между работами (задачами, процессами, процедурами, операторами, макрокомандами и т.д.), на которые разбит алгоритм. Граф G — взвешенный, ориентированный, без контуров.

Для исследования графа и диспетчирования используют матрицы следования S; их дополняют столбцом T весов — получают расширенные матрицы следования S\* ([рис. 6.12](http://www.intuit.ru/department/hardware/paralltech/2/4.html#image.2.4)).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | T |
| 1 |  |  |  |  |  |  | 2 |
| 2 |  |  |  |  |  |  | 1 |
| 3 |  |  |  |  |  |  | 2 |
| 4 |  |  |  |  |  |  | 3 |
| 5 |  |  |  |  |  |  | 2 |
| 6 | 1 |  | 1 |  |  |  | 2 |

Рис. 6.12.  Исходная информация для распараллеливания

Здесь предполагаем, что ВС — однородная, с общей (разделяемой) памятью, т.е. потерями времени на обмен между работами можно пренебречь.

Пусть ВС содержит два процессора (*n* = 2). Тогда в результате оптимального распределения получим план ([рис. 6.13](http://www.intuit.ru/department/hardware/paralltech/2/4.html#image.2.5)).

0

2

5

6

1

1

3

4

2

1

2

3

4

5

6

7

*t*

Рис.6.13.  Временная диаграмма параллельного выполнения работ

План действительно совпадает с оптимальным, т.к. длина расписания T = 7, что совпадает с длиной критического пути в графе, Tкр = 7 (путь 1 mhtml:file://F:\INTUIT_ru%20Курс%20Архитектура%20__%20Лекция%20№2%20Микропроцессорные%20системы%20и%20способы%20распараллеливания4.mht!http://www.intuit.ru/img/symbols/srarr.gif3 mhtml:file://F:\INTUIT_ru%20Курс%20Архитектура%20__%20Лекция%20№2%20Микропроцессорные%20системы%20и%20способы%20распараллеливания4.mht!http://www.intuit.ru/img/symbols/srarr.gif4).

В общей схеме организации параллельного вычислительного процесса мы не полностью раскрыли содержание блока 3 — интерпретации потока макроинструкций в виде, удобном для работы диспетчера. Сейчас мы определили, что такой вид — это матрица следования. Значит, в случае необходимости автоматического формирования матрицы следования надо определять информационную взаимосвязь макроинструкций в пределах видимости, т. е. в «окне просмотра». Таким образом, по текущему содержимому «окна просмотра» надо формировать текущий вид матрицы следования.

Вспомним, что мы уже в упрощенном виде решали подобную задачу, например, когда по формируемому потоку трехадресных команд определяли их информационную взаимосвязь и определяли возможность одновременного выполнения этих команд.

Обобщим эту задачу.

Возвращаясь к названной схеме, представим себе, что поток макроинструкций (блок 2) следует через «окно просмотра» так, что для планирования оптимальной загрузки процессоров диспетчер может анализировать некоторое множество этих макроинструкций и из них выбирать вариант назначения их на процессоры для выполнения. Каждая макроинструкция может интерпретироваться и как процедура, где можно выделить имя θμ множество {mhtml:file://F:\INTUIT_ru%20Курс%20Архитектура%20__%20Лекция%20№2%20Микропроцессорные%20системы%20и%20способы%20распараллеливания4.mht!http://www.intuit.ru/img/symbols/alpha.gifμ} входных параметров, множество {βμ} выходных параметров. На [рис. 6.14](http://www.intuit.ru/department/hardware/paralltech/2/4.html#image.2.6) отображено «окно просмотра», через которое следует поток макроинструкций.

m



«окно просмотра»

Рис. 6.14.  Обработка «окна просмотра»

Составим по его содержимому соответствующую матрицу следования размерности *m* × *m*:

По матрице следования *S* диспетчер производит назначение.

После выполнения макроинструкций они исключаются из "окна прросмотра", оставшиеся макроинструкции уплотняются вверх, а снизу "окно просмотра" пополняется новыми макроинструкциями. С учетом вновь поступившихмакроинструкций уточняется текущий вид матрицы следования *S* и процесс диспетчирования продолжается.

По такой же схеме, а именно, на основе первого способа распараллеливания — по управлению — решается другая важная задача распараллеливания: *компоновки длинных командных слов в оптимизирующем трансляторе*. Назначение работы на ИУ осуществляется здесь в виде записи соответствующей инструкции в позицию длинного командного слова, соответствующую ИУ. т.е. план параллельного выполнения работ (команд, операций) фиксируется в длинных командных словах, в которых предусмотрены инструкции каждому ИУ, которые они должны начать выполнять с данного такта.

**Второй способ распараллеливания** — по информации — используется тогда, когда можно распределить обрабатываемую информацию между процессорами для обработки по идентичным алгоритмам (по одному алгоритму)

1. **Рассмотрим задачу умножения матриц:**

Развернем матрицу — результат *С* — в линейный (одномерный) массив, переименуем ее элементы и заменим два индекса на один:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | … |  |  | … |  |  | … |  |
|  |  | … |  |  | … |  |  | … |  |

Пусть ВС содержит *n* процессоров. Выберем следующий план счета элементов матрицы *C*:

процессор 1 считает элементы *d*1, *d*1+*n*, *d*1+2*n*, ...

процессор 2 считает элементы *d*2, *d*2+*n*, *d*2+2*n*, ...

…………………………………………………………..

процессор *n* считает элементы *dn*, *d*2*n*, *d*3*n*, ...

По-видимому, все они будут выполнять одну и ту же программу, но обрабатывать разные наборы данных. (Мы снова столкнулись с целесообразностью *SPMD*-технологии.)

Здесь не потребовалась какая-либо синхронизация параллельного вычислительного процесса.

**2. РАССМОТРИМ ЗАДАЧУ СЧЕТА СПОСОБОМ «ПИРАМИДЫ».**

Эту задачу мы исследовали при рассмотрении ВС типа *SPMD*. Посмотрим еще раз, какая синхронизация нам здесь потребуется.

Пусть необходимо перемножить все элементы некоторого массива {*a*1,*a*2,... , *a*10}. Каждый элемент занимает одну ячейку памяти. Пусть число процессоров в ВС *n*=4. Чтобы распараллелить этот процесс, примем схему счета «пирамидой» ([рис. 6.15](http://www.intuit.ru/department/hardware/paralltech/2/5.html#image.2.7)).

Рис. 6.15.  Граф-схема выполнения операции «свёртки»

Количество уровней операций в ней ]log2 *m*[=]log210[=4 (]*x*[ — ближайшее целое, не меньшее *x*).

Расширим массив, дополнив его ячейками, в которых будем хранить промежуточные частные произведения. Тогда весь план счета примем таким, как показано на [рис. 6.16](http://www.intuit.ru/department/hardware/paralltech/2/5.html#image.2.8). Отмечены процессоры, выполняющие указанную операцию.

Рис. 6.16.  Схема выполнения операции свёртки четырьмя процессорами

Следовательно, надо так написать программу, одну для всех процессоров, предусмотрев необходимую переадресацию для выборки и вычисления «своих» данных, чтобы по ней выбирались два соседних элемента этого удлиненного массива, а результат их умножения отправлялся в очередную ячейку этого «удлинения».

Возникает только одна трудность: для первых пяти произведений данные есть, а вот последующие произведения должны выполняться тогда, когда для них будут найдены исходные данные.

Значит, процессоры, которым выпало произвести такие умножения, должны «уметь» обнаруживать отсутствие данных и дожидаться их появления. Т. е. требуется синхронизация процессоров по использованию общих данных.

Здесь распараллеливание по данным смыкается с распараллеливанием по управлению.

Возможная схема общей для всех процессоров программы — на [рис.6.17](http://www.intuit.ru/department/hardware/paralltech/2/5.html#image.2.9). Она реализована в примере для типа *SPMD.*

Общая синхронизация ВС

Запрет ображения по считыванию к ячейке, в которую будет помещен результат операции

Выполнение операции, запись результата в «закрытую» ячейку, снятие запрета на использование этой ячейки

Переадресация или выход

ВС

|  |
| --- |
| Рис. 6.17 Конвейеры операций  Выполнение любой операции складывается из нескольких последовательных этапов, каждый из которых может выполняться своим функциональным узлом. Это легко показать на операциях сложения и умножения. Выполнение деления мантисс (порядки вычитаются) чаще всего производятся с помощью вычитания из делимого делителя, сдвига влево полученного остатка, нового вычитания делителя из результата сдвига и т.д. В некоторых ВС находится обратная величина делителя с помощью аппроксимирующих полиномов. Затем делимое умножается на эту величину.  Пусть задана операция, выполнение которой разбито на n последовательных этапов. Пусть *ti* — время выполнения *i*-го этапа. При последовательном их выполнении операция выполняется за время  Выберем *время такта* — величину *t*T = maxt*i*} и потребуем при разбиении на этапы, чтобы для любого *i* = 1, ...,n выполнялось условие *ti* + *t*(*i*+1) mod *n*>*t*T. Т.е. чтобы никакие два последовательных этапа (включая конец и новое начало операции) не могли быть выполнены за время одного такта.  Функциональные узлы, выполняющие последовательные этапы одной операции, целесообразно выстроить в единую конвейерную линию, где устройство, выполняющее некоторый этап, закончив его для операции над одним набором данных, переходило бы в следующем такте к выполнению этого же этапа той же операции для другого набора исходных данных.  Например, на [рис. 6.1](http://www.intuit.ru/department/hardware/paralltech/3/#image.3.1)8 представлен конвейер выполнения операции сложения.  Рис. 6.18.  Выполнение операции сложения на конвейере  Загрузка операндов (этап 1)  Выравнивание порядков (этап 2)  Сложение мантисс (этап 3)  Загрузка операндов (этап 4)  Запись результата (этап 5)  Пусть реализуется поток команд одного процессора или существует доступ к этому устройству нескольких процессоров так, что в каждом такте возможно задание на выполнение сложения новой пары чисел. Тогда временная диаграмма работы конвейера может иметь вид, представленный на [рис. 6.1](http://www.intuit.ru/department/hardware/paralltech/3/#image.3.2)9.  2  1  3  4  5  1  2  3  1  2  3  4  уровень 1  уровень 2  уровень 3  уровень 4  уровень 5  0  *t*T  2*t*T  3*t*T  4*t*T  5*t*T  6*t*T  …  *t*  1  2  3  4  5  6  2  3  1  2  Рис. 6.19.  Схема заполнения конвейера  Максимальное быстродействие процессора при полной загрузке конвейера составляет  Число *n* — *количество уровней конвейера*, или *глубина перекрытия*, т. к. каждый такт на конвейере параллельно выполняются n операций. Чем больше число уровней (станций), тем больший выигрыш в быстродействии может быть получен.  В проекте МВК «Эльбрус-3» АЛУ его ЦП имеет конвейерные ИУ сложения (*n*=5), умножения (*n*=5), деления (*n*=8 для полусловного формата, — 32 разряда, *n*=16 для словного формата). Логические операции также выполняются на конвейере с *n*=2.  Известна оценка  т. е. выигрыш в быстродействии получается в *n*-раз.  Реальный выигрыш в быстродействии оказывается всегда меньше, чем указанный выше, поскольку:   1. некоторые операции, например, над целыми, могут выполняться за меньшее количество этапов, чем другие арифметические операции. Тогда отдельные станции конвейера будут простаивать. 2. при выполнении некоторых операций на определённых этапах могут требоваться результаты более поздних, ещё не выполненных этапов предыдущих операций. Приходится приостанавливать конвейер. 3. поток команд порождает недостаточное количество операций для полной загрузки конвейера.   **6.7. Векторные конвейеры. «Зацепление» векторов**  Наряду с использованием конвейеров для обработки единичных (скалярных) данных, используют так называемые *векторные конвейеры*, единичной информацией для которых являются вектора — массивы данных. Применение векторных конвейеров определило класс ВС — *векторно-конвейерных ВС*, сегодня ещё являющихся основой построения некоторых супер-ЭВМ — ВС сверхвысокой производительности.  Для эффективности векторно-конвейерных ВС (например, для подготовки алгоритмов решения задач на ВС «Электроника-ССБИС») необходима векторизация задач. Это — такое преобразование алгоритма, при котором максимально выделяются (если не вся задача сводится к этому) элементы обработки массивов данных одинаковыми операциями. Сюда входят все задачи, основанные на матричных преобразованиях, обработка изображений, сигналов, моделирование поведения среды и т. д.  В основе векторного конвейера лежит то же самое разбиение операции на уровни или этапы выполнения, но он дополняется средствами аппаратной поддержки, позволяющими по информации о векторах организовать последовательную загрузку конвейера элементами векторов, учитывая их длину.  Пусть необходимо выполнить операцию C = A × B, т. е. c*j* = *aj* + *bj*, *j* = 1, ...,N. Пусть на регистрах СОЗУ записаны вектора A и B. Группа регистров отведена для результатов C. Для управления этим процессом известны дескрипторы векторов DA ,DB ,DC, где Dα = {*a*α, *h*α, N}, α= A,B,C,hα — шаг переадресации. Если загрузка векторов производится всегда в одни и те же регистры АЛУ, то достаточно знать значение N. Может задаваться маска M длиной N, состоящая из нулей и единиц. Каждый элемент M соответствует элементу вектора-результата C. Если элемент *mj* = 1 (логическая переменная), то операция получения c*j* производится, в противном случае соответствующие элементы векторов пропускаются. Это применимо для альтернативного счёта в соответствии со значением логических переменных.  Пусть операция умножения выполняется за три этапа. Тогда можно представить временную диаграмму получения N результатов при предположении о назначении функциональных устройств ([рис. 6.20](http://www.intuit.ru/department/hardware/paralltech/3/#image.3.3))*.*  В составе АЛУ может быть два и более конвейерных устройств, специализированных каждое для выполнения некоторой операции. Тогда возможно и эффективно ***«зацепление» векторов***, иллюстрируемое примером на [рис. 6.](http://www.intuit.ru/department/hardware/paralltech/3/#image.3.4)20 для выполнения сложной операции над векторами: D=A× B+C.  Рис.6.20.  Умножение векторов на конвейере  *N*-3  *N*-2  *N*-1  *N*-2  *N*-1  *N*  *N*  1  2  3  Загрузка очередной пары (*a*,*b*), переадресация или остановка  Первый уровень сумматоров  Второй уровень сумматоров  Третий уровень сумматоров  Запись cj  1  2  3  4  5  6  1  2  3  4  5  1  2  3  4  1  2  1  2  3  4  5  6  …………  *N*  *N*+1  *N*+2  *N*+3  *N*+4  *N*  *N*-1  *N*  Рис. 6.21. «Зацепление» векторов  1  2  *n*x  …….  1  2  *n*x  …….  A  B  C  Здесь два конвейера образовали один, с глубиной перекрытия *n*  =  *n*× + *n*+. Очередной результат умножения немедленно направляется на конвейер сложения, куда параллельно направляется необходимый сомножитель. |
| mhtml:file://F:\INTUIT_ru%20Курс%20Архитектура%20__%20Лекция%20№3%20Распараллеливание%20в%20ВС%20на%20уровне%20исполнительных%20устройств1.mht!http://www.intuit.ru/img/empty.gif |
| **6.8. Выполнение операций на стеке**  В вычислительной технике повсеместно используется такая структура данных, как стек. Его применение при выполнении арифметических и логических операций в арифметическо-логическом устройстве (АЛУ) позволяет реализовать безадресную систему команд, что, в свою очередь, дает возможность минимизировать число обращений к оперативной памяти, осуществить буферизацию при многоуровневой памяти, кодировать большое число команд в одном слове.  Рассмотрим механизмы обработки информации на стеке и формирования безадресных команд.  Прежде всего - об идее применения безадресных команд.  Рассматривая структуру команд, мы можем отметить, что практически команды бывают одно-, двух-, трехадресные. Трехадресная команда, как правило, по законченности соответствует одному оператору, т.к. отображает действие над двумя операндами и результат. Двухадресная команда — это такая команда, где один из адресов подразумевается (например, аккумулятор или сумматор), или результат направляется по одному из указанных адресов. То же касательно одноадресной команды: по числу адресов - число обращений к памяти, не считая считывания самой команды.  Возникает вопрос: можно ли отделить загрузку регистров сверхоперативного запоминающего устройства (СОЗУ) от собственно выполнения команды, сделать эту загрузку опережающей, выполнение команды производить только с использованием подразумеваемых адресов СОЗУ и при этом в целом минимизировать количество обращений к ОП? Ответ приводит к структуре безадресных команд и к выполнению операций на стеке.  ***Стек*** представляет собой множество последовательно пронумерованных регистров СОЗУ или ячеек ОП, снабженное ***указателем вершины стека УВС***, в котором автоматически при записи и считывании устанавливается номер (адрес) последнего занятого регистра — ***вершины стека***. При записи в стек (загрузка стека) слово пишется в следующий по номеру регистр, а УВС увеличивается на единицу. При считывании извлекается слово с адресом, указанным в УВС. Затем УВС уменьшается на единицу. Таким образом, в стеке реализуется правило последний «пришел — первый ушел».  Адреса памяти фигурируют только в командах загрузки стека и записи из стека в память.  Значения выражений можно вычислять полностью безадресным способом.  Команда, состоящая только из кода операции, извлекает из стека один или два операнда, выполняет операцию и заносит результат в стек. На [рис. 6.](http://www.intuit.ru/department/hardware/paralltech/3/2.html#image.3.5)22 приведен пример преобразования стека при выполнении трех команд программы.    Рис. 6.22.  Выполнение команд на стеке  2  2  2  ……  17  3  25  3  ……  4  17  3  2  ……  21  3  25  1  ……  3  25  25  УВС:  УВС:  УВС:  УВС:  2  1  0  0  1  1  3  0  1  0  Команда: Загрузка стека изR, (R)=4  «+»  Записать в A  Вычисления с использованием стеков удобно задавать, описывать и программировать с помощью ***польской инверсной (бесскобочной) записи*** арифметических выражений ***ПОЛИЗ*** (предложил польский математик Лукашевич в 1929 г.). Запись производится по правилу "читаем арифметическое выражение слева направо и последовательно друг за другом выписываем встречающиеся операнды". Как только окажется, что все операнды некоторой операции выписаны, записываем знак этой операции, условно учитываем вместо операндов и знака операции потенциальное имя ее результата. Это может породить возможность записи новой операции; или же будем продолжать выписывать операнды, если условно учтенный результат не позволяет записать знак операции. Запись на ПОЛИЗ состоит из цепочек имен и цепочек операций.  Например,y := (*k* + (*l* - *mn*))(*a* - *b*) mhtml:file://F:\INTUIT_ru%20Курс%20Архитектура%20__%20Лекция%20№3%20Распараллеливание%20в%20ВС%20на%20уровне%20исполнительных%20устройств2.mht!http://www.intuit.ru/img/symbols/srarr.gify*klmn* × - + *ab* - ×  Последовательность символов в ПОЛИЗ может рассматриваться как программа вычисления значения данного выражения, если под именами понимать команды загрузки стека (кроме случая, когда имени следует знак операции :=, тогда это имя определяет запись из стека), а под знаками операций — безадресные команды, содержащие только коды операций. Удобно ввести команду вида Зп У — записи из стека по адресу У. Тогда мы получим окончательный вид программы  *klmn* × - + *ab* - × Зп Y (3.1)  Не будем отображать УВС, интерпретируя стек магазинной памятью. Тогда схема выполнения данной программы сводится к представленной на [рис. 6.](http://www.intuit.ru/department/hardware/paralltech/3/2.html#image.3.6)23.    Рис. 6.23.  Выполнение программы на стеке  *n*  *m*  *l*  *k*  *m*x*n*  *l*  *k*  ……..  *l*-*mn*  *k*  …….  *k*+(*l*-*mn*)  ……..  *b*  *a*  *k*+(*l*-*mn*)  ……….  *a*-*b*  *k*+(l-*mn*)  ………..  результат  ………..  A |
| mhtml:file://F:\INTUIT_ru%20Курс%20Архитектура%20__%20Лекция%20№3%20Распараллеливание%20в%20ВС%20на%20уровне%20исполнительных%20устройств2.mht!http://www.intuit.ru/img/empty.gif |
| mhtml:file://F:\INTUIT_ru%20Курс%20Архитектура%20__%20Лекция%20№3%20Распараллеливание%20в%20ВС%20на%20уровне%20исполнительных%20устройств2.mht!http://www.intuit.ru/img/empty.gif |
|  |
| **7. РАСПАРАЛЛЕЛИВАНИЕ В ВС НА УРОВНЕ ИСПОЛНИТЕЛЬНЫХ УСТРОЙСТВ** |

Очевидно, что одновременно или с перекрытием во времени могут выполняться такие две команды, каждая из которых не использует в качестве операнда результат выполнения другой. Если же команда меняет значение некоторой величины, то предшествующие команды в «окне просмотра» должны раньше выполнения данной команды успеть использовать предыдущее значение. При этом должна соблюдаться последовательность присваивания значений одной величине.

Пусть α*j* — логическая переменная (признак), соответствующая возможности назначения *j*-й команды для выполнения. Тогда в процессе назначения команд на ИУ *j*-я команда может быть назначена для выполнения в соответствии со значением истинности предикатов.

При наличии незанятых ИУ необходимой специализации или при наличии свободных регистров в буферах этих ИУ, команды с признаком α*j*=1 назначаются для выполнения. Так, в АЛУ процессора МВК «Эльбрус-2» каждое специализированное конвейерное ИУ имеет небольшой буфер, в которые записываются назначенные команды.

Назначенные команды снабжаются признаком назначения β*j* и не исключаются из «окна просмотра» до сигнала ИУ о завершении выполнения. Это необходимо для исключения преждевременного начала выполнения других команд, использующих результат данной.

Легко видеть, что в каждой из одновременно выполняющихся команд, на разных ИУ, в данном случае используются адреса СОЗУ, которых нет ни в одной из других команд. Т. е. одновременно работающие ИУ используют непересекающиеся множества адресов. Это исключает конфликты обращения к регистрам СОЗУ, ускоряет работу. В таком случае говорят, что ИУ *полностью сегментированы*. Хотя в общем случае работы на общих вычислительных ресурсах (решающих полях) это необязательно.

**Пример.** Пусть «окно просмотра» вместило в себя весь фрагмент ранее рассмотренной программы. АЛУ содержит необходимые устройства сложения, умножения и деления, а также два одновременно работающих канала обращения к ОП. Последовательное преобразование содержимого «окна просмотра» показано на [рис. 7.](http://www.intuit.ru/department/hardware/paralltech/3/4.html#image.3.9)1.

Рис. 7.1.  Распределение работ между исполнительными устройствами

+

Сч

Сч

×

:

3n

r1

<*m*>

<v>

r4

r3

r1

r2

r5

r6

r3

r4

r5

r6

r1

<*A*>

1

1

1

0

0

0

1

1

1

0

0

0

θ*j*

Сч

Сч

+

Сч

Сч

×

:

3n

A*j*1

<*a*>

<*b*>

r1

<m>

<*n*>

r4

r3

r1

A*j*2

r2

r5

r6

A*j*3

r1

r2

r3

r4

r5

r6

rl

3n

 α

1

1

0

1

1

0

0

0

β

1

1

0

0

0

0

0

0

x

:

3n

r4

r3

r1

r5

r6

r6

r1

<*A*>

1

0

0

1

0

0

:

3n

r3

r1

r6

Q

r1

<*A*>

1

0

1

0

Данный способ распределения команд, потребовавший промежуточного перевода безадресных команд в трехадресные, эффективен в случае многофункционального АЛУ. Универсальные исполнительные устройства, образующие АЛУ, — решающие поля, могут потребовать разработки других способов распределения работ.

Рассматривается проблема параллельного выполнения программы, которая представлена в безадресной системе команд, имитирующей выполнение операций на стеке. Параллельная обработка стека применима к программам счёта арифметических операторов, содержащих условия. Это позволяет значительно сократить количество условных переходов.

**8. ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА СТЕКА И СТАТИЧЕСКОЕ РАСПАРАЛЛЕЛИВАНИЕ В РЕШАЮЩЕМ ПОЛЕ**

**8.1. Подстеки и их взаимодействие**

В процессорах супер-ЭВМ используются многофункциональные АЛУ, состоящие из специализированных по операциям исполнительных устройств. Широкое применение микропроцессоров позволяет реализовать в составе АЛУ *решающие поля* на основе универсальных исполнительных устройств, которые могут выполнять последовательности команд или целые процедуры. Рассмотрим возможность динамического распараллеливания в таких АЛУ при выполнении арифметических операторов программы в безадресной системе команд процессора, воспроизводящей выполнение работ на стеке.

Существуют пути обобщения такой структуры ВС на основе комплектации многопроцессорных ВС на общем вычислительном ресурсе, - решающем поле. Такая структура в наибольшей степени адекватна концепции двух основных уровней распараллеливания: уровня программ и уровня команд.

Процессор в обычном смысле выполняет функции устройства управления - управления выполнением программы. Исполнительные команды выполняются процессорными элементами (ПЭ) решающего поля РП, которые составляют общий вычислительный ресурс ВС. Т. е. РП выполняет заявки процессоров на выполнение операций. При этом порядок использования ПЭ определяется динамически в соответствии с потоком команд программы и с возможностями их параллельного выполнения.

Однако при такой структуре и универсальности ПЭ целесообразно распределять между ними не отдельные команды, а группы команд или законченные операторы. И принимать решение об этом можно на основе анализа бесскобочной записи (польской инверсной записи — ПОЛИЗ) арифметических операторов программы, или, что то же самое, — на основе потока безадресных команд программы. Выясняется, что производить распределение (распараллеливание) можно, минуя промежуточный перевод команд из *безадресной формы* в трехадресную, — непосредственно по тексту программы.

Тогда, если представить, что программа в ПОЛИЗ предполагает ее непосредственное выполнение на стеке, то, следовательно, производится распараллеливание обработки стека. Работа одного стека сводится к параллельной работе нескольких взаимосвязанных *подстеков* данного стека. Каждый подстек реализуется на стеке выделенного для этого ПЭ. Если предположить, что данные находятся в СОЗУ и не подлежат перемещению при организации совместной работы ПЭ, то для их обработки целесообразно организовать в локальной регистровой памяти каждого ПЭ адресные стеки, как это рассматривалось выше.

Процессорный элемент и его адресный стек со своим окружением показан на [рис 8.1](http://www.intuit.ru/department/hardware/paralltech/4/#image.4.1).

Рассмотрим арифметическое выражение

*A* :=*a* - (*b* × (*c* + *d*) - *e* : *f*) : (*g* × *h* × *i*)

Его бесскобочная запись

*Aabcd* + × *ef* : - *gh* × *i* × : - :=.

Указатель вершины стека

Ссылка налевый (или два левых) подстек

Цепочка (список) операций

Адресный стек

Рис. 8.1. Подстек

Запишем программу, произведя очевидное оптимизирующее преобразование, сокращающее количество цепочек имен и операций,

*abcd* + × *ef* : - *ghi* × × : - Зп*А*.

Составим информационный граф G, соответствующий порядку выполнения операций на стеке при счете значения этого выражения ([рис. 8.2](http://www.intuit.ru/department/hardware/paralltech/4/#image.4.2)).

Рис. 8.2.  Граф-схема счёта арифметического выражения

Строить этот граф будем в порядке выполнения операций. Сначала изобразим вершины *a,b,c,d* в соответствии с вызовом их в стек. Затем изобразим вершину +, соответствующую сложению *c* и *d*. Затем — вершину ×, соответствующую умножению *b* на *c*+*d*. Так как цепочка операций закончилась, изобразим вершины *e* и *f* и т.д. В результате последовательных действий развернется граф G, иллюстрирующий параллельную структуру алгоритма счета значения данного выражения. Выделены подструктуры, которые могут выполняться параллельно.

Важный вывод на основе анализа графа: можно независимо (и параллельно) извлекать из памяти величины по всем именам, составляющим цепочки имен в ПОЛИЗ. Т. е. можно их заблаговременно считывать в любой последовательности, в том числе и параллельно. Однако при возможности последовательного обращения к памяти предпочтительным является анализ цепочек имен слева направо, а каждой цепочки — справа налево, т. к. использование величин в операциях производится справа налево. (В нижеследующих примерах загрузка стеков производилась традиционно.)

Тогда разовьем стековый механизм, позволяющий производить распараллеливание вычислений.

По числу цепочек имен (и операций) в ПОЛИЗ сформируем несколько подстеков, каждый из которых имеет свою вершину. Подстеки, сформированные для счета одного выражения, упорядочены так, что каждый из них может обладать *ссылкой*на свой *правый*подстек (кроме последнего). Левый по отношению к данному подстек образуется цепочкой имен, находящейся в бесскобочной записи непосредственно левее цепочки имен, соответствующей данному подстеку. Каждому подстеку ставится в соответствие следующая за цепочкой имен цепочка операций и разрешается ее выполнение. На [рис 8.3, *а*](http://www.intuit.ru/department/hardware/paralltech/4/#image.4.3) показаны сформированные подстеки и их начальная загрузка (не по стековому принципу — последовательно через вершину, а по возможности параллельно или с перекрытием во времени).

Рис. 8.3.  Взаимодействие подстеков

*d*

*c*

*b*

*a*

Ссылка 2

*f*

*e*

Ссылка 3

*i*

*h*

*g*

Ссылки нет

*c*+*d*

*b*

*a*

Ссылка 2

*e*:*f*

Ссылка 3

*h*×*i*

*g*

Ссылки нет

*b*(*c*+*d*)

*a*

Ссылка 2

*e*:*f*

Ссылка 3

*g*×*h*×*i*

Ссылки нет

*b*(*c*+*d*)-*e*:*f*

*a*

Ссылка 3

*g*×*h*×*i*

Ссылки нет

*a*

*б*

*в*

*г*

1

2

3

1

2

3

1

2

3

1

3

+×

:-

× ×:-

×:-

-

×

:-

Определим возможный вариант реализации: подстеки могут заполняться не соответствующими величинами, а их адресами в СОЗУ. Т. е. традиционные приемы использования КЭШ-памяти должны быть применены здесь.

Взаимосвязь подстеков следует из неравенства длин цепочек имен и соответствующих им цепочек операций. Возможны ситуации:

1. *подстек выродился* в свою вершину, но цепочка операций оказалась не исчерпанной при том, что первая из этих операций - двуместная; такой подстек назовем *не полностью вырожденным;*
2. цепочка операций исчерпалась, но на подстеке содержится одна или более величин (или их имен, адресов), — такой подстек назовем *вырожденным;* если на подстеке находится единственная величина, составляющая вершину, то такой стек назовем *полностью вырожденным.*

Если данный подстек не полностью вырожденный, то последующая операция возможна над вершиной этого подстека и вершиной левого подстека, если тот является вырожденным или полностью вырожденным.

Если левый подстек также не полностью вырожденный или выполнение на нем возможных операций не закончилось, то данный подстек находится в состоянии ожидания. Для выполнения операции над вершинами двух подстеков (если правый подстек является вырожденным) правый подстек переводится наданныйподстек так, что его вершина становится новой вершиной данного левого подстека. Ссылка правого подстека заменяет собой ссылку левого подстека. Таким образом, на основе двух подстеков формируется один и появляется возможность дальнейшего выполнения операций из цепочки.

При правильной бесскобочной записи каждый не полностью вырожденный подстек дождется возможности продолжения выполнения операций за счет вырожденного или полностью вырожденного левого подстека. Он, в свою очередь, превратится в вырожденный или полностью вырожденный и пополнит левый не полностью вырожденный, если такой имеется. Окончательное значение данного выражения образуется в вершине единственного оставшегося полностью вырожденного подстека, не содержащего ссылку на правый подстек.

Продолжим рассмотрение примера, для простоты считая одинаковым время выполнения всех операций.

На первом шаге (в первом такте) выполняются операции на всех подстеках ([рис. 8.3, *б*](http://www.intuit.ru/department/hardware/paralltech/4/#image.4.3)). После этого подстек 2 оказывается не полностью вырожденным; в дальнейшем он будет ждать вырождения левого — подстека 1.

После выполнения операций на втором шаге ([рис. 8.3, *в*](http://www.intuit.ru/department/hardware/paralltech/4/#image.4.3)) подстек 1 оказывается вырожденным, его дополняет подстек 2 со своей ссылкой на третий подстек, который становится правым для первого. Подстек 3 оказывается не полностью вырожденным, он ждет вырождения своего левого подстека.

После выполнения операций на третьем шаге ([рис. 4.3, *г*](http://www.intuit.ru/department/hardware/paralltech/4/#image.4.3)) подстек 1 оказывается вырожденным; он дополняется подстеком 3.

После выполнения операций на подстеке 1 на четвертом и пятом шагах получается окончательный результат.

При произвольных временах выполнения операций динамическая картина меняется, но правила взаимодействия подстеков те же.

Таким образом, все подстеки образуют очередь заданий, которые распределяются между операционными или исполнительными универсальными устройствами — ПЭ для выполнения. ПЭ действительно образуют распределяемый ресурс системы, в общем случае — многопроцессорной, и назначаются динамически по необходимости. Реализуется концепция виртуальных исполнительных устройств или процессорных элементов решающего поля.

Усложним пример, введя в рассмотрение арифметические операторы, содержащие условие, а также образующие сложные конструкции с использованием условий.

Пусть необходимо распараллелить счет арифметического выражения

Y:=(*a*+*e*:*f*)× if *a*×lmhtml:file://F:\INTUIT_ru%20Курс%20Архитектура%20__%20Лекция%20№4%20Параллельная%20обработка%20стека%20и%20статическое%20распараллеливание%20в%20решающем%20поле1.mht!http://www.intuit.ru/img/symbols/le.gif*c* then if *l* <*g*×*h* then A else A+*l* else *c*×(*h*+16).

Предварительно необходимо распространить правила формирования ПОЛИЗ для отображения условий. Целесообразно оставлять на месте разграничителиif, then, else. Знак операции, в которой участвует условное выражение, необходимо предпосылать каждому альтернативному оператору.

Процессор, как указывалось ранее, выполняя функции лишь устройства управления, производит последовательный анализ символов программы. Обнаружив цепочку имен, он выделяет ПЭ и настраивает его на обработку этой цепочки и последующей цепочки операций. При нахождении каждой следующей цепочки имен, если анализ производится вне условного оператора, он формирует ссылку «левого» подстека на свой «правый». Однако после формирования последнего (а возможно — единственного) подстека оператора if он формирует ему ссылку на два подстека: первого подстека оператора then и первого подстека оператора else — после их анализа.

Учебное издание

**Кобайло** Александр Серафимович

**Архитектура компьютера**

Конспект лекций

Редактор Ю. Д. Нежикова

Компьтерная верстка

Корректор Ю. Д. Нежикова

Издатель:

УО «Белорусский государственный технологический университет».

Свидетельство о государственной регистрации издателя, изготовителя, распространителя печатных изданий

№1/227 от 20.03.2014

Ул. Свердлова, 13а, 220006, г.Минск