ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Б1.Б.14 Аппаратные средства вычислительной техники

Направление подготовки (специальность) 10.03.01 Информационная безопасность

Профиль образовательной программы Безопасность автоматизированных систем

Форма обучения *Очная*

СОДЕРЖАНИЕ

1. КОНСПЕКТ ЛЕКЦИЙ

1. 1 Лекция №1 (2 часа).

Тема: «Арифметические основы ЭВМ»

1.1.1 Вопросы лекции:

1. Введение в дисциплину.
2. Арифметические основы ЭВМ.
3. Перевод чисел из одной системы счисления в другую.
4. Формы представления чисел в ЭВМ.
5. Кодирование информации в ЭВМ.
6. Арифметические операции с двоичными числами.

1.1.2 Краткое содержание вопросов:

1. Введение в дисциплину

Все современные ЭВМ имеют достаточно развитую систему команд, включающую десятки и сотни машинных операций. Однако выполнение любой операции основано на использовании простейших микроопераций типа сложения и сдвиг. Это позволяет иметь единое арифметико-логическое устройство для выполнения любых операций, связанных с обработкой информации.

2. Арифметические основы ЭВМ

Подобные таблицы можно было бы построить для любой другой арифметической и логической операции (вычитание, умножение и т.д.), но именно данные этой таблицы положены в основу выполнения любой операции ЭВМ. Под знак чисел отводится специальный знаковый разряд. Знак “+” кодируется двоичным нулем, а знак “-” - единицей. Действия над прямыми кодами двоичных чисел при выполнении операций создают большие трудности, связанные с необходимостью учета значений знаковых разрядов:

Таблица 1.

Правила сложения двоичных цифр

Значения двоичных чисел A и B			**Разряд Суммы S_i**	**Перенос в следующий разряд Pi**
a_i	b_i	P_{i-1}		
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- • во-первых, следует отдельно обрабатывать значащие разряды чисел и разряды знака;

- • во-вторых, значение разряда знака влияет на алгоритм выполнения операции (сложение может заменяться вычитанием и наоборот).

Во всех ЭВМ без исключения все операции выполняются над числами, представленными специальными машинными кодами. Их использование позволяет обрабатывать знаковые разряды чисел так же, как и значащие разряды, а также заменять операцию вычитания операцией сложения,

Различают **прямой код (П), обратный код (ОК) и дополнительный код (ДК)**двоичных чисел.

3. Перевод чисел из одной системы счисления в другую

Система счисления - это способ записи чисел с помощью заданного набора специальных знаков (цифр).

Существуют позиционные и непозиционные системы счисления.

В **непозиционных с/с** вес цифры (т.е. тот вклад, который она вносит в общее значение числа) **не зависит от ее позиции в записи числа**. Так, в римской с/с в числе XXXII (тридцать два) вес цифры X в любой позиции равен просто десяти. Для обозначения основных чисел используются знаки:

I V X L C D M

1 5 10 50 100 500 1000

Остальные числа получаются как результат сложения или вычитания основных.

VL(45) CL1(151) MCMXCIII(1993)

В **позиционных с/с вес каждой цифры изменяется в зависимости от ее положения** (позиции) в последовательности цифр, изображающих число. Например, в числе 757,7 первая семерка обозначает 7 сотен, вторая - 7 единиц, третья - 7 десятых долей единиц.

Сама же запись числа 757,7 означает сокращенную запись выражения

$700+50+7+0,7=7*10^{2}+5*10^{1}+7*10^{0}+7*10^{-1}$

Любая позиционная с/с характеризуется своими базисом и основанием.

Базис - совокупность различных знаков или символов, используемых в позиционной с/с для записи чисел.

Основание - это количество различных знаков или символов, используемых для изображения цифр в данной системе.

За основание можно принять любое число - два, три, четыре и т.д. Следовательно, возможно **бесчисленное множество позиционных систем счисления:**двоичная, троичная, четверичная и т.д.

Система счисления	Основание	Базис
2с/с	2	0,1
5с/с	5	0,1,2,3,4
8с/с	8	0,1,2,3,4,5,6,7
10с/с	10	0,1,2,3,4,5,6,7,8,9
16с/с	16	0-9,A,B,C,D,E,F

Запись чисел в каждой из с/с с основанием “g” означает сокращенную запись выражения

$a_{n-1}g^{n-1}+a_{n-2}g^{n-2}+\ldots a_1g^1+a_0g^0+\ldots a_{-m}g^{-m}$,

a_i - цифры с/с;

n и m - число целых и дробных разрядов соответственно.

Например,

$1011,1_2=1*2^3+0*2^2+1*2^1+1*2^0+1*2^{-1}$

$276,52_8=2*8^2+7*8^1+6*8^0+5*8^{-1}+2*8^{-2}$

4. Формы представления чисел в ЭВМ

* двоичная (0,1)

* восьмиричная (0,1,2,3,4,5,6,7)

* шестнадцатиричная (0-9,A,B,C,D,E,F)

Использование 2 с/с в ЭВМ объясняется рядом причин:

* для ее реализации нужны технические элементы с двумя возможными состояниями (есть ток-нет тока, намагничен-ненамагничен и т.п.);

* представление информации посредством двух состояний надежно и помехоустойчиво;

* возможно применение аппарата булевой алгебры для выполнения логических преобразований информации;

* двоичная арифметика значительно проще десятичной;

* двоичные таблицы сложения и умножения предельно просты.

Двоичная таблица сложения Двоичная таблица умножения

0+0=0 0*0=0

0+1=1+0=1 0*1=1*0=0

1+1=10 1*1=1

1+1+1=11 1*1*1=1

Почему используются 8 с/с и 16 с/с?

Двоичная система, удобная для ЭВМ, для человека неудобна из-за громоздкости и непривычной записи.

Перевод чисел из 10 с/с в 2 с/с и наоборот выполняет машина. Однако, чтобы профессионально использовать ЭВМ, следует научиться понимать слово машины. Для этого и разработаны 8 с/с и 16 с/с.

Числа в этих с/с читаются почти так же легко, как и десятичные, тебуют соответственно в три (8с/с) и в четыре (16с/с) раза меньше разрядов, чем в 2с/с (числа 8 и 16 соответственно 3-я и 4-я степени числа 2).

Перевод чисел из 8 с/с и 16 с/с в 2 с/с очень прост: достаточно каждую цифру заменить эквивалентной ей двоичной триадой (тройкой цифр) или тетрадой (четверкой цифр):

Например,

$537,1_8$=101 011 111,011_2

$1A3,F_{16}$=1 1010 0011,1111_2

Чтобы перевести число из 2с/с в 8с/с или 16с/с, нужно разбить влево и вправо от запятой на триады (для 8с/с) или тетрады (для 16с/с) и каждую такую группу заменить соответствующей 8-ричной или 16-ричной цифрой.

$10101001,10111_2$=10 101 001, 101 110_2=$251,56_8$

$10101001,10111_2$=1010 1001, 1011 1000_2=$A9,B8_{16}$

Перевод чисел из 10 с/с в 2с/с, 8 с/с, 16 с/с и др (Р-ую с/с).

Перевод целых чисел:

* Исходное число разделить на основание новой с/с (Р). Остаток от деления записать в новой с/с. Это младшая цифра искомого числа.

* Если частное от деления равно 0, то искомое число получено

* Если частное не равно 0, то разделить его на Р. Остаток от деления записать в новой с/с. Это предыдущая цифра искомого числа.

Число с основанием Р записывается как последовательность остатков от деления в обратном порядке, начиная с последнего.

Перевод правильных десятичных дробей:

Правильную десятичную дробь при переводе необходимо последовательно умножать на основание той системы счисления, в которую она переводится, отделяя после каждого умножения целую часть произведения.

Число в новой с/с записывается как последовательность полученных целых частей произведения.

Умножение производится до тех пор, пока дробная часть произведения не станет равна 0. Это значит, что сделан точный перевод. В противном случае, перевод осуществляется до заданной точности.

Перевод смешанных чисел осуществляется в 3 этапа:

1.Осуществляется перевод целой части числа в новую с/с.

2.Осуществляется перевод дробной части числа в новую с/с.

3.Складываются результаты переводов первых 2-х пунктов.

Перевод чисел из 2 с/с, 8 с/с, 16 с/с и т.д. (из Р-ой с/с) в 10 с/с.

1. Записать переводимое число в виде полинома в старой с/с.

2. В полученном полиноме заменить основание и все коэффициенты числами в новой с/с (10 с/с).

3. Выполнить арифметические действия в новой (10-ой) с/с.

Например,

$9B3_{17}=9*17^2+B*17^1+3*17^0=2791_{10}$

$110101,01_2=1*2^5+1*2^4+0*2^3+1*2^2+0*2^1+1*2^0+0*2^{-1}+$
$+1*^{-2}=32+16+4+1+0,25=53,25_{10}$

5. Кодирование информации в ЭВМ

Кодирование – операция, связанная с переходом от исходной формы представления информации в форму, удобную для хранения, передачи или обработки.

Декодирование – связано с обратным переходом к исходному представлению информации.

В настоящее время существуют разные способы кодирования и декодирования информации в компьютере.

Выбор способа зависит от вида информации, которую необходимо кодировать: текст, число, графическое изображение и т.д.

ЭВМ может обрабатывать информацию, представленную только в числовой форме. Любая другая информация (текстовая, графическая) преобразуется в числовую. Так, например, при вводе текста, каждый символ кодируется определенным числом (существуют специальные таблицы кодировки, наиболее известные и распространенные коды ASCII), а при выводе наоборот, каждому числу соответствует изображение определенного символа.

Восемь двоичных разрядов позволяют закодировать 2^8=256 символов, этого достаточно, чтобы закодировать любую букву, цифру или служебный символ. Нажатие клавиши на клавиатуре приводит к тому, что сигнал посылается в компьютер в виде двоичного числа, которое хранится в кодовой таблице. **Кодовая таблица символов**– это внутреннее представление символов в компьютере. Во всем мире в качестве стандарта принята таблица ASCII (American Standart Code for Information Interchange) – Американский стандартный код для обмена информацией.

Первые 128 символов (от 0 до 127) – это цифры, прописные и строчные буквы латинского алфавита, управляющие символы. Вторая половина кодовой таблицы (от 128 до 255) предназначена для национальных символов (в том числе кириллицы), математических символов и так называемых псевдографических символов, которые используются для рисования рамок.

В разных странах, на разных моделях компьютеров могут использоваться и разные варианты второй половины кодовой таблицы.

Нужно иметь ввиду три особенности алфавита в кодовой таблице и их следствия:

1) прописные и строчные буквы представлены разными кодами, т.е. “А” и “а” – разные объекты;

2) при упорядочивании слов по алфавиту сравниваются между собой десятичные коды букв. Поэтому, чтобы избежать недоразумений, если не указано “нечувствителен к регистру”, используйте только латинский или русский алфавит и только прописные или только строчные первые буквы. Имейте ввиду, что любая цифра “меньше” любой буквы, код латинских букв “меньше” чем русских;

3) Многие латинские и русские буквы имеют визуально неразличимое начертание, но разные коды.

Итак, компьютер способен распознавать только значения бита. Однако он редко работает с конкретными битами в отдельности, а совокупность из 8 битов, воспринимаемая компьютером как единое целое, называется **байтом.**

Вся работа компьютера – это управление потоками байтов, которые устремляются в компьютер с клавиатуры или дисков (или по линии связи), преобразовываются по командам программ, запоминаются временно или записываются на постоянное хранение на магнитный диск, а также выводятся на экран дисплея или бумагу принтера в виде букв, цифр, значков.

6.Арифметические операции с двоичными числами

Сложение

Вообще, сложение чаще всего заменяет все операции в ЭВМ, в том числе и вычитание, поэтому этот пункт будет самым большим, ибо нам надо разобрать все возможные случаи. Для примера будем брать 2 числа: A и B, которые будут менять свои значения в зависимости от примера. Стоит заметить, что все операции сложения чаще всего ведутся в обратных кодах. Не забываем, что у положительного числа все коды одинаковы.

Основные принципы сложения:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 10

В последнем случае при сложении в столбик мы пишем 0, а 1 запоминаем. Все, как и в обычном сложении.

Сложение в обратных кодах

Случай 1. A и B положительные числа.

При суммировании таких чисел просто складываются их обратные коды (которые равны прямым), включая знаковый разряд. Учитывая тот факт, что у положительных чисел знаковый разряд равен нулю, у суммы он также равен нулю, следовательно, число получается положительное.

$13_{10} = 0000\ 1101_2$ (ПК/ОК)

$23_{10} = 0001\ 0111_2$ (ПК/ОК)

```
  0000 1101
+ 0001 0111
-----------
  0010 0100
```

$(13+23)_{10} = 0010\ 0100_2$ (ПК/ОК) $= 36_{10}$

Случай 2. A — положительное, а B — отрицательное, по модулю больше A.

В этом случае мы просто складываем числа A и B. Результат получаем, как обычно, в обратном коде:

$13_{10} = 0000\ 1101_2$ (ПК/ОК)

$-23_{10} = 1001\ 0111_2$ (ПК) $= 1110\ 1000_2$ (ОК)

```
  0000 1101
+ 1110 1000
-----------
  1111 0101
```

$(13 + (-23))_{10} = 1111\ 0101_2$ (ОК) $= 1000\ 1010_2$ (ПК) $= -10_{10}$

Случай 3. A — положительное, а B — отрицательное, по модулю меньше A.

На первый взгляд, кажется, что этот случай не отличается от рассмотренного выше второго, однако, давайте посмотрим, что же получится в итоге:

$-13_{10} = 1000\ 1101_2$ (ПК) $= 1111\ 0010_2$ (ОК)

$23_{10} = 0001\ 0111_2$ (ПК)

```
  1111 0010
+ 0001 0111
-----------
1 0000 1001
```

Обратите внимание на результат. У нас выходит, будто сумма — отрицательное число (ибо в знаковом разряде единица), причем, цифр также больше. И, даже если бы не будем учитывать знак, а просто переведем модуль, то получим 9_{10}, а не 10_{10}, как должно быть.

Компьютер решает эту проблему, прибавляя получившуюся «лишнюю» единицу к получившемуся числу. Выглядит это следующим образом:

```
  0000 1001
+ 0000 0001
-----------
  0000 1010
```

Как видите, в результате мы получили положительное число $0000\ 1010_2$, переведя которое в десятичную систему счисления мы получим 10_{10}. А это и есть правильный ответ.

Случай 4. A и B отрицательные числа.

При таких условиях нам точно так же, как и в третьем случае, придется прибавлять единицу из знакового разряда к основному числу.

Проверим:

$-13_{10} = 1000\ 1101_2$ (ПК) = $1111\ 0010_2$ (ОК)

$-23_{10} = 1001\ 0111_2$ (ПК) = $1110\ 1000_2$ (ОК)

```
  1111 0010
+ 1110 1000
-----------
1 1101 1010
```

```
  1101 1010
+ 0000 0001
-----------
  1101 1011
```

Переведем получившийся обратный код в десятичное число: $1101\ 1011_2$ (ОК) = $1010\ 0100_2$ (ПК) = -36_{10}. Как мы видим, ответ получился верным.

Однако, не всегда сложение проходит так гладко. Если вдруг числа у нас достаточно большие, то может возникнуть переполнение, то есть, результат не будет помещаться в выделенное для него место в памяти. Как вам понять, что возникло переполнение? Конечно, можно перевести число в десятичную систему счисления и проверить, больше ли модуль этого числа, чем максимально возможный. Однако, есть и другой способ. Если знак результата получается не таким, какой вы ожидали (не такой, как в примерах выше), значит, произошло переполнение.

К слову, «максимально возможный модуль» для чисел со знаком равен 2^{n-1}, где n — количество бит, отведенных для хранения числа. Напомню, что в 1 байте 8 бит, в 2 байтах 16 бит и так далее. Получается, что если на хранение числа у вас отводится один байт, то максимально возможный модуль это $2^{8-1} = 2^7 = 128$. Если число у вас получается по модулю больше, значит, для операции требуется выделить на слагаемые больше памяти.

Отмечу, что переполнение может возникнуть только в том случае, когда оба числа положительны или отрицательны, ибо, когда знаки у них разные, модуль результата будет меньше как минимум одного слагаемого.

Случай 5. Переполнение, когда A и B положительные.

$100_{10} = 0110\ 0100_2$ (ПК/ОК)

$30_{10} = 0001\ 1110_2$ (ПК/ОК)

```
  0110 0100
+ 0001 1110
-----------
  1000 0010
```

Заметьте, что, складывая два положительных числа, мы получили отрицательное. Следовательно — переполнение. Решается это, как я уже говорил, расширением диапазона:

$100_{10} = 0000\ 0000\ 0110\ 0100_2$ (ПК/ОК)

$30_{10} = 0000\ 0000\ 0001\ 1110_2$ (ПК/ОК)

```
  0000 0000 0110 0100
+ 0000 0000 0001 1110
--------------------
  0000 0000 1000 0010
```

Обратите внимание, что теперь в старшем разряде у нас 0, а значит, получили мы положительное число. И, естественно, результат верен: $0000\ 0000\ 1000\ 0010_2 = 130_{10}$.

Случай 6. Переполнение, когда A и B — отрицательные.

Тут та же история, что и в пятом случае. Убедимся в этом:

$-100_{10} = 1110\ 0100_2$ (ПК) = $1001\ 1011_2$ (ОК)

$-30_{10} = 1001\ 1110_2$ (ПК) = $1110\ 0001_2$ (ОК)

```
  1001 1011
+ 1110 0001
-----------
1 0111 1100
```

Как обычно, убираем лишнюю единицу, прибавляя ее к результату:

```
  0111 1100
+ 0000 0001
-----------
  0111 1101
```

Итог: положительное число. Что делаем? Расширяем диапазон!

$-100_{10} = 1000\ 0000\ 0110\ 0100_2$ (ПК) = $1111\ 1111\ 1001\ 1011_2$ (ОК)

$-30_{10} = 1000\ 0000\ 0001\ 1110_2$ (ПК) = $1111\ 1111\ 1110\ 0001_2$ (ОК)

```
  1111 1111 1001 1011
+ 1111 1111 1110 0001
---------------------
1 1111 1111 0111 1100
```

```
  1111 1111 0111 1100
+ 0000 0000 0000 0001
---------------------
  1111 1111 0111 1101
```

На этот раз получается отрицательное число. Значит, все верно.

Сложение в дополнительных кодах

Случай 1. A и B — положительные числа.

В этом случае ничего не поменяется, ибо, как вы помните, у положительного числа прямой, обратный и дополнительный коды совпадают.

Случай 2. A — положительное, а B — отрицательное, по модулю больше A.

В этом случае мы получим верный результат, но уже в дополнительном коде. Оно и логично, складываем-то мы в дополнительном 🙂

Случай 3. A — положительное, а B — отрицательное, по модулю меньше A.

В случае с обратными кодами мы прибавляли лишнюю единицу к полученному коду. Здесь же нам придется ее просто отбрасывать. Давайте посмотрим:

$-13_{10} = 1000\ 1101_2$ (ПК) = $1111\ 0010_2$ (ОК) = $1111\ 0011_2$ (ДК)

$23_{10} = 0001\ 0111_2$ (ПК)

```
  1111 0011
+ 0001 0111
-----------
1 0000 1010
```

$0000\ 1010_2$ — это уже 10_{10}, а раз мы получили правильный ответ, то нам не нужна лишняя единица. Мы просто отбрасываем ее.

Случай 4. A и B — отрицательные.

Здесь у нас два отрицательных числа, и оба в дополнительном коде, то есть, по логике вещей, результат у нас итак должен получиться в дополнительном коде.. но ведь у нас еще и единица там дополнительная, помните? Давайте взглянем, что же получится:

$-13_{10} = 1000\ 1101_2$ (ПК) = $1111\ 0010_2$ (ОК) = $1111\ 0011_2$ (ДК)

$-23_{10} = 1001\ 0111_2$ (ПК) = $1110\ 1000_2$ (ОК) = $1110\ 1001_2$ (ДК)

```
  1111 0011
+ 1110 1001
-----------
1 1101 1100
```

Взглянем на число, не учитывая «лишнюю единицу»: $1101\ 1100_2$ (ДК) = $1101\ 1101_2$ (ОК) = $1010\ 0010_2$ (ПК) = -36_{10}. Заметили? И без единицы у нас снова получился правильный ответ, поэтому мы просто отбрасываем единицу, как и в предыдущем примере, и ничего более не делаем.

Пятый и шестой случаи — переполнение — для дополнительного кода аналогичны обратному.

Стоит заметить, что перевод в дополнительный код для компьютера дольше, чем в обратный, ибо там не нужно прибавлять единицу. Однако, как вы смогли заметить,

отрицательные складывать легче в дополнительном коде, ибо нет лишнего прибавления единицы.

Итог: *Стоит запомнить, что во всех случаях, кроме особого (случай 4, дополнительный код), если у вас получается лишняя единица — вы прибавляете ее к остальной части числа. И не забывайте про то, что если результат у вас получается не того знака, который вы ожидали — это переполнение.*

Вычитание

Любое вычитание сводится к сложению двух чисел, поэтому я не буду сейчас здесь разбирать все случаи подробно, а просто напишу про замену вычитания сложением.

Случай 1. A и B — положительные.

$A - B = A + (-B)$

В итоге получаем сложение мы заменяем на вычитание, остается только проверить, меньше или больше абсолютная величина (модуль) B, чем A, для того, чтобы выбрать правильный способ решения.

Случай 2. A — положительное, B — отрицательное.

Не имеет значения, какое B по модулю, больше или меньше A.

$A - (-B) = A + B$

Это тот случай, при котором возможно переполнение. Значит, если вы получаете отрицательное число в этом случае — переполнение.

Случай 3. A — отрицательное, B — положительное.

$-A - B = -A + (-B)$

Получаем сложение двух отрицательных чисел. Это тоже случай, при котором возможно переполнение.

Случай 4. A и B — отрицательные числа.

Получаем ситуацию аналогичную первой:

–A – (–B) = –A + B

Смотрим на модули чисел A и B, и решаем, как именно будем вычислять.

Умножение

Вообще, умножение сложная операция, и специально для нее в компьютере имеется регистр, который называется «накапливающий сумматор» и с помощью последовательности сдвигов и сложений получается результат. Однако, мы умножаем по той же логике «в столбик», поэтому так и будем действовать.

Умножение в двоичной системе, на практике, получается быстрее, чем сложение и вычитание, ибо единственная сложность — много складывать, но всё сложение можно производить в прямом коде, а это значит, что нет всяких там «случаев», а все просто и понятно.

Определиться со знаком результата легко. Если знаковые разряды множителей были разными, то получится отрицательное число, то есть, «1» в знаке, иначе — «0». Если говорить еще более «заумно», то над знаковыми разрядами совершается операция сложения по модулю, она же «строгое (исключающее) ИЛИ», она же «XOR».

Попробуем умножить два числа: –13 и 23.

$-13_{10} = 1000\ 1101_2$ (ПК)

$23_{10} = 0001\ 0111_2$ (ПК)

```
  1          1101
× 0         10111
-------------------
            1101
           1101
          1101
         1101
-------------------
  1    100101011
```

Обратите внимание на несколько фактов:

- нули, содержащиеся между знаковым разрядом и первой единицей модуля числа писать нет смысла;
- ряды из нулей (при умножении «0» на первый множитель) тоже писать не надо, вы ведь не пишите их при обычном умножении в столбик? 😀
- полученный результат может занимать памяти больше, чем множители.

В результате у нас получилось число $1\ 100101011_2$, однако, давайте приведем его к нормальному виду: $1\ 1\ 0010\ 1011_2$. Но у нас нет записи числа в 10 цифр, только кратные байтам, то есть: 8 цифр, 16 цифр и так далее. В 8 это число уже никак не поместить, а вот в 16 в самый раз. Значит, что мы делаем? Мы добавляем нули между знаковым разрядом и модулем числа: $1\mathbf{000\ 000}1\ 0010\ 1011_2$. Я выделил зеленым цветом добавленные мною нули.

Однако, давайте для примера возьмем два таких числа, чтобы в двоичной записи их были нули в младших разрядах. Например, перемножим 10 и 40:

$10_{10} = 0000\ 1010_2$ (ПК)

$40_{10} = 0010\ 1000_2$ (ПК)

×	0	101	0
	0	101	000
		101	
		101	
	0	11001	0000

Обратите внимание. При умножении подобных чисел, мы выравниванием их так, дабы младшие разряды, заполненные нулями, оказались справа, а умножаем только значимые части. Нули же потом просто дописываем в конце, столько, сколько их было. Как и при обычном умножении в столбик 😀

Нормализовав ответ, получаем: $0000\ 0001\ 1001\ 0000_2$. Проверим: $256 + 128 + 16 = 400_{10}$. Ответ верен.

В принципе, это все знания, которые вам потребуются для умножения целых чисел в двоичном коде. О дробных поговорим чуть позже.

Деление

Этот раздел, как и в случае с десятичной системой счисления, можно разделить на три подраздела:

- деление без остатка;
- деление с остатком;
- деление с заданной точностью.

Первый случай самый «удачный». Остатка не остается тогда, когда делимое кратно делителю. Для примера возьмем два числа:

$65_{10} = 0100\ 0001_2$

$13_{10} = 0000\ 1101_2$

Разделим их, предварительно отбросив знаковый разряд и лишние, незначимые нули. Знак частного мы получаем так же, как и при умножении: знак делимого XOR знак делителя. В остальном все идентично обычному делению в столбик:

```
 1000001 | 1101
-1101    | 101
   1101
 - 1101
     0
```

Получили ответ: $0000\ 0101_2 = 5_{10}$.

Однако, при делении также возможен остаток. Например, найдем частное и остаток от деления этих чисел:

$39_{10} = 0010\ 0111_2$

$4_{10} = 0000\ 0100_2$

```
 100111 | 100
-100    | 1001
   111
 - 100
    11
```

Заметим, что 11_2 у нас уже не делится на 100_2, ибо оно меньше, чем делитель. Следовательно — это остаток. Теперь у нас два варианта. Или записать ответ как: 0000 1001_2(остаток: 0000 0011_2), или продолжать делить дальше, пока не получим дробь:

```
100111 | 100
-100   | 1001,11
  111
 -100
  110
 -100
   100
  -100
     0
```

В результате получили при делении частное: $1001{,}11_2$. Можете проверить его с помощью перевода в двоичную систему, если считаете нужным 😃 Несмотря на то, что я сейчас пишу про операции над целыми числами, результат получился вещественным. Что ж поделаешь.

К слову, у деления с остатком возможны еще две вариации. Первая — периодическая дробь в частном. Для примера разделим 68_{10} на 5_{10}:

$68_{10} = 0100\ 0100_2$

$5_{10} = 0000\ 0101_2$

```
1000100 | 101
- 101   | 1101,10011001
  111
 -101
   1000
  - 101
     110
    - 101
       1000
      - 101
         110
        - 101
           1000
          - 101
             ...
```

Обратите внимание на голубые и оранжевые части деления. Они повторяются, и, судя по тому, как мы делили, будут повторяться и дальше. Вывод: результат стоит записать с периодом: $1101{,}(1001)_2$

Вторая вариация деления с остатком — деление чисел с разным знаком. Вообще в теории деления натуральных чисел, остаток — число неотрицательное, однако, с целыми числами не все так однозначно. Давайте примем за данность, что у нас остаток может получаться отрицательным, ибо в учебных заведениях, чаще всего, именно так и преподают деление с остатком.

Для деления с остатком справедлива формула:

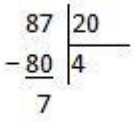
$$\boldsymbol{a = p \times b + q}$$

где:

- $\boldsymbol{a}$ — делимое;
- $\boldsymbol{b}$ — делитель;
- $\boldsymbol{p}$ — частное;
- $\boldsymbol{q}$ — остаток.

Знак остатка *всегда такой же, как и знак делимого!*

Для интереса попробуем разделить два числа в десятичной системе счисления. Пусть первое будет отрицательным, например «−87», а второе положительным, например, «20»:



Теперь немного поразмыслим. При делении отрицательного на положительное мы должны получить отрицательное число, следовательно, частное у нас не «4», а «−4», попробуем подставить эти числа в формулу:

$$-87 = -4 \times 20 + 7$$

Ответ выходит неверным, ведь в правой части получается «−73», а не ожидаемые «−87». Так и должно быть, ведь остаток у нас не такого же знака, как и делимое, следовательно, изменим его:

$$-87 = -4 \times 20 - 7$$

Теперь все верно. Закрепим нашу теорию на практике, поделив, например, «−23» на «13» с остатком. Напомню, что при записи в столбик мы не пишем знаки чисел. Их мы вычисляем потом отдельно.

$13_{10} = 0000\ 1101_2$ (ПК)

$-23_{10} = 1001\ 0111_2$ (ПК)

```
 10111 |1101
-1101  |1
 1010
```

Остаток получился 1010_2. Вычислим знак частного, применив операцию XOR к знакам делимого и делителя:

1 XOR 0 = 1

Выходит, что частное у нас не $0000\ 0001_2$, а $1000\ 0001_2$. Учитывая то, что остаток от деления у нас по знаку должен совпадать с делимым, получим остаток равный: $1000\ 1010_2$.

Выполним проверку:

0000 1101 × 1000 0001 + 1000 1010. Выполним по действиям:

```
  0      1101
× 1         1
-------------
  1      1101
```

Результат умножения: $1000\ 1101_2$. Выполним сложение:

$1000\ 1101_2$ (ПК) = $1111\ 0010_2$ (ОК)

$1000\ 1010_2$ (ПК) = $1111\ 0101_2$ (ОК)

```
  1111 0010
+ 1111 0101
-----------
1 1110 0111
```

```
  1110 0111
+ 0000 0001
-----------
  1110 1000
```

Переведем результат в прямой код и десятичную систему счисления, для проверки:

1110 1000 (ОК) = 1001 0111 (ПК) = -23_{10}

Деление с остатком выполнено верно.

И напоследок хочется сказать про деление с заданной точностью. Вы можете встретить такое задание, как: «вычислить с точностью до шести знаков после запятой» или «с точностью до 0,000001», что, в принципе, одно и то же. Это значит, что вам нужно делить число до седьмого знака после запятой, после чего нужно взглянуть на него и правильно округлить число до шестого знака. То есть, если седьмой знак 0, то просто отсекаете его и все последующее, оставляя результат до шестого знака. Если же 1, то прибавляете 1 к шестому знаку, после чего отсекаете все, что идет после него.

1. 2 Лекция №2 (2 часа).

Тема: «Логические основы построение ЭВМ**»**

1.2.1 Вопросы лекции:

1. Общие сведения о дискретных автоматах.
2. Основные логические функции и элементы.
3. Основные законы (тождества) булевой алгебры.
4. Элементы памяти.

1.2.2 Краткое содержание вопросов:

1. Общие сведения о дискретных автоматах.

Современная техника, особенно такие ее разделы, как вычислительные машины, машины автоматизированного контроля и управления, телемеханика, автоматическая телефония и т.д., базируется на применении так называемых дискретных устройств (автоматов).

В общем смысле дискретным устройством (ДУ) называется техническое устройство, предназначенное для преобразования входной информации, поступающей в виде дискретных сигналов, в выходную дискретную информацию по заранее заданной программе.

Дискретное устройство (ДУ) можно разделить на три основных части:

1. Управляющую часть – это элементы, непосредственно реагирующие на поступающие воздействия извне;
2. Управляемую часть – это элементы или цепи, создающие сигналы на выходах дискретного автомата;
3. Промежуточную часть – это элементы, зависящие как от входных сигналов, так и друг от друга и осуществляющие, как правило, запоминание поступающих сигналов и их последовательностей.

Кроме того, в дискретном автомате можно выделить вспомогательную часть, включающую различные детали, элементы или цепи, которые не влияют на переработку или передачу информации, но обеспечивают правильную работу устройства, согласование отдельных его частей и формирование параметров его сигналов.

В качестве промежуточной части в дискретных устройствах используются специальные элементы, которые способны сохранять свое состояние или выходной сигнал после того, как входной сигнал, вызвавший переход в это состояние, перестал действовать, а для вторичного изменения состояния или возврата в исходное состояние требуется новое входное воздействие. Эти элементы называются элементами памяти, типичными представителями которых являются широко применяемые триггеры типа D, T, JK и SR.

Элементы памяти принято выделять в отдельный блок, который называется блоком памяти или памятью дискретного автомата.

Часть схемы дискретного устройства, не содержащая элементов памяти, называется логическим преобразователем или комбинационным блоком, который помимо формирования выходных сигналов обеспечивает управление блоком памяти. Таким образом, по функциональному признаку логический преобразователь условно можно представить в виде двух отдельных блоков: блока формирования выходных сигналов и блока управления памятью (рис. 1).

Дискретные устройства, не содержащие элементов памяти, могут быть рассмотрены, как частный случай ДА с памятью.

Процесс функционирования ДУ заключается в том, что при подаче на его вход некой последовательности сигналов оно переходит из одного состояния в другое и формирует последовательность выходных сигналов.

Задачей разработчика является правильно описать работу дискретного устройства, для этого необходимо математически задать все эго элементы.

Математической моделью подобных устройств является абстрактный автомат, работающий в некотором идеализированном дискретном времени. Термин “абстрактный” используется в связи с идеализацией времени, а так же абстрагирования от реальной физической природы входных и выходных сигналов, рассматривая их просто как буквы некоторого алфавита.

Абстрактный автомат определяется как шестикомпонентный кортеж , у которого:

1. - множество состояний автомата (алфавит состояний);
2. - множество входных сигналов (входной алфавит);
3. - множество выходных сигналов (выходной алфавит);
4. функция δопределяет состояние автомата в следующий момент времени$t+1$в зависимости от состояния автомата и входного сигнала в момент времениt. Другими словами, функцияδставит в соответствие паре состояние – входной сигнал состояние , в которое он переходит из под действием сигнала , т.е. ;
5. функция выводов λ, вид которой приводит к разделению автоматов на два класса, получивших наибольшее распространение, - автоматы Мили и Мура. В автомате Мили функцияλставит паре состояние – входной сигнал выходной сигнал , т.е. ; в автомате Мура функцияλв соответствие состоянию выходной сигнал , т.е. ;
6. - начальное состояние автомата (), в котором он находится в момент времени$t=0$.

Абстрактный автомат (рис.2) можно представить в виде черного ящика, который имеет один вход, один выход и работает в дискретном времени, принимающем целые неотрицательные значения $t=0,1,2,....$ В любой момент дискретного времени t автомат находится в некотором состоянии , причем в начальный момент времени ($t=0$) он установлен в начальное состояние . Будучи в момент времени t в состоянии , автомат способен воспринять на своем входе сигнал . В соответствии с функцией выходов в этот же момент времени он выдаст выходной сигнал и в следующий момент времени ($t+1$) согласно функции переходов перейдет в состояние . Если на вход автомата, установленного в начальное состояние , подавать некоторую последовательность входных сигналов $z(0), z(1), z(2),...$- входное слово, то на его выходе будут формироваться выходные сигналы $w(0), w(1), w(2)...$ - выходное слово. Относя каждому входному слову соответствующее ему выходное слово, получаем отображение, индуцированное абстрактным автоматом. Комбинационные схемы (КС) так же можно представить в качестве абстрактного автомата с одним внутренним состоянием. Автоматы с числом внутренних состояний более одного составляют класс автоматов с памятью.

И так, что бы задать автомат, необходимо описать все компоненты его кортежа .

Среди многообразия различных способов задания автомата наибольшее распространение получили табличный и графический.

Автомат Мили задан в таблице №1, в которой каждый элемент , записанный на пересечении столбца и строки , определяется следующим образом: . Для указанной таблицы $A=\{a_1,a_2,a_3,a_4\}$, $Z=\{z_1,z_2\}$, $W=\{w_1,w_2,w_3,w_4,w_5\}$.

Автомат Мура задается таблицей №2, в которой каждому столбцу приписаны не только состояние , но и выходной сигнал , соответствующий этому состоянию, где , $A=\{a_1,a_2,a_3,a_4\}$, $Z=\{z_1,z_2\}$, $W=\{w_1,w_2,w_3\}$.

Таблица 1

	a_1	a_2	a_3	a_4
z_1	a_2/ w_1	a_2/ w_1	a_1/ w_2	a_1/ w_4

z_2	a_4/ w_5	a_3/ w_3	a_4/ w_4	a_3/ w_5

Таблица 2

	w_3	w_2	w_3	w_1
	a_1	a_2	a_3	a_4
z_1	a_1	a_3	a_1	a_4
z_2	a_2	a_4	a_4	a_1

Для частных автоматов Мили и Мура в таблицах на месте неопределенных состояний и выходных сигналов ставится прочерк.

Другим способом задания автомата является ориентированный граф, вершины которого соответствуют состояниям, а дуги переходам между ними. Дуга, направленная из вершины в вершину , задает переход в автомате из состояния в состояние . В начале каждой дуги пишется сигнал , который вызывает данный переход. Для графа автомата Мили (рис. 3а) выходной сигнал, формируется на переходе, записывается в конце дуги, а для автомата Мура (рис. 3б) - рядом с вершиной, в которой он формируется. Если переход в автомате производится под действием нескольких входных сигналов, то дуге графа, направленной из в , приписываются все эти входные и соответствующие выходные сигналы.

Следующим этапом проектировки ДУ является переход от абстрактного автомата к структурному, в котором уже учитывается структура входных и выходных сигналов, а так же внутренне устройство автомата на уровне структурных схем. На этом этапе синтеза автомат принято представлять в виде двух частей – памяти и комбинационной схемы (КС). Память автомата состоит из предварительно выбранных элементов памяти – элементарных полных автоматов Мура, которые являются типовыми ДУ, иначе говоря, это совокупность функционально связанных элементов, обеспечивающих реализацию одной или нескольких операций, которые выполняются по специальным командам или непрерывно с момента включения устройства. Наиболее часто используемые элементы памяти, как уже отмечалось, это D, T, JK и SR триггеры.

В отличие от абстрактного автомата, имеющего один вход и один выход, на которые поступают сигналы во входном и выходном алфавитах, структурный автомат (рис. 4) имеет *L* входных и *N* выходных полюсов, на каждом из которых сигнал может принимать два значения 0 и 1. В этом случае каждому входному сигналу абстрактного автомата соответствует некоторый двоичный вектор () и . Для кодирования входных сигналов абстрактного автомата различными двоичными векторами должно быть выполнено условие , где означает ближайшее целое число, не меньшее .

Компоненты вектора () представляют собой набор значений переменных , поставленный во взаимно-однозначное соответствие символу алфавита *Z*. Предположим, что некоторая переменная может принимать в любом этом наборе значения как 0, так и 1, т.е. символ кодируется двумя двоичными векторами () и (). В этом случае будем считать, что закодирован одним троичным вектором (), в котором . Тогда компоненты любого такого вектора могут принимать значения из множества {0,1,-}, причем если , то в соответствующем наборе значений переменных безразлично, равно ли нулю или единице.

Аналогично, сигналу () абстрактного автомата соответствует некоторый двоичный вектор () и . Для кодирования входных сигналов абстрактного автомата различными двоичными векторами должно быть выполнено условие . Как и ранее, допустим, что рассмотренный вектор () может быть троичным и у него .

Идентично кодированию входных и выходных сигналов каждому состоянию () абстрактного автомата в структурном автомате поставим в соответствие двоичный вектор (),. Для кодирования состояний абстрактного автомата различными двоичными векторами должно быть выполнено условие . Допустим, что и в этом случае двоичный вектор () может быть заменен троичным, у которого .

Далее предположим, что каждому переходу абстрактного автомата из состояния в состояние по воздействием входного сигнала с выдачей выходного сигнала соответствует переход структурного автомата из состояния () в состояние () под воздействие входного сигнала () с выдачей выходного сигнала (). Изменение состояний элементов памяти на таком переходе происходит под действием сигналов на входах памяти автомата (рис. 5). Таким образом, после выбора элементов памяти и кодирования символов алфавитов *Z,W* и *A* синтез структурного автомата сводится к синтезу КС, реализующей функции:

г

де - функции обратной связи от памяти автомата к его КС, а - функции возбуждения элементов памяти автомата. Далее остается применить правила и законы минимизации и привести полученные функции к Совершенной ДНФ (СДНФ).

Закодируем состояния, входные и выходные сигналы абстрактного автомата Мили, заданного таблицей №1, следующими кодами:

Заменим состояния, входные и выходные сигналы полученными входами, в результате чего получим совмещенную таблицу (таб. 3) переходов и выходов структурного автомата Мили. Обратите внимание, что в качестве элемента памяти используется D-триггер.

Таблица 3

	T_1T_2			
x_1	00	01	10	11
0	01/ 000	01/ 000	00/ 001	00/ 011
1	11/ 100	10/ 010	11/ 011	10/ 100

Полученной табличной записи соответствует следующая ДНФ:

Следующий этап структурного синтеза построение логической схемы автомата на выбранной базе элементов.

Как правило, в технике используются ДУ большой сложности, условия функционирования которых вообще нельзя выразить на формализованном языке, либо эта формулировка оказывается чрезвычайно громоздкой. Анализируя задачи, решаемые сложными дискретными устройствами, не смотря на огромное их разнообразие не трудно

заметить, что процедура решения одной из них сводится к упорядоченному выполнению ограниченного числа определенных операций, таких, как хранение дискретной информации, ее кодирование, подсчет, суммирование, коммутация, сравнение, преобразование и т.д.

Следствием этого является то, что сложные дискретные автоматы возможно строить на основе сравнительно небольшого числа простых, так называемых типовых дискретных устройств (блоков).

Наиболее интересным представителем типовых ДУ является программируемая логическая матрица (ПЛМ).

Структура этого устройства считается фундаментальной, т.к. на базе ПЛМ были разработаны более универсальные дискретные блоки. В следующем разделе рассмотрим структуру, принципы работы ПЛМ и приведем пример реализации КС на ее базе.

2.Основные логические функции и элементы.

Функция "НЕ", инвертор

Простейшим логическим элементом является инвертор, который просто изменяет значение входного сигнала на прямо противоположное значение. Его функция записывается в следующем виде:

$$F(x) = \bar{x},$$

где черта над входным значением обозначает изменение его значения на противоположное. То же самое действие можно записать при помощи таблицы истинности, приведённой в таблице 2.1. Так как вход у этого логического элемента только один, то его таблица истинности состоит только из двух строк.

Таблица 2.1 – Таблица истинности логического инвертора

В качестве инвертора в простейшем случае можно использовать обычный усилитель с транзистором, включенном по схеме с общим эмиттером или истоком. Схема усилителя, выполненная на биполярном n-p-n транзисторе и позволяющая реализовать функцию логического инвертирования, приведена на рисунке 2.1.

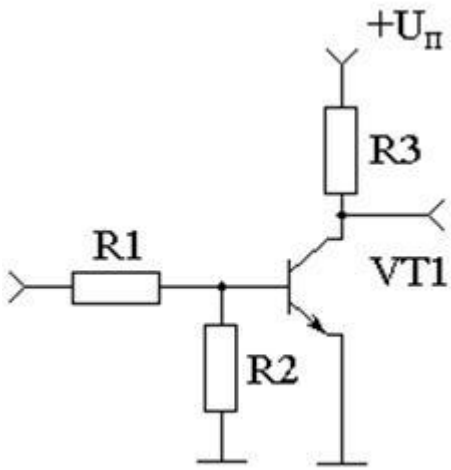


Рисунок 2.1 – Схема, позволяющая реализовать функцию логического инвертирования

Схемы инверторов могут обладать различным временем распространения сигнала и могут работать на различные виды нагрузки. Они могут быть выполнены на одном или на нескольких транзисторах, но независимо от схемы и её параметров они осуществляют одну и ту же логическую функцию.

Для того чтобы особенности включения транзисторов не затеняли выполняемую функцию, для цифровых микросхем введены специальные условно-графические обозначения. Условно-графическое изображение инвертора приведено на рисунке 2.2.

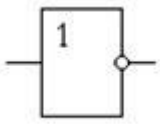


Рисунок 2.2 – Условно-графическое изображение логического инвертора

Функция "И", логическое умножение

Следующим простейшим логическим элементом является схема, реализующая операцию логического умножения "И":

$F(x_1, x_2) = x_1 \wedge x_2$,

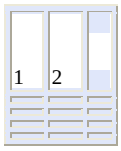
где символ ^ обозначает функцию логического умножения (конъюнкцию). Иногда эта же функция записывается в другом виде:

$F(x_1, x_2) = x_1 \wedge x_2 = x_1 \times x_2 = x_1 \& x_2$

То же самое действие можно записать при помощи таблицы истинности, приведённой в таблице 2.2. В формуле, приведенной выше, использовано два аргумента. Поэтому элемент, выполняющий эту функцию, имеет два входа. Такой элемент обозначается "2И". Для элемента “2И” таблица истинности будет состоять из четырех строк. Количество строк таблицы истинности можно определить по формуле $N = 2^n$, где N — это количество строк в таблице истинности, а n — количество входов логического элемента. В нашем случае $N = 2^2 = 4$.

Таблица 2.2 – Таблица истинности схемы, выполняющей

логическую функцию "2И"



Как видно из приведённой таблицы истинности активный сигнал на выходе этого логического элемента появляется только тогда**,**когда и на входе x_1 и на входе x_2будут присутствовать логические единицы. То есть этот логический элемент действительно реализует операцию "И".

Условно-графическое изображение схемы, выполняющей логическую функцию "2И", на принципиальных схемах приведено на рисунке 2.3, и с этого момента схемы, выполняющие функцию "И", будут приводиться именно в таком виде. Это изображение не зависит от конкретной принципиальной схемы устройства, реализующей функцию логического умножения.

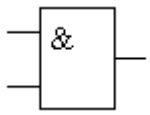


Рисунок 2.3 – Условно-графическое изображение схемы, выполняющей логическую функцию "2И"

Проще всего понять, как работает такой элемент при помощи схемы, построенной на идеализированных ключах с электронным управлением, как это показано на рисунке 2.4. В приведённой схеме ток будет протекать только тогда, когда оба ключа будут замкнуты, а значит, единичный уровень на выходе схемы появится только при подаче на ее вход двух логических единиц.

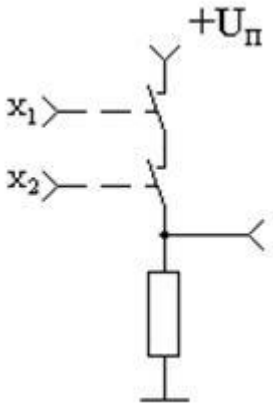


Рисунок 2.4 – Эквивалентная схема, реализующая логическую функцию "2И"

Аналогично описывается и функция логического умножения трёх переменных:

$F(x_1, x_2, x_3) = x_1 \wedge x_2 \wedge x_3$

Её таблица истинности будет содержать уже восемь строк ($2^3 = 4$). Таблица истинности трехвходовой схемы логического умножения "3И" приведена в таблице 2.3, а условно-графическое изображение этого логического элемента на рисунке 2.5. При этом в схеме, построенной по принципу схемы, приведённой на рисунке 2.4, добавляется третий ключ.

Таблица 2.3 – Таблица истинности схемы, выполняющей

логическую функцию "3И"

1	2	3

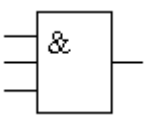


Рисунок 2.5 – Условно-графическое изображение схемы, выполняющей логическую функцию "3И"

2.3 Функция "ИЛИ", логическое сложение

Следующим простейшим элементом является схема, реализующая операцию логического умножения "ИЛИ":

$F(x_1, x_2) = x_1 \acute{U} x_2$,

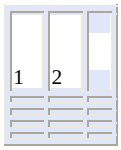
где символ Ú обозначает функцию логического сложения (дизъюнкцию). Иногда эта же функция записывается в другом виде:

$F(x_1, x_2) = x_1 \acute{U} x_2 = x_1 + x_2 = x_1 \mid x_2$

То же самое действие можно записать при помощи таблицы истинности, приведённой в таблице 2.4. В формуле, приведенной выше, использовано два аргумента. Поэтому элемент, выполняющий эту функцию, имеет два входа. Такой элемент обозначается "2ИЛИ". Для элемента "2ИЛИ" таблица истинности будет состоять из четырех строк ($2^2 = 4$).

Таблица 2.4 –Таблица истинности схемы, выполняющей

логическую функцию "2ИЛИ"



Как и в случае, рассмотренном для схемы логического умножения, воспользуемся для реализации схемы логического элемента "2ИЛИ" идеализированными ключами с электронным управлением. На этот раз соединим ключи параллельно. Эквивалентная схема, реализующая таблицу истинности 2.4, приведена на рисунке 2.6. Как видно из приведённой схемы, уровень логической единицы появится на её выходе, как только будет замкнут любой из ключей.

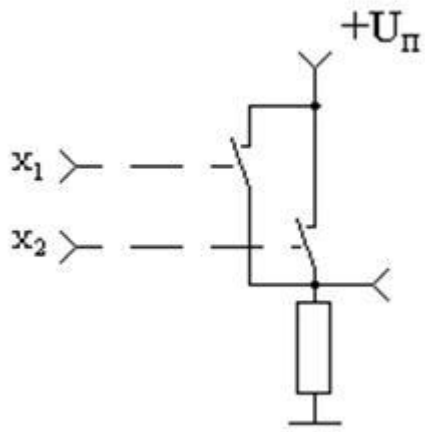


Рисунок 2.6 – Эквивалентная схема, реализующая логическую функцию "2ИЛИ"

Так как функция логического суммирования может быть реализована устройствами, собранными по разным принципиальным схемам, то для обозначения этой функции используется свое условно-графическое обозначение. На условно-графическом изображении логического элемента "ИЛИ" используется специальный символ "1", как это приведено на рисунке 2.7.

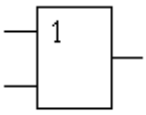


Рисунок 2.7 – Условно-графическое изображение схемы, выполняющей логическую функцию "2ИЛИ"

Подобным образом описывается и функция логического сложения трёх переменных:

$F(x_1, x_2) = x_1 \acute{U} x_2 \acute{U} x_3$

Её таблица истинности будет содержать уже восемь строк (2^3=4). Таблица истинности трёхвходовой схемы логического умножения "3ИЛИ" приведена в таблице 1.5, а условно-графическое изображение на рисунке 2.8. В схеме, построенной по принципу схемы, приведённой на рисунке 2.6, придётся добавить третий ключ.

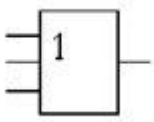


Рисунок 2.8 – Условно-графическое изображение схемы, выполняющей логическую функцию "ЗИЛИ"

Таблица 1.5 – Таблица истинности схемы, выполняющей

логическую функцию "ЗИЛИ"

1	2	3	

Теперь, после того, как мы рассмотрели принципы построения логических элементов, можно подробно остановиться на особенностях реализации логических элементов в различных схемотехнических решениях.

3.Основные законы (тождества) булевой алгебры.

Основными понятиями булевой алгебры являются понятия логической переменной и логической функции.

Логической переменной называется величина, которая может принимать одно из двух возможных состояний (значений), одно из которых обозначается символом “0”, другое – “1” (для обозначения состояний возможно применение и других символов, например, “Да” и “Нет” и др.). Сами двоичные переменные чаще обозначают символами x_1, x_2,… В силу определения логические переменные можно называть также двоичными переменными.

Логической (булевой) функцией (обычное обозначение –*y*) называется функция двоичных переменных (аргументов), которая также может принимать одно из двух возможных состояний (значений): “0” или “1”. Значение некоторой логической функции **n** переменных определяется или задается для каждого набора (сочетания) двоичных переменных. Количество возможных различных наборов, которые могут быть составлены из **n**аргументов, очевидно, равно 2^n . При этом, поскольку сама функция на каждом наборе может принимать значение “0” или “1”, то общее число возможных функций от n переменных равно 2^{2^n} .

Таким образом, множество состояний (значений), которые могут принимать как аргументы, так и функции, равно двум. Для этих состояний в булевой алгебре определяются отношение эквивалентности, обозначаемое символом равенства (=) и три

операции: а) логического сложения (дизъюнкции), б) логического умножения (конъюнкции), в) логического отрицания (инверсии), обозначаемые соответственно символами:

+ или $\vee$ - операция дизъюнкции,

• или $\wedge$ или & - операция конъюнкции,

$\bar{*}$ - операция инверсии (* - символ аргумента или функции).

Постулативно полагается, что при выполнении перечисленных операций отношения эквивалентности имеют вид:

а) 0 + 0 = 0, б) 0 × 0 = 0, в) $\bar{0} = 1$,

(1) 0 + 1 = 1, 0 × 1 = 0, $\bar{1} = 0$.

1 + 0 = 1, 1 × 0 = 0,

1 + 1 = 1; 1 × 1 = 1;

На основании постулатов (1) можно вывести следующие соотношения (законы) алгебры логики:

1. Законы одинарных элементов (универсального множества – а), нулевого множества – б), тавтологии – в)):

(2) а) $x + 1 = 1$, б) $x + 0 = x$, в) $x + x = x$,

$x \times 1 = x$; $x \times 0 = 0$; $x \times x = x$.

2. Законы отрицания (двойного отрицания – а), дополнительности – б), двойственности – в)):

(3) а) $\bar{\bar{x}} = x$ б) $x + \bar{x} = 1$, в) $\overline{x_1 + x_2} = \bar{x}_1 \cdot \bar{x}_2$,

$x \cdot \bar{x} = 0$; $\overline{x_1 \cdot x_2} = \bar{x}_1 + \bar{x}_2$.

3. Законы абсорбции или поглощения – а) и склеивания – б):

(4) а) $x_1 + x_1 \cdot x_2 = x_1$, б) $x_1 \cdot x_2 + x_1 \cdot \bar{x}_2 = x_1$, $x_1 \cdot (x_1 + x_2) = x_1$;
$(x_1 + x_2) \cdot (x_1 + \bar{x}_2) = x_1$.

Законы двойственности (3, в), называемые также законами деМоргана, были обобщены К. Шенноном на случай произвольного (n) числа аргументов.

Кроме законов, перечисленных выше и не имеющих аналогов в обычной алгебре (алгебре чисел), для алгебры логики справедливы законы обычной алгебры: коммутативные или переместительные, дистрибутивные или распределительные, ассоциативные или сочетательные.

Любая логическая функция y n двоичных переменных $x_1, x_2, \ldots, x_n$ может быть задана таблично. Такие таблицы, получившие название **таблиц истинности**, содержат 2^n строк, в которые записываются все возможные двоичные наборы значений аргументов, а также соответствующее каждому из этих наборов значение функции.

4.Элементы памяти.

Основой любого компьютера является ячейка памяти, которая может хранить данные или команды. Основой любой ячейки памяти является функциональное устройство, триггер (или защелка), которое может по команде принять или выдать один двоичный бит, а, главное, сохранять его. Триггер строится на основе базового набора логических схем (рис. 2.1.3.).

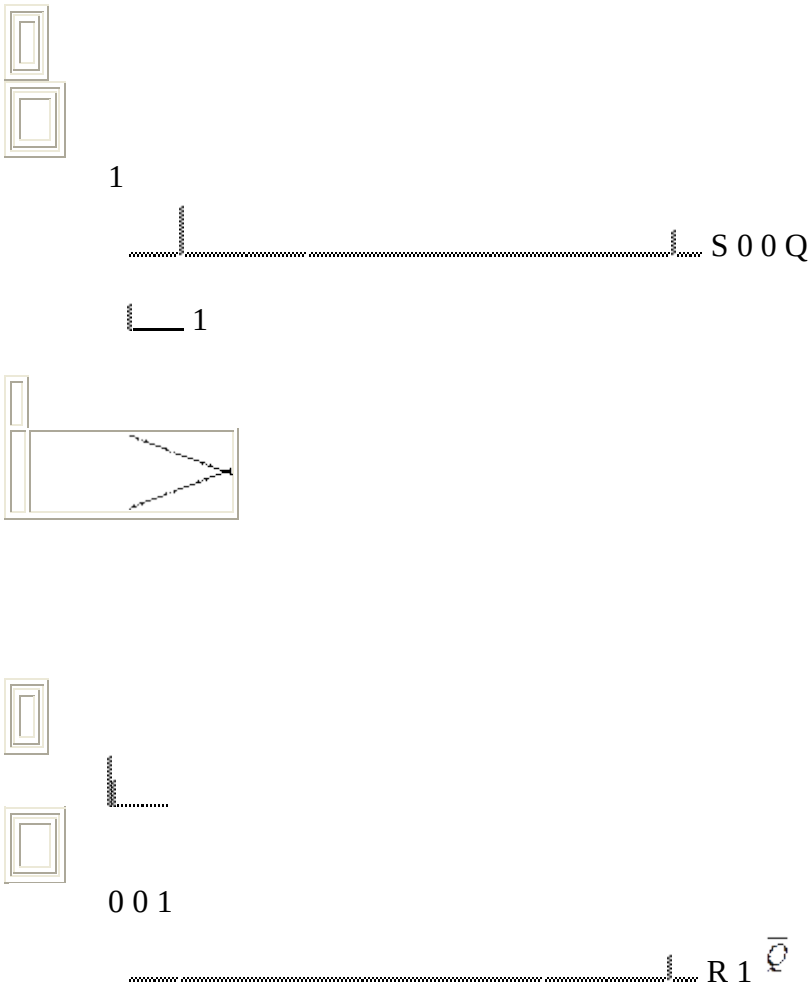


Рис. 3.1.3. Схема триггера в состоянии хранения бита информации.

1 и 2 – это два элемента «логическое НЕ», 3 и 4 – два элемента «логическое И-НЕ», которые представляют собой комбинацию логических элементов «И» и «НЕ». Такой элемент на входе выполняет операцию логического умножения, результат которой инвертируется на выходе логическим отрицанием. Триггер имеет два выхода Q и $\overline{Q}$. Сигнал на выходе Q соответствует значению, хранящемуся в триггере. Выход $\overline{Q}$ используется для получения инверсного значения сигнала. Входы S и R предназначены для записи в триггер одного бита со значением 0 или 1.

Для записи в триггер 1 на вход S подается 1 (рис.2.1.4.). На выходе схемы 1 получится 0, который обеспечит на выходе схемы 3 единицу. С выхода схемы 3 единица поступит на вход схемы 4, на выходе которой значение изменится на ноль ($\overline{1 \cdot 1}$=0). Этот ноль на входе схемы 3 будет поддерживать сигнал на выходе в состоянии единицы. Теперь можно снять единичный сигнал на входе S, на выходе схемы 3 все равно будет высокий уровень, т.е. триггер сохраняет записанную в него 1. Единичный сигнал на входе 3 необходимо удерживать некоторое время, пока на выходе схемы 4 не появится нулевой сигнал. Затем на входе S вновь устанавливается нулевой сигнал, но триггер поддерживает единичный сигнал на выходе Q, т.е. сохраняет записанную в него единицу. Точно также, подав единичный сигнал на вход R, можно записать в триггер ноль.

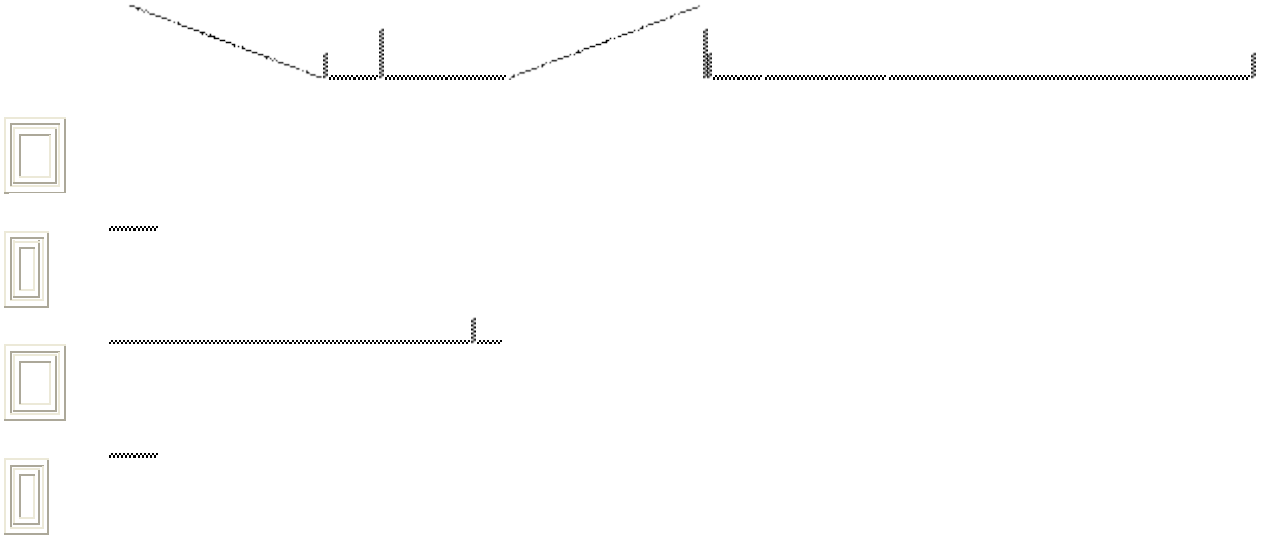


Рис. 3.1.4. Запись в триггер единицы.

1. 3 Лекция №3 (2 часа).

Тема: «**Структуры запоминающих устройств ЭВМ**»

1.3.1 Вопросы лекции:

1. Структуры адресных ЗУ
2. Элементы ЗУ с произвольным обращением
3. Постоянные ЗУ (ПЗУ, ППЗУ)

1.3.2 Краткое содержание вопросов:

1. Структуры адресных ЗУ

В полупроводниковых ЗУ адресного типа можно выделить две основные функциональные части: матрицу запоминающих элементов, обеспечивающую хранение данных, и схему выборки, обеспечивающую запись или считывание информации в выбранной ячейке памяти (рис. 4.2). Матрица ЗЭ – прямоугольная и в ранних образцах ЗУ имела размерность $M = k \cdot m$, где M – информационная емкость памяти в битах; k – число хранимых слов; m – их разрядность. Схема выборки состоит из дешифратора адресного кода ***DC*** и усилителей записи/считывания.

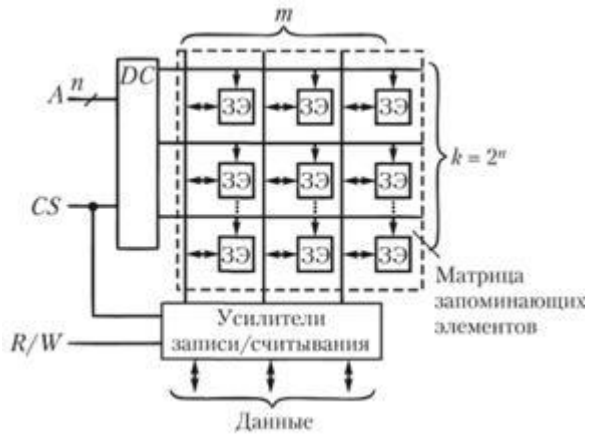


Рис. 4.2. **Структура ЗУ типа *2D***

Каждая строка матрицы ЗЭ соответствует ячейке памяти и хранит одно слово данных, а номера строк соответствуют адресам ячеек. Адрес выбранной ячейки по шине адреса поступает на все микросхемы памяти, но воспринимается только той микросхемой, на которую пришел разрешающий сигнал ***CS (Chip Select*** – выбор кристалла). С приходом сигнала ***CS*** дешифратор адресного кода активизирует одну из выходных линий, разрешая одновременный доступ ко всем элементам выбранной строки. Одноименные разряды всех ячеек образуют столбцы матрицы, соединяемые вертикальными линиями – внутренними линиями данных. Если осуществляется операция чтения, то состояния ЗЭ выбранной строки транслируются по этим линиям, усиливаются и выдаются на внешнюю шину данных. Если же выполняется операция записи, то усилители устанавливают на линиях уровни напряжения в соответствии с записываемыми данными, и ЗЭ переходят в соответствующие состояния "пуля" или "единицы". Подобная структура ЗУ получила название структуры 2D.

Структура 2D может быть использована только в ЗУ малой информационной емкости. С ростом емкости возрастает сложность дешифратора, поскольку число его выходных линий равно числу хранимых слов. Для ЗУ большой информационной емкости структура была усовершенствована, она получила название структуры 2***DM.*** На рис. 4.3 показана такая структура для ЗУ типа ROM. В ней возбужденный выход дешифратора ***DC***X по-прежнему выбирает целую строку матрицы ЗЭ. Однако, в отличие от предыдущей структуры, длина строки не равна разрядности хранимых слов, а многократно ее превышает. Число строк матрицы, а значит, и число выходов дешифратора теперь меньше количества хранимых слов. Для выбора одной из строк служат не все разряды адресного кода, а их часть ***A***n-1,..., ***A***к. Остальные разряды адреса (от ***A***к-1, до A0) используются для того, чтобы выбрать необходимое слово из множества слов, содержащихся в этой строке. Это выполняется с помощью мультиплексоров, на адресные входы которых подаются коды ***A***к***_1,*** ..., ***A***0. Длина строки равна ***m2***к, где ***m*** – разрядность хранимых слов. Каждый отрезок строки длиной ***2***к хранит все одноименные (нулевые, первые, ..., ***m-e)***разряды всех ***2***к слов этой строки. Из каждого такого отрезка мультиплексор выбирает один бит. Количество мультиплексоров соответствует количеству разрядов в слове. Таким образом, на выходах мультиплексоров формируется выходное слово. По разрешению сигнала ***CS,*** поступающего на входы ***OE*** управляемых буферов с тремя состояниями, выходное слово передается на внешнюю шину.

Структура 2***DM*** используется не только для ЗУ типа ROM, но и для ЗУ типа RAM с операциями чтения и записи. В этих ЗУ вместо мультиплексоров используются управляемые буферы данных, выполняющие не только функции мультиплексирования и выдачи данных в шину ***DO,*** но и функции приема данных по шине ***DI*** и формирования сигналов записи для соответствующих ЗЭ. Направление передачи данных определяется сигналом ***R/W,*** а управление буферами данных осуществляется дешифратором.

2. Элементы ЗУ с произвольным обращением

Запоминающие устройства с произвольным обращением

В вычислительной технике в качестве ЗУ с произвольным обращением, используемых в оперативных памятях ЭВМ, применяются **полупроводниковые интегральные ЗУ.**

Полупроводниковые ЗУ имеют ряд важных достоинств: большее быстродействие, компактность, меньшую стоимость, совместимость по сигналам с логическими схемами, общие с другими электронными устройствами ЭВМ технологические и конструктивные принципы построения.

Недостатком полупроводниковых ЗУ с произвольным обращением является их энергозависимость, выражающаяся в том, что они потребляют энергию в режиме хранения информации и теряют информацию при выключении напряжения питания (потери информации можно избежать автоматическим переключением на аварийное питание от аккумуляторов).

По типу ЗЭ различают биополярные ЗУ с биполярными транзисторами (с ТТЛ- или ЭСЛ-схемами) и МОП-ЗУ с МОП-транзисторами. **В биполярных интегральных ЗУ** в качестве ЗЭ используется статический триггер на двух многоэмиттерных транзисторах с непосредственными связями (рис. 4.9).

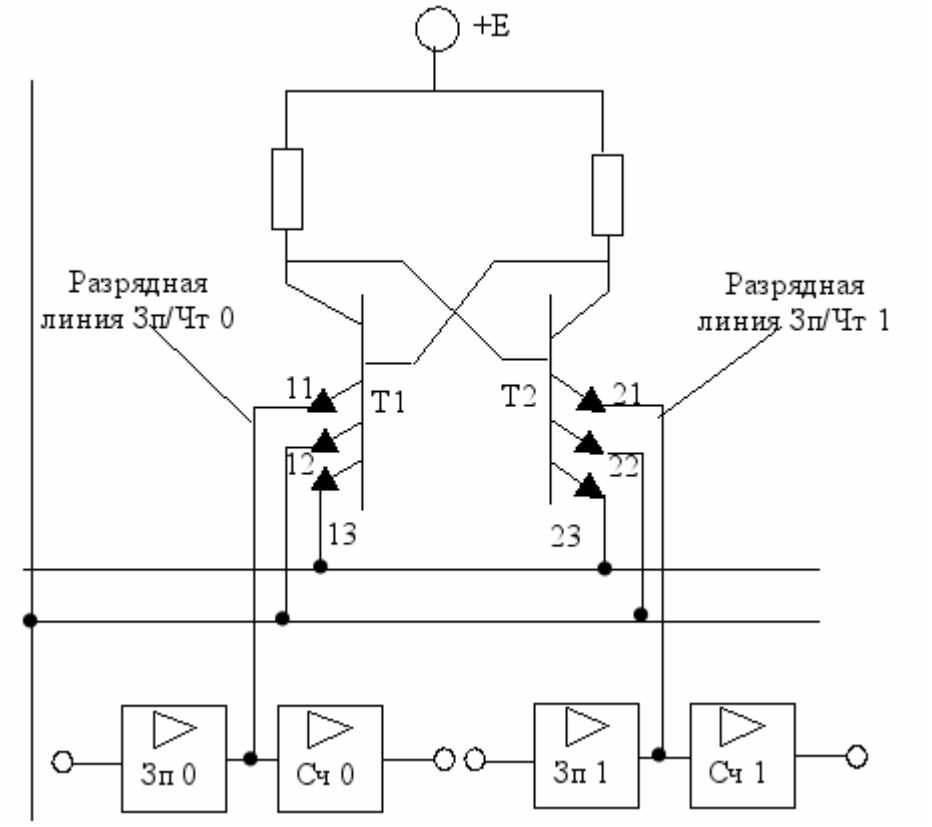


Рис. 4.9. Запоминающий элемент полупроводникового биполярного ЗУ

Эмиттеры 11 и 21 являются парафазными информационными входами ЗЭ и служат для записи в триггер **1** или **0**. Эти же эмиттеры используются как выходы при считывании информации. Адресные эмиттеры 12, 22, 13 и 23 образуют два конъюнктивно связанных входа выборки.

Организация ЗУ из триггеров осуществляется по схеме типа 3D.В режиме хранения (ЗЭ не выбран) эмиттерный ток открытого транзистора замыкается на землю через адресные эмиттеры и адресные линии (или только через один такой эмиттер и одну линию), находящиеся под потенциалом логического **0** (≤0,4 В). При этом информационные эмиттеры должны быть заперты, для чего на них подается потенциал (1-1,5В), который больше потенциала адресных эмиттеров (больше максимального значения уровня сигнала логического **0**, равного 0,4В, но меньше

минимального значения сигнала логической **1**, составляющего 2,4В), с тем чтобы при выборке ЗЭ через информационные эмиттеры протекали токи, необходимые для операций считывания и записи.

При выборке данного ЗЭ на его адресные эмиттеры с выходов адресных дешифраторов подается потенциал логической **1** (≥2,4 В), превышающий потенциал информационных эмиттеров. Поэтому адресные эмиттеры оказываются запертыми, а коллекторный ток открытого транзистора течет через его информационный эмиттер, чем обеспечивается возможность считывания из ЗЭ и записи в него информации.

Состояния **1** и **0** ЗЭ распознаются по наличию тока соответственно в разрядной линии **^ 0** (открыт транзистор T_1) или в разрядной линии **1** (открыт транзистор T_2).

Считывание происходит без разрушения информации. Хранимая в ЗЭ информация доступна для считывания все время, пока ЗЭ находится в выбранном состоянии, и в него не производится запись (отсутствует импульс «разрешение записи»).

При считывании на входы обоих усилителей записи подается потенциал логического **^ 0**, в результате чего на выходах этих усилителей оказывается потенциал логической **1,** запирающий усилители записи и тем самым предотвращающий ответвление в них тока считывания (тока информационного эмиттера).

При считывании ток вытекает из информационного эмиттера открытого транзистора и втекает в базовую цепь входного транзистора соответствующего усилителя считывания, в результате чего выходной транзистор последнего полностью открывается.

Для записи в ЗЭ **1** или **0** с соответствующего усилителя записи на подключенный к нему информационный эмиттер подается потенциал логического **0** (≤0,4 В), а на другой информационный эмиттер продолжает поступать с его невозбужденного усилителя записи потенциал, равный примерно 1,5В.

Если, допустим, производится запись **^ 1** в триггер, находившийся перед этим в состоянии **1** (открыт транзистор T_2), то подача потенциала низкого уровня на эмиттер 21 не меняет состояние триггера. Если до записи триггер находился в состоянии **0**, то при подаче потенциала низкого уровня на эмиттер 21 (запись 1) открывается транзистор T_2, при этом транзистор T_1 закрывается и триггер устанавливается в состояние **1.**

Интегральная микросхема биполярного ЗУ представляет собой кристалл кремния, в котором образованы массив ЗЭ (триггеров) со всеми межсоединениями, а также адресные дешифраторы, усилители-формирователи записи и считывания и другие схемы для управления адресной выборкой, записью и считыванием. Для повышения быстродействия ЗУ эти обслуживающие схемы могут быть выполнены на основе ЭСЛ-элементов, работающих в линейной области, в то время как построенные на основе ТТЛ-элементов триггеры ЗЭ работают с насыщением. В таком случае кристалл содержит схемы согласования уровней сигналов для перехода от схем ТТЛ к схемам ЭСЛ и обратно.

Полупроводниковые ЗУ размещаются в стандартных корпусах интегральных микросхем. Число выводов ограничивают число слов и разрядов запоминающего массива интегральной микросхемы. Для получения ЗУ с большим числом разрядов и (или) слов, чем в запоминающем массиве в корпусе схемы, применяются несколько корпусов.

Данный вид памяти получил в современных ЭВМ название **SRAM** — это сокращение от Static RAM (Статическая оперативная память). Она названа так потому, что в отличие от динамической оперативной памяти (Dynamic RAM — DRAM), для сохранения ее содержимого не требуется периодическая регенерация. Преимущество памяти **SRAM** состоит не только в том, что не требуется регенерация, но и в том, что память SRAM имеет более высокое быстродействие, чем динамическая оперативная память, и может работать на той же частоте, что и современные процессоры.

Однако для хранения каждого бита в конструкции памяти SRAM используется кластер из шести транзисторов. Использование транзисторов без каких-либо конденсаторов означает, что нет необходимости в регенерации. (Ведь если нет никаких конденсаторов, то и заряды не теряются.) Пока подается питание, **SRAM** будет помнить то, что сохранено. Почему же тогда микросхемы **SRAM** не используются для всей системной памяти?

Это происходит потому, что по сравнению с динамической оперативной памятью, память ^ **SRAM** намного быстрее, но плотность ее намного ниже, а цена довольно высокая. Более низкая плотность означает, что микросхемы **SRAM**, имеют большие габариты, хотя их информационная емкость намного меньше. Например, емкость модуля динамической оперативной памяти может равняться 64 Мбайт или больше, в то время как емкость модуля **SRAM** приблизительно того же самого размера равна только 2 Мбайт; стоить он будет столько же, сколько тот же самый модуль динамической оперативной памяти емкостью 64 Мбайт. Таким образом, габариты памяти **SRAM** в среднем в 30 раз больше, и во столько же раз эта память дороже динамической оперативной памяти. Высокая стоимость и большие габариты не позволяют использовать память типа **SRAM** в качестве оперативной памяти в персональных компьютерах.

Даже притом, что память типа **SRAM** слишком дорога для использования в качестве оперативной памяти в персональных компьютерах, разработчики все-таки применяют ее с целью, значительного повышения эффективности ПК. Но чтобы избежать значительного увеличения стоимости, устанавливают только небольшой объем высокоскоростной памяти**SRAM**, которая используется в качестве кэш-памяти.

^ **Динамические МОП-ЗУ** сравнительно дешевы, потребляют небольшую мощность, позволяют достигнуть очень высокой плотности размещения ЗЭ на кристалле и, следовательно, большей емкости в одном корпусе микросхемы. В настоящее время динамические МОП-ЗУ широко используются для построения основной (оперативной) памяти ЭВМ.

В динамических ЗУ двоичные коды хранятся на «запоминающих емкостях», в качестве которых используются паразитные емкости некоторых цепей схем. Примем, что отсутствие заряда на запоминающей емкости означает состояние **0**, а наличие - состояние **1**. В таком случае считывание информации состоит в определении, заряжены или нет запоминающие емкости.

Запоминающая емкость может неопределенно долго сохранять состояние ^ **0** (разряд отсутствует), но только ограниченное время из-за утечки заряда - состояние **1**. Поэтому в рассматриваемых ЗУ необходимо периодически (примерно через каждые 2 мс) производить восстановление хранимой информации. Операция динамического восстановления информации называется **рефреш.** Рассматриваемые ЗУ получили название динамических.

Схема динамического ЗУ на МОП-транзисторах в памяти со структурой 2D-M представлены

на рис. 4.10. Запоминающей емкостью служит паразитная емкость **С** затвора транзистора T_2. Линия разрядно-адресного коммутатора **Y** используется для ввода в ЗЭ бита информации при записи и съема его при считывании (см. рис. 4.8). Так как ЗЭ использует источник питания только при считывании, то им может служить паразитная емкость $\mathbf{C_Y}$ линии **Y.**

Предварительно перед считыванием от разрядно-адресного коммутатора подается сигнал **R**, с помощью которого подготавливается считывание с мультиплексированием для ЗЭ, выбираемых линией разрядно-адресного

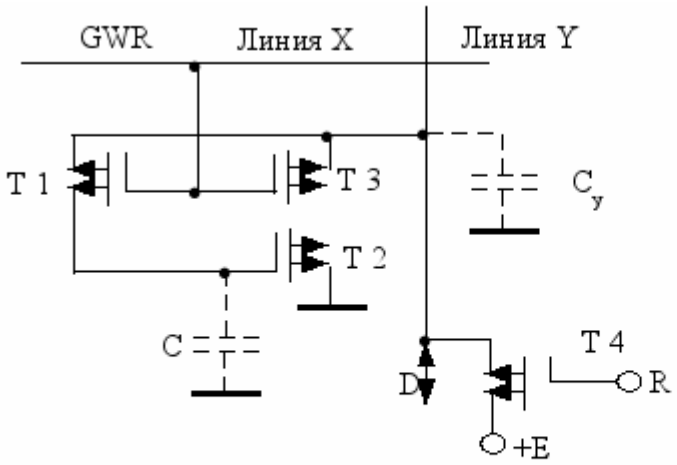


Рис 4.10. Принципиальная электрическая схема динамического МОП - ЗУ

формирователя. Сигнал ^ **R** открывает транзистор T_4, и емкость $\mathbf{C_Y}$подзаряжается от источника. Затем на линию **X** подается от адресного формирователя сигнал считывания - промежуточный уровень сигнала**CWR**, который открывает транзистор Тз, но не может открыть T_2. Если ЗЭ хранит **1**, то конденсатор **С** заряжен и открыт транзистор T_2. В этом случае через открытые транзисторы Тз и T_2 конденсатор $\mathbf{C_Y}$разряжается и низкий уровень (уровень **0**) сигнала **D** на линии **Y**указывает, что ЗЭ хранил инверсное значение, т.е. **1**. Если ЗЭ хранит**0**, то емкость **С** разряжена, T_2 закрыт и сигнал **CWR** не может вызвать разряд емкости $\mathbf{C_Y}$. Высокий уровень сигнала **D** (уровень **1**) указывает, что ЗЭ хранил **0**. Далее сигнал **D** через разрядно-адресный. коммутатор поступает на выход ЗУ.

При записи на линию **Y** поступает сигнал **D**, соответствующий записываемому двоичному знаку. Затем на линию **X** подается высокий уровень сигнала **CWR**, открывающий транзистор T_1, который подключает к линии **Y** конденсатор **С**. В результате независимо от своего предыдущего состояния емкость оказывается заряженной, если записывается **1**, и разряженной, если записывается **0.**

В ЗУ периодически производится регенерация информации. При регенерации в ЗЭ записывается инверсное значение хранимого до считывания кода. После каждой четной регенерации - его инверсия. В ЗУ имеется схема, сигнал которой указывает, какой код хранить в данный момент ЗЭ - прямой или инверсный.

Динамическая оперативная память **DRAM** — тип памяти, используемsй в большинстве систем оперативной памяти современных персональных компьютеров. Основное

преимущество динамической оперативной памяти состоит в том, что ее ячейки упакованы очень плотно, т.е. в очень малый кристалл (микросхему) можно упаковывать много битов, а значит, на их основе можно построить память большой емкости.

Регенерация памяти, к сожалению, "отнимает время" у процессора — каждый цикл регенерации по длительности занимает несколько циклов центрального процессора. В старых компьютерах циклы регенерации могли занимать до 10 % (или больше) процессорного времени, но в современных системами, работающих на частотах, равных сотням и тысячам мегагерц, расходы на регенерацию составляют 1 % (или меньше) процессорного времени.

К сожалению, динамическая оперативная память не отличается высоким быстродействием, обычно она намного медленнее, чем процессор. По этой причине имеется много различных типов организации динамической оперативной памяти, позволяющих улучшить эту характеристику.

3. Постоянные ЗУ (ПЗУ, ППЗУ)

Постоянные ЗУ в рабочем режиме ЭВМ допускают только считывание хранимой информации. В зависимости от типа ПЗУ занесение в него информации производится или в процессе изготовления, или в эксплуатационных условиях путем настройки, предваряющей использование ПЗУ в вычислительном процессе. В последнем случае ПЗУ называются постоянными запоминающими устройствами с изменяемым в процессе эксплуатации содержимым или программируемыми постоянными запоминающими устройствами (ППЗУ).

Постоянные ЗУ обычно строятся как адресные. Функционирование ПЗУ можно рассматривать как выполнение однозначного преобразования k-разрядного кода адреса ячейки запоминающего массива ЗМ в n-разрядный код хранящегося в ней слова.

По сравнению с ЗУ с произвольным обращением, допускающим как считывание, так и запись информации, конструкции ПЗУ значительно проще, их быстродействие и надежность выше, а стоимость ниже. Это объясняется большей простотой ЗЭ, отсутствием цепей для записи вообще или, по крайней мере, для оперативной записи, реализацией неразрушающего считывания, исключающего процедуру регенерации информации.

Одним из важнейших применений ПЗУ является хранение микропрограмм в микропрограммных управляющих устройствах ЭВМ. Для этой цели необходимы ПЗУ значительно большего, чем в ОП, быстродействия и умеренной емкости (10 000 - 100 000 бит).

Постоянные ЗУ широко используются для хранения программ в специализированных ЭВМ, в том числе в микроЭВМ, предназначенных для решения определенного набора задач, для которых имеются отработанные алгоритмы и программы, например в бортовых ЭВМ самолетов, ракет, космических кораблей, в управляющих вычислительных комплексах, работающих в АСУ технологических процессов. Такое применение ПЗУ позволяет существенно снизить требования к емкости ОП, повысить надежность и уменьшить стоимость вычислительной установки.

Очень широко ПЗУ используются в универсальных ЭВМ всех классов для хранения стандартных процедур начальной инициализации вычислительной системы и внешних

устройств, например BIOS в PC фирмы IBM. Программное обеспечение контроллеров интеллектуальных внешних устройств ЭВМ обычно также хранится во встроенных ПЗУ.

1. 4 Лекция №4 (2 часа).

Тема: «Структура ОЗУ**»**

1.4.1 Вопросы лекции:

1. Быстродействие памяти
2. Достоверность хранения данных
3. Кэширование ОП

1.4.2 Краткое содержание вопросов:

1. Быстродействие памяти

При замене неисправного модуля или микросхемы памяти новый элемент должен быть того же типа. Обычно проблемы возникают при использовании микросхем или модулей, не удовлетворяющих определенным (не слишком многочисленным) требованиям, например к длительности циклов регенерации. Можно также столкнуться с несоответствием в разводках выводов, емкости, разрядности или конструкции. Если вы не знаете, какие модули памяти позволяет использовать материнская плата, обратитесь к документации.

При установке более быстродействующих модулей памяти производительность компьютера, как правило, не повышается, поскольку система обращается к ней с прежней частотой. В системах, использующих модули DIMM и RIMM, быстродействие и прочие временные характеристики считываются из специального ПЗУ SPD, установленного на модуле. После этого контроллер памяти конфигурируется с применением этих параметров. Производительность таких систем можно повышать, устанавливая более скоростные модули памяти, вплоть до предела, поддерживаемого набором микросхем системной логики.

Чтобы акцентировать внимание на проблемах синхронизации и надежности, Intel и JEDEC создали стандарты для высокоскоростных модулей памяти, определяющие их типы, удовлетворяющие определенным уровням быстродействия. Согласно этим стандартам и выполняется классификация модулей памяти по временным характеристикам.

Основными признаками недостаточного быстродействия памяти или ее несоответствия временным характеристикам системы являются ошибки памяти и четности, а также “зависание” и неустойчивая работа системы. В этом случае тест POST также может выдать ошибки. Если точно не известно, какие модули памяти допустимы для вашей системы, свяжитесь с производителем компьютера и постарайтесь приобрести модули памяти от хорошо зарекомендовавшего себя поставщика.

2. Достоверность хранения данных

В любой из многих миллионов ячеек памяти возможен случайный сбой или окончательный отказ, приводящий к ошибке. Вероятность ошибки, естественно, возрастает с увеличением объема памяти. Современные технологии позволяют выпускать высоконадежные микросхемы памяти, у которых при корректной эксплуатации вероятность ошибки довольно мала, но все-таки она не нулевая.

В первых моделях PC, когда микросхемы памяти имели существенно худшие характеристики надежности по сравнению с современными, обязательно применялся контроль четности. В этом случае каждый байт памяти сопровождается битом четности (parity bit), дополняющим количество единиц в байте до нечетного. Значение бита четности аппаратно генерируется при записи в память и проверяется при считывании. При обнаружении ошибки четности схемой контроля вырабатывается немаскируемое прерывание

Со временем качество применяемых микросхем памяти улучшилось, и в целях удешевления модулей памяти от контроля четности стали.

Вопрос о необходимости контроля четности не имеет однозначного ответа. Контроль четности не всесилен, он выявляет в пределах каждого байта ошибки только нечетной кратности (искажение одного, трех, пяти или семи битов): правда, вероятность одновременного отказа или сбоя двух битов у работающей памяти весьма мала.

В компьютерах особо ответственного применения используют память с обнаружением и коррекцией ошибок (Error Checking and Correcting, ECC).

ОП современных ЭВМ и ВС реализуется на микросхемах статических (Static RAM — SRAM) и динамических (Dynamic RAM — DRAM) ОЗУ. Микросхемы статических ОЗУ имеют меньшее время доступа и не требуют циклов регенерации, Микросхемы динамических ОЗУ характеризуются большей емкостью и меньшей стоимостью, но требуют схем регенерации и имеют значительно большее время доступа.

Статическая память, SRAM (англ. Static RAM) – энергозависимая память, обладает очень малым временем доступа, основана на использовании триггеров в качестве запоминающего элемента. Триггер может быть построен из 5-6 транзисторов. Статические ОЗУ применяются для построения микроконтроллерных схем из-за простоты построения принципиальной схемы и возможности работать на сколь угодно низких частотах. Кроме того, статические ОЗУ применяются для построения КЭШ-памяти в универсальных компьютерах из-за высокого быстродействия статического ОЗУ. Статические ОЗУ требуют для своего построения большой площади кристалла, поэтому их емкость относительно невелика, что является их недостатком по сравнению с динамической памятью. Структура статического ОЗУ представлена на рисунке 2.3.3, где ЭП – это элемент памяти, а УВВ - устройство ввода-вывода.

Один элемент памяти может хранить бит информации. Запоминающим устройством, на этом рисунке является D-триггер, который находится на пересечении i-ой строки и j-го столбца, которые являются выходами дешифраторов строк и столбцов соответственно. Сигнал WR является управляющим и указывает на вид операции (WR=1 – операция записи, WR = 0 – операция чтения). При записи в триггер записывается информация, которая поступает на вход D через шинный формирователь SW. При чтении информация появляется на выходе шинного формирователя, которая поступает с выхода триггера.

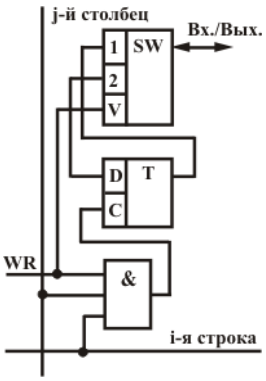


Рисунок 2.3.4. – Элемент памяти статического ОЗУ

На рисунке 2.3.5 показано условно-графическое изображение ОЗУ. Сигнал записи показан как WR, сигнал чтения – RD. Сигнал CS (chip select – выбор кристалла) активирует схему памяти для использования, т.е. в пассивном состоянии схема памяти находится в режиме хранения информации.

Временные диаграммы чтения и записи приведены на рисунке 2.3.6. На рисунке “а” изображена диаграмма обращения к ОЗУ для схем совместимая со стандартом от компании INTEL,

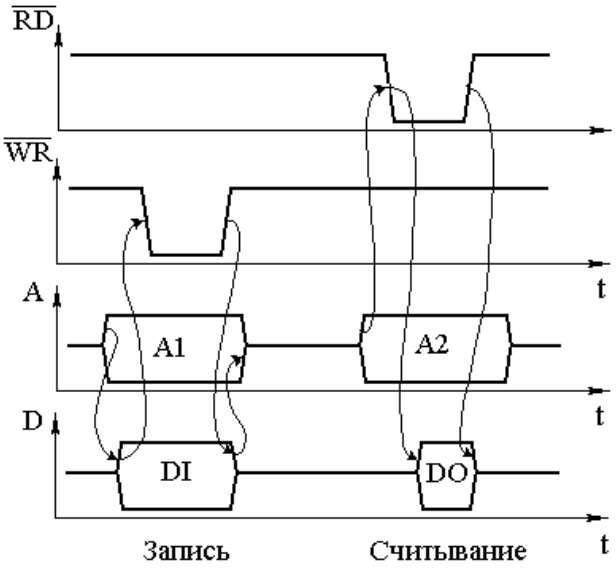


Рисунок 2.3.6 – Временные диаграммы обращения к статической памяти

Стрелочками показаны последовательности, в которых должны формироваться управляющие сигналы. DI – (data in)- входные (записываемые в память) данные, DO (data out) – выходные (читаемые из памяти) данные.

Динамическая память DRAM (англ. Dynamical RAM) в отличие от статической памяти, в динамической запоминающим элементом является конденсатор. Запись и считывание информации производится путем открывания транзисторов Т1 и Т2 и подключением накопительной емкости С к шине данных.

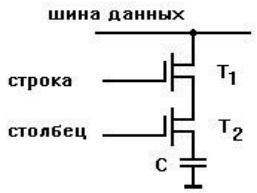


Рисунок 2.3.7 – Схема запоминающего элемента динамической памяти.

Обычно микросхемы ОЗУ организуются в виде матрицы ячеек, каждая из которых состоит из одного или более запоминающих элементов (ЗЭ) и имеет свои адрес. Каждый ЗЭ способен хранить один бит информации. Для ЗЭ любой полупроводниковой памяти характерны следующие свойства:

1. два стабильных состояния, представляющие двоичные 0 и 1;

2. в ЗЭ может быть произведена запись информации посредством перевода его в одно из двух возможных состояний;

3. для определения текущего состояния ЗЭ его содержимое может быть считано.

Для экономии числа контактов ячейки памяти организованы как элементы некой таблицы. При считывании или записи содержимое целой строки переносится в специальный буфер, а после считывания содержимое буфера перезаписывается в ту же строку ЗЭ динамической памяти, производится перезаряд конденсаторов,

Время хранения заряда конденсатором из-за паразитных утечек заряда ограничено. Чтобы не потерять имеющиеся данные, необходима периодическая перезапись информации, которая выполняется во время регенерации (refresh cycle). Это требование, кроме всего прочего, означает, что система основной памяти оказывается недоступной процессору, так как вынуждена «рассылать» сигналы регенерации каждой микросхеме. Операции разрядки-перезарядки занимают определенное время (в современных ЭВМ — от 1 до 5% от общего времени работы с памятью), снижая скорость работы динамической памяти, что является одним из основных ее недостатков. Однако учитывая информационную емкость, низкую стоимость и энергопотребление, этот тип памяти во многих случаях предпочтительнее статической.

При матричной организации ОЗУ (рис. 3.2) реализуется координатный принцип адресации ячеек. Адрес ячейки, поступающий по шине адреса ВМЭ пропускается через логику выбора, где он разделяется на две составляющие: адрес строки и адрес столбца. Адреса строки и столбца запоминаются соответственно в регистре адреса строки и регистре адреса столбца микросхемы. Регистры соединены каждый со своим дешифратором. Выходы дешифраторов образуют систему горизонтальных и вертикальных линий, к которым подсоединены запоминающие элементы матрицы, при этом каждый ЗЭ расположен на пересечении одной горизонтальной и одной вертикальной линии.

ЗЭ, объединенные общим «горизонтальным» проводом, принято называть строкой (row). Запоминающие элементы, подключенные к общему «вертикальному» проводу, называются столбцом (column). Фактически «вертикальных» проводов в микросхеме должно быть, по крайней мере, вдвое больше, чем этого требуется для адресации, поскольку к каждому ЗЭ необходимо подключить линию, по которой будет передаваться считанная и записываемая информация.

Совокупность запоминающих элементов и логических схем, связанных с выбором строк и столбцов, называют ядром микросхемы памяти. Помимо ядра, в микросхеме ОЗУ имеется еще интерфейсная логика, обеспечивающая взаимодействие ядра с внешним миром. В ее задачи, в частности, входит коммутация нужного столбца на выход при считывании и на вход — при записи.

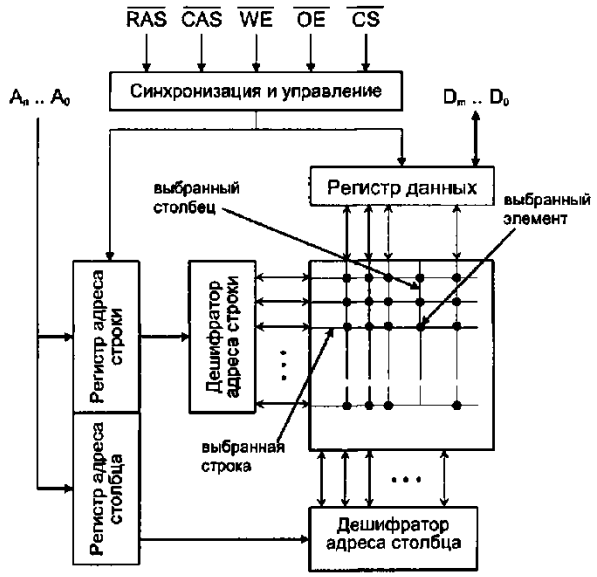


Рис. 3.2. **Матричная организация ОЗУ**

На физическую организацию ядра, как матрицы однобитовых ЗЭ, накладывается логическая организация памяти, под которой понимается разрядность микросхемы, т. е. количество линий ввода-вывода. Разрядность микросхемы определяет количество ЗЭ, имеющих один и тот же адрес (такая совокупность запоминающих элементов называется ячейкой), т. е. каждый столбец содержит столько разрядов, сколько есть линий ввода-вывода данных,

Для сокращения наполовину количества контактов корпуса микросхемы, необходимых для передачи адреса, адресные линии мультиплексируются, адреса строки и столбца в большинстве микросхем подаются в микросхему через одни и те же контакты последовательно во времени и запоминаются соответственно в регистрах адреса строки и столбца микросхемы. Обращение к микросхеме ОЗУ обычно происходит в два этапа. Первый этап начинается с выдачи сигнала RAS (Row-Access Strobe — «строб адреса строки»), который фиксирует в микросхеме поступивший адрес строки. Второй этап включает переключение адреса для указания адреса столбца и подачу сигнала CAS (Column-Access Strobe — «строб адреса столбца»), который фиксирует этот адрес и разрешает работу регистра данных микросхемы.

Сигнал выбора микросхемы CS (Crystal Select) активизирует микросхему и используется для ее выбора в системах, состоящих из нескольких микросхем, Вход WE (Write Enable — «разрешение записи») определяет вид выполняемой операции (считывание или запись).

Записываемая информация, поступающая по шине данных, первоначально заносится во входной регистр данных, а затем — в выбранную ячейку. При выполнении операции чтения информация из ячейки до ее выдачи на шину данных буферизуется в выходном регистре данных. Обычно роль входного и выходного выполняет один и тот же регистр. На все время, пока ОЗУ не использует шину данных, информационные выходы микросхемы

переводятся в третье (высокоимпедансное) состояние. Управление переключением в третье состояние обеспечивается сигналом OE (Output Enable — «разрешение выдачи выходных сигналов»). Этот сигнал активизируется при выполнении операции чтения.

Управление операциями с основной памятью осуществляется контроллером памяти. Обычно этот контроллер входит в состав центрального процессора либо реализуется в виде внешнего по отношению к памяти устройства. Хотя работа микросхем ОЗУ может быть организована как по синхронной, так и по асинхронной схеме, контроллер памяти — устройство синхронное, т. е. срабатывающее исключительно по тактовым импульсам. По этой причине операции с памятью принято описывать с привязкой к тактам. В общем случае на каждую такую операцию требуется, как минимум, пять тактов, которые используются следующим образом:

указание типа операции (чтение или запись) и установка адреса строки;

формирование сигнала RAS;

установка адреса столбца;

формирование сигнала CAS;

возврат сигналов RAS и CAS в неактивное состояние.

Следует учитывать также задержки, необходимые для стабилизации электрических процессов, порождаемых управляющими сигналами.

Первые ЗУ, которые впоследствии стали называть асинхронными динамическими ОЗУ, выполняли операции чтения и записи, получив лишь запускающий сигнал (обычно, сигнал строба адреса) независимо от каких-либо внешних синхронизирующих сигналов. Диаграмма циклов чтения и записи для таких ЗУ представлена на рис. 2.3.8, *а*) и 2.3.8, *б*) соответственно. Любой цикл (чтения или записи) начинается по спаду (фронту “1” →“0”) сигнала RAS#.

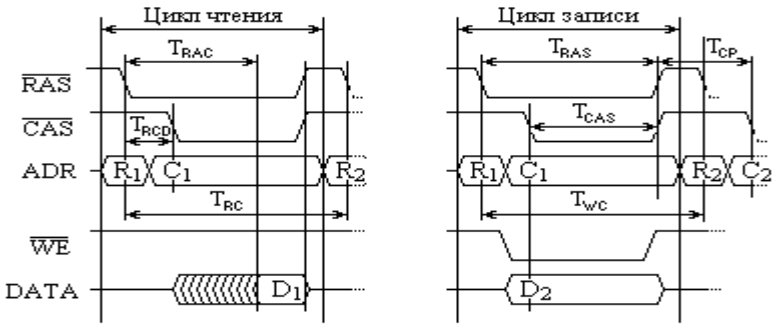


Рисунок 2.3.8. Временные диаграммы простых циклов чтения а) и записи б) асинхронной динамической памяти.

Как видно из диаграмм, адрес на шины адреса поступает двумя частями: адрес строки (обозначенный как R_1 или R_2) и адрес столбца (C_1 и C_2). В момент, когда на адресной шине

установилось требуемое значение части адреса, соответствующий сигнал строба (RAS# или CAS#) переводится в активное (нулевое) состояние.

Цикл записи начинается так же, как и цикл чтения, по спаду сигнала RAS# после подачи адреса строки. Записываемые данные выставляются на шину данных одновременно с подачей адреса столбца, а сигнал разрешения записи WE# при этом переводится в нулевое состояние (известен и несколько иной цикл “задержанной” записи). По истечении времени, достаточного для записи данных в элементы памяти, сигналы данных, WE#, RAS# и CAS# снимаются, что говорит об окончании цикла записи.

Помимо названного параметра T_{RAC} – времени доступа по отношению к сигналу RAS# (его значение для микросхем второй половины 90-х годов XX столетия составляло от 40 нс до 80 нс), - на временной диаграмме, представленной на рисунке 2.3.8, указаны еще несколько времен:

T_{RCD} – минимальное время задержки между подачей сигналов RAS# и CAS# (RAS-to-CAS Delay);

T_{RAS} и T_{CAS} – длительности (активного уровня) сигналов RAS# и CAS#;

T_{RC} и T_{WC} – длительности циклов чтения и записи соответственно;

T_{RP} и T_{CP} – времена регенерации строки и столбца соответственно (время регенерации определяет минимальную задержку, необходимую перед подачей очередного сигнала RAS# или CAS# после снятия (подъема в “1”) текущего).

Значения времен T_{RC} и T_{WC} для памяти (90-х годов) составляли порядка 50 – 100 нс, так что на одно (полное) обращение уходило от 5 до 7 циклов системной шины в зависимости от ее частоты, особенностей используемого чипсета и, собственно, быстродействия памяти. Так, для системной шины с частотой 66 Мгц длительность цикла составляет порядка 15 нс, что для 5 – 7 циклов дает диапазон 75 – 100нс, если частота системной шины составляла 100МГц, то 5 циклов занимают 50нс.

Поскольку адрес строки является старшей частью адреса, то для последовательных адресов памяти адрес строки одинаков (исключение составляет переход через границу строки). Это позволяет в (пакетном) цикле обращений по таким адресам задать адрес строки только для обращения по первому адресу, а для всех последующих задавать только адрес столбца. Такой способ получил название FPM (Fast Page Mode – быстрый страничный режим) и мог реализовываться обычными микросхемами памяти при поддержке контроллера памяти, обеспечивая сокращение времени обращения к памяти для всех циклов пакета, кроме первого.

Режим пакетной передачи (burst mode) предназначен для ускорения операций пересылки. Микросхемы пакетной памяти обычно имеют входной сигнал или бит в регистре режима, задающий порядок следования адресов (линейный или с чередованием) для конкретного применения. Длина пакетного цикла может быть фиксированной или программируемой и составлять 2, 4 или 8 передач.

Временная диаграмма пакетных циклов обращения к памяти (главным образом, чтения) является основной характеристикой производительности памяти компьютера. Ее

описывают числом тактов системной шины, требуемых для каждой передачи пакета. При этом, естественно, оговаривают и саму частоту.

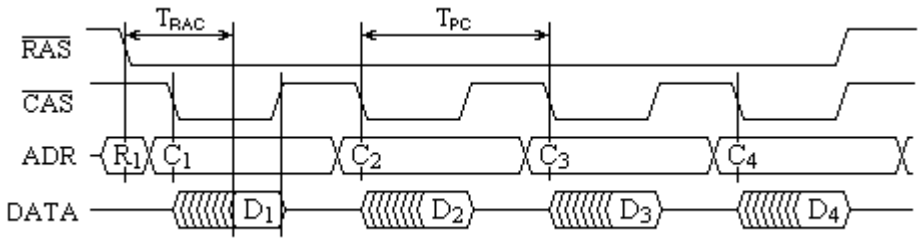


Рисунок 2.3.9. –Временная диаграмма цикла чтения последовательных адресов

динамической памяти DRAM в режиме FPM

Цикл чтения первого слова пакета выполняется так же, как и одиночное обращение. Второй и последующие циклы чтения оказываются короче первого из-за отсутствия фазы подачи адреса строки, и их длительность определяется минимально допустимым периодом следования импульсов **CAS#** – TPC (Page CAS Time). Соотношение длительностей первого и последующих циклов при частоте системной шины может достигать 5:3, откуда и обозначение 5-3-3-3, используемое как характеристика памяти, которая указывает, что первый из циклов пакета занимает по времени 5 циклов системной шины, а последующие – по 3 цикла.

Длительность (низкого уровня) импульса CAS# определяется не только временем извлечения данных из памяти, но и временем удержания их на выходе микросхемы памяти. Последнее необходимо для фиксации прочитанных данных (контроллером памяти), так как данные присутствуют на выходе только до подъема сигнала CAS#.

Однако у отечественных поставщиков этот тип памяти не получил широкого распространения, так как на смену асинхронной памяти пришла синхронная – SDRAM (англ. Synchronous Dynamic RAM).

В отличие от других типов DRAM, использовавших асинхронный обмен данными, ответ на поступивший в устройство управляющий сигнал возвращается не сразу, а лишь при получении следующего тактового сигнала. Это позволяет контроллеру точно знать время готовности данных. В микросхемах SDRAM внешние управляющие сигналы фиксируются положительным фронтом сигнала синхронизации и используются для формирования команд обращения к памяти. Тактовые сигналы позволяют организовать работу SDRAM в виде конечного автомата, исполняющего входящие команды. При этом входящие команды могут поступать в виде непрерывного потока, не дожидаясь, пока будет завершено выполнение предыдущих инструкций (конвейерная обработка): сразу после команды записи может поступить следующая команда, не ожидая, когда данные окажутся записаны. Поступление команды чтения приведёт к тому, что на выходе данные появятся спустя некоторое количество тактов — это время называется задержкой (англ. SDRAM latency) и является одной из важных характеристик данного типа устройств. Первый стандарт SDRAM с появлением последующих стандартов стал именоваться SDR (Single Data Rate). Использование шины данных в SDRAM оказалось осложнено задержкой в 2 или 3 такта между подачей сигнала чтения и появлением данных на шине данных, тогда как во время записи никакой задержки быть не должно. Потребовалась разработка достаточно сложного

контроллера, который не позволял бы использовать шину данных для записи и для чтения в один и тот же момент времени.

DDR SDRAM (англ. Double Data Rate Synchronous Dynamic Random Access Memory — синхронная динамическая память с произвольным доступом и удвоенной скоростью передачи данных). При использовании DDR SDRAM достигается удвоенная скорость работы, нежели в SDRAM, за счёт считывания команд и данных не только по фронту, как в SDRAM, но и по спаду тактового сигнала. За счёт этого удваивается скорость передачи данных без увеличения частоты тактового сигнала шины памяти. Таким образом, при работе DDR на частоте 100 МГц мы получим эффективную частоту 200 МГц (при сравнении с аналогом SDR SDRAM). В спецификации JEDEC есть замечание, что использовать термин «МГц» в DDR некорректно, правильно указывать скорость «миллионов передач в секунду через один вывод данных».

В DDR-памяти каждый буфер ввода-вывода на каждой линии шины данных передает два бита за один такт, то есть фактически работает на удвоенной тактовой частоте, оставаясь при этом синхронизованным с ядром памяти. Такой режим работы возможен, если два бита доступны буферу ввода-вывода на каждом такте работы памяти. Для этого требуется, чтобы каждая команда чтения приводила к передаче из ядра памяти в буфер ввода-вывода сразу 2n бит. С этой целью разрядность внутренней шины данных модуля от ядра памяти к буферам ввода-вывода увеличена вдвое. А из буфера они забираются с той же частотой, но только по противоположным фронтам тактового сигнала. Такая схема доступа называется 2n-Prefetch. В ней используются две независимые линии передачи, откуда биты поступают на шину данных.

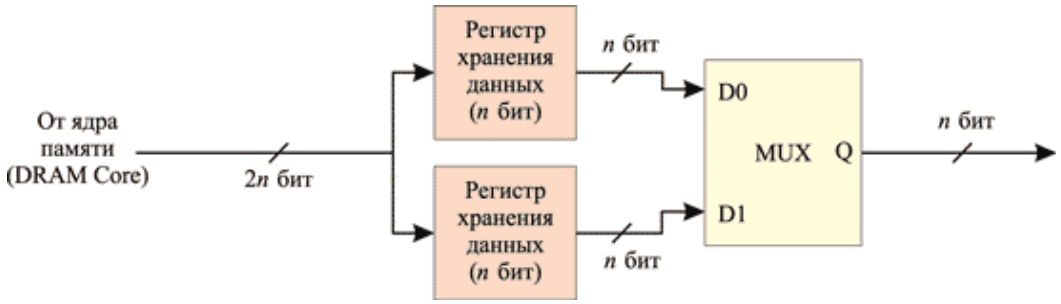


Рисунок 2.7 – Реализация технологии 2n-Prefetch

В остальном свойства памяти DDR не имеют значительных отличий от памяти типа SDR.

Все последующие технологии памяти организованы по рассмотренным выше принципам работы памяти типов DDR и SDR. Основная тенденция развития памяти основана не на увеличении частоты ядра памяти, а на увеличении ширины шины данных и снижении нагрузки на ядро. Поэтому фактически модули памяти, работающие на высоких эффективных частотах, используют внутри микросхемы с рабочей частотой около 100МГц.

Отличие DDR2 от DDR - вдвое большая частота работы шины, по которой данные передаются в буфер микросхемы памяти.

Отличие DDR3 от DDR2 уменьшено на 40% потребление энергии, что обусловлено пониженным (1,5 В, по сравнению с 1,8 В для DDR2 и 2,5 В для DDR) напряжением питания для ячеек памяти. Снижение напряжения питания достигается за счёт использования 90-нм

(вначале, в дальнейшем 65-, 50-, 40-нм) техпроцесса при производстве микросхем и применения транзисторов с двойным затвором Dual-gate (что способствует снижению токов утечки).

DDR4 Будет поддерживать частоты от 2133 до 4266 МГц, при этом напряжение питания ячеек памяти снижено до 1.2 В.

3. Кэширование ОП

Основная память, реализуемая на относительно медленных по своей природе микросхемах динамической памяти, обычно требует ввода тактов ожидания процессора (wait states) в циклы обращения к памяти. Статическая память, построенная, как и процессор, на триггерных ячейках, по своей природе способна догонять современные процессоры по быстродействию и избежать (или хотя бы сократить количество) тактов ожидания. Реализация основной памяти на микросхемах SRAM технически и экономически не оправдана, поскольку плотность упаковки информации у них существенно ниже, а удельная стоимость хранения и энергопотребление (или, что важнее, тепловыделение) существенно выше, чем у DRAM. Разумным компромиссом для построения экономичных и производительных систем явился иерархический способ построения оперативной памяти. Идея этого способа заключается в сочетании основной памяти большого объема на DRAM с относительно небольшой кэш памятью на быстродействующих микросхемах SRAM.

В переводе слово «кэш» (cache) означает склад или тайник. Тайна этого склада заключается в его «прозрачности» – для программы он не представляет собой дополнительной адресуемой области памяти. **«Кэш» (cache)** - дополнительное и быстродействующее хранилище копий блоков информации основной памяти, к которым, вероятно, в ближайшее время будет обращение. Кэш не может хранить копию всей основной памяти, поскольку его объем во много раз меньше объема основной памяти. Он хранит лишь ограниченное количество блоков данных и каталог (cache directory) – список их текущего соответствия областям основной памяти. Кроме того, кэшироваться может не вся память, доступная процессору: обычно кэшируется только основная динамическая память системной платы (память, установленная на адаптерах, не кэшируется), и из этой памяти кэшируется только часть.

При каждом обращении к кэшируемой памяти контроллер кэш-памяти по каталогу проверяет, есть ли действительная копия затребованных данных в кэше. Если она там есть, то это случай кэш-попадания (cache hit), и обращение за данными происходит только к кэш-памяти. Если действительной копии там нет, то это случай кэш-промаха (cache miss), и данные берутся из основной памяти. В соответствии с алгоритмом кэширования блок данных, считанный из основной памяти при определенных условиях, заместит один из блоков кэша. От качества алгоритма зависит процент попаданий и, следовательно, эффективность кэширования. Поиск блока в списке должен производиться достаточно быстро, чтобы не свести на нет выигрыш от применения быстродействующей памяти. Обращение к основной памяти может начинаться одновременно с поиском в каталоге, а в случае попадания – прерываться (архитектура Look Aside). Это экономит время, но лишние обращения к основной памяти ведут к излишнему энергопотреблению. Другой вариант – обращение к внешней памяти начинается только после фиксации случая промаха (архитектура Look Through), но на этом теряется, по крайней мере, один такт процессора, зато экономится энергия. В современных компьютерах кэш обычно строится по двухуровневой схеме. Первичный кэш (L1 Cache) встроен во все процессоры. Объем его невелик (8-32 Кбайт), и для повышения производительности для данных и команд часто

используется раздельный кэш (так называемая Гарвардская архитектура — противоположность Принстонской, использующей общую память для команд и данных). Быстродействие его таково, что он работает на внутренней тактовой частоте процессора (CPU Clock). Вторичный кэш (L2 Cache) обычно устанавливается на системной плате. Новые чипсеты поддерживают до 2 Мбайт L2 Cache. Его быстродействие обеспечивает работу на внешней тактовой частоте процессора – частоте системной шины (Host Bus Clock).

Кэш-контроллер должен обеспечивать **когерентность (coherency)** — согласованность данных кэш-памяти обоих уровней с данными в основной памяти, причем обращение к этим данным может производиться не только со стороны процессора (а процессоров может быть и несколько, и у каждого может быть свой внутренний кэш), но и со стороны других активных (bus-master) адаптеров, подключенных к шинам (PCI, VLB, ISA...).

Контроллер кэша оперирует строками (cache line) фиксированной длины. Строка может хранить копию блока основной памяти, размер которого, естественно, совпадает с длиной строки. С каждой строкой кэша связана информация об адресе скопированного в нее блока основной памяти и признаки ее состояния. Строка может быть действительной (valid) или недействительной (пустой). **Действительная строка** - строка, которая в текущий момент времени достоверно отражает соответствующий блок основной памяти. Информация о том, какой именно блок занимает данную строку (то есть старшая часть адреса или номер страницы), и ее состояние называется **тегом** (tag) и хранится в связанной с данной строкой ячейке специальной *памяти тегов* (tag RAM). В операциях обмена с основной памятью обычно строка участвует целиком (**несекторированный кэш**), для процессоров Pentium — 4´8=32 байт. Возможен и вариант **секторированного (sectored) кэша**, при котором одна строка содержит несколько смежных ячеек – секторов, размер которых соответствует минимальной порции обмена данных кэша с основной памятью. При этом в записи каталога, соответствующей каждой строке, должны храниться *биты действительности* для каждого сектора данной строки. Секторирование позволяет экономить память, необходимую для хранения каталога при увеличении объема кэша, поскольку большее количество бит каталога отводится под тег и выгоднее использовать дополнительные биты действительности, чем увеличивать глубину индекса (количество элементов) каталога.

Строки кэша под отображение блока памяти обычно выделяются только при операциях чтения. Запись блока, не имеющего копии в кэше, производится только в основную память (для повышения быстродействия она может производиться через буфер отложенной записи, но это отдельный механизм, не имеющий непосредственного отношения к рассматриваемому кэшированию). Поведение кэш-контроллера при операции записи в память, когда копия затребованной области находится в некоторой строке кэша, определяется его политикой записи (Write Policy). Существуют два основных алгоритма записи данных из кэша в основную память: сквозная запись WT (Write Through) и обратная запись WB (Write Back).

Алгоритм сквозной записи WT – алгоритм, предусматривающий выполнение каждой операции записи (даже однобайтной), попадающей в кэшированный блок, одновременно и в строку кэша, и в основную память, при этом процессору при каждой операции записи придется ожидать окончания относительно длительной записи в основную память. Достоинства алгоритма: 1) достаточно прост в реализации; 2) легко обеспечивает целостность данных за счет постоянного совпадения копий данных в кэше и основной памяти; 3) для него нет необходимости хранения признаков присутствия и модифицированности – вполне достаточно только информации тега (при этом считается, что

любая строка всегда отражает какой-либо блок, а какой именно – указывает тег). Недостаток – низкая эффективность записи. Существуют варианты этого алгоритма с применением отложенной буферированной записи, при которой данные в основную память переписываются через FIFO-буфер во время свободных тактов шины.

Алгоритм обратной записи WB – алгоритм, позволяющий уменьшить количество операций записи на шине основной памяти, который заключается в том, что если блок памяти, в который должна производиться запись, отображен и в кэше, то физическая запись сначала будет произведена в эту действительную строку кэша, и она будет отмечена как грязная (dirty), или модифицированная, то есть требующая выгрузки в основную память, и только после этой выгрузки (записи в основную память) строка станет чистой (clean), и ее можно будет использовать для кэширования других блоков без потери целостности данных, а в основную память данные переписываются только целой строкой (после заполнения всех ее секторов в случае секторированного кэша) или непосредственно перед ее замещением в кэше новыми данными. Данный алгоритм сложнее в реализации, но существенно эффективнее, чем WT. Поддержка системной платой кэширования с обратной записью требует обработки дополнительных интерфейсных сигналов для обеспечения выгрузки модифицированных строк в основную память, если к этой области производится обращение со стороны таких контроллеров шины, как графические адаптеры, контроллеры дисков, сетевые адаптеры и т.п.

В зависимости от способа определения взаимного соответствия строки кэша и области основной памяти различают три архитектуры кэш-памяти: кэш прямого отображения (direct-mapped cache), полностью ассоциативный кэш (fully associative cache) и их комбинация – частично- или наборно-ассоциативный кэш (set-associative cache).

1. 5 Лекция №5 (2 часа).

Тема: «Структура основной памяти»

1.5.1 Вопросы лекции:

1. Размещение информации в основной памяти компьютера
2. Внешняя память ЭВМ
3. Флэш-память

1.5.2 Краткое содержание вопросов:

1. Размещение информации в основной памяти компьютера

Адресуемой единицей информации основной памяти IBM PC является байт. Это означает, что каждый байт, записанный в ОП, имеет уникальный номер (адрес). При использовании 20-битной шины адреса абсолютный (физический) адрес каждого байта является пятиразрядным шестнадцатеричным числом, принимающим значения от 00000 до FFFFF. В младших адресах располагаются блоки операционной системы (векторы прерываний, зарезервированная область памяти BIOS), в этой же части могут размещаться драйверы устройств, дополнительные обработчики прерываний DOS и BIOS, командный процессор операционной системы. Затем располагается область памяти, отведенная пользователю. Область памяти пользователя заканчивается адресом 9FFFF. Этот адрес является физической границей оперативного ЗУ, последним адресом 640-Кбайтной основной памяти. Остальное адресное пространство (128 Кбайт с адреса AOOOO по

BFFFF) отведено под видеопамять, которая физически размещается не в ОП, а в адаптере дисплея. После видеопамяти расположено адресное пространство (256Кбайт) постоянного запоминающего устройства (ПЗУ), хранящего программы базовой системы ввода-вывода (BIOS - “Basic Input — Output System”). Эта часть ОП еще называется ROM-BIOS. Из отведенных 256 Кбайт непосредственно ПЗУ занимает 64 Кбайта, а остальные 192 Кбайт оставлены для расширения ПЗУ. Поскольку большая часть оставленной для расширения BIOS части адресного пространства не используется, в этих адресах часто располагается информация, необходимая для работы сетевых карт, графических расширителей и др.

Запись в ОП (и чтение из нее) может осуществляться не только байтами, но и машинными словами. При этом машинное слово при размещении в памяти занимает несколько смежных байтов. Каждый байт ОП имеет свой адрес. Но машинное слово характеризуется не всеми адресами занятых байтов, а только одним - адресом младшего байта слова. Обычно графически машинное слово изображается так, что младший байт находится

Рис.5.3. Стереотипное представление машинного слова

При записи слова младший байт размещается по адресу, который является адресом машинного слова, старший байт машинного слова размещается в следующем по порядку байте ОП, имеющем номер, увеличенный на 1 (здесь действует мнемоническое правило “младший байт - по младшему адресу”).

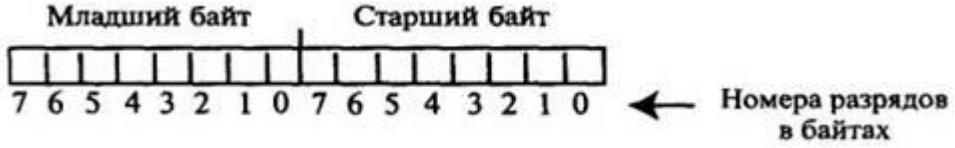


Рис. 5.4. “Вращение” байтов при чтении машинного слова из ОП

При чтении из ОП двух следующих подряд байтов машинного слова их принято размещать слева направо: сначала первый из прочитанных байтов (с меньшим адресом), а затем - следующий. В результате происходит “вращение” байтов (рис.5.4), которое психологически трудно воспринимается. Необходимо помнить, что при записи отдельных байтов каждый байт располагается в ОП по своему адресу, при чтении никакого вращения не происходит. При записи же в ОП единиц информации, имеющих в своем составе больше одного байта, адресом информационной единицы является адрес самого младшего байта, запись в ОП ведется побайтно, начиная с самого младшего байта, каждый последующий байт располагается в ячейке, адрес которой на 1 больше предыдущего. Иными словами, запись машинного или двойного слова производится справа налево, тогда как при чтении считанные байты обычно располагаются слева направо - происходят “вращение” байтов, перестановка их местами, что необходимо учитывать при работе с ОП на физическом уровне.

2. Внешняя память ЭВМ

Внешняя память - это память, предназначенная для длительного хранения программ и данных. Целостность содержимого ВЗУ не зависит от того, включен или выключен компьютер

Дисковод (накопитель) - устройство записи/считывания информации. Накопители имеют собственное имя – буква латинского алфавита, за которой следует двоеточие. Для подключения к компьютеру одного или несколько дисководов и управления их работой нужен Дисковый контроллер

Носитель информации (носитель записи) – материальный объект, способный хранить информацию. Информация записывается на носитель посредством изменения физических, химических и механических свойств запоминающей среды

По типу доступа к информации внешнюю память делят на два класса:

Устройства прямого (произвольного) доступа – время обращения к информации не зависит от места её расположения на носителе;

Устройство последовательного доступа – такая зависимость существует

В состав внешней памяти входят: 1) накопители на жестких магнитных дисках (НЖМД); 2) накопители на гибких магнитных дисках (НГМД); 3) накопители на магнитооптических компакт дисках; 4) накопители на оптических дисках (CD-ROM); 5) накопители на магнитной ленте и др.

НГМД - накопители на гибких магнитных дисках

- Предназначены для хранения небольших объемов информации

- Следует оберегать от сильных магнитных полей и нагревания

- Это носители произвольного (прямого) доступа к информации

- Используются для переноса данных с одного компьютера на другой

- Для работы с информации носитель должен быть отформатирован, т.е. должна быть произведена магнитная разметка диска на дорожки и секторы

- Скорость обмена информации зависит от скорости вращения дисковода. Для обращения к диску, вставленному в дисковод, присваивается имя **A:**

- Объём ГМД сравнительно небольшой (3,5 дюйма - 1,44 Мбайт)

- Рекомендуется делать копии содержимого ГМД

Диски называются гибкими потому, что их рабочая поверхность изготовлена из эластичного материала и помещена в твердый защитный конверт. Для доступа к магнитной поверхности диска в защитном конверте имеется закрытое шторкой окно. Поверхность диска покрыта специальным магнитным слоем (1- намагниченный участок, 0 – не намагниченный). Информация записывается с двух сторон диска на дорожки в виде концентрических

окружностей. Дорожки разбиваются на секторы. Современные дискетки имеют программную разметку. На каждом секторе выделяется участок для его идентификации, а на остальное место записываются данные. Дисковод снабжен двумя двигателями. Один обеспечивает вращение внутри защитного конверта. Второй перемещает головку записи/чтения вдоль радиуса поверхности диска. В защитном конверте имеется специальное окно защиты записи. С помощью бегунка это окно открывают и дискета становится доступна только на чтение, а на запись доступа не будет. Это предохраняет информацию на диске от изменения и удаления.

НЖМД - накопители на жестких магнитных дисках

- Предназначены для хранения той информации, которая наиболее часто используется в работе - программ операционной системы, компиляторов, сервисных программ, прикладных программ пользователя, текстовых документов, файлов базы данных

- Следует оберегать от ударов при установке и резких перемещений в пространстве

- Это носители с произвольным доступом к информации

- Для хранения информации разбивается на дорожки и секторы

- Скорость обмена информации значительно выше ГД

- Объём ЖД измеряется от Мбайт до сотен Гбайт

НЖМД встроены в дисковод и являются несъемными. Они представляют собой несколько алюминиевых дисков с магнитным покрытием, заключенных в единый корпус с электродвигателем, магнитными головками и устройством позиционирования. К магнитной поверхности диска подводится записывающая головка, которая перемещается по радиусу диска с внешней стороны к центру. Во время работы дисковода диск вращается. В каждом фиксированном положении головка взаимодействует с круговой дорожкой. На эти концентрические дорожки и производится запись двоичной информации. Благодаря хорошей защищенности от пыли, влаги и других внешних воздействий достигают высокой плотности записи, в отличии от дискет.

Для обращения к НЖМД используется имя, задаваемое прописной латинской буквой, начиная с **C:** , но с помощью специальной системной программы можно разбить свой физический ЖД на несколько логических дисков, каждому из которых дается соответствующее имя.

Накопители на жестких магнитных дисках часто называют винчестер - по первой модели ЖД, имевшего 30 дорожек по 30 секторов, что совпало с калибром 30?/30? охотничьего ружья

Оптические (лазерные) CD и DVD диски

- Предназначены для хранения любого вида информации

- Информацию на CD записывается с помощью лазерного луча

- Следует оберегать от царапин и загрязнения поверхности

- Это носители прямого (произвольного) доступа к информации

- Объем (ёмкость) CD составляет сотни Мбайт; DVD -более 1Гбайта

- Более долговечны и надежны, чем магнитные диски

CD – Compact Disk. Изготовляют из органических материалов с напылением на поверхность тонкого алюминиевого слоя. Лазерный диск имеет одну дорожку в виде спирали. Информация записывается отдельными секторами мощным лазерным лучом, выжигающим на поверхности диска углубления, и представляет собой чередование впадин и выпуклостей. При считывании информации выступы отражают свет слабого лазерного луча и воспринимаются как «1», впадины поглощают луч и, воспринимаются как «0». Это бесконтактный способ считывания информации. Срок хранения 50-100лет

DVD – Digital Video Disk. Имеет те же размеры, что и CD. Объем - Гбайт. Может быть односторонним или двухсторонним, а на каждой стороне может быть 1 или 2 рабочих слоя.

Накопители на магнитных лентах (НМЛ)

- Используют для резервного (относительно медленного) копирования и хранения больших объемов информации (архивы)

- Устройство для записи и считывания магнитных лент называется стример

- Это устройство последовательного доступа к информации

3. Флэш-память

Флэш-память - особый вид энергонезависимой, перезаписываемой полупроводниковой памяти.

Рассмотрим подробнее: энергонезависимая - не требующая дополнительной энергии для хранения данных (энергия требуется только для записи), перезаписываемая - допускающая изменение (перезапись) хранимых в ней данных и полупроводниковая (твердотельная) то есть не содержащая механически движущихся частей (как обычные жёсткие диски или CD), построенная на основе интегральных микросхем (IC-Chip).

Буквально у нас на глазах флэш-память превратилась из экзотического и дорогостоящего средства хранения данных в один из самых массовых носителей. Твёрдотельная память этого типа широко используется в портативных плеерах и карманных компьютерах, в фотоаппаратах и миниатюрных накопителях "флэш-драйвах". Первые серийные образцы работали с низкой скоростью, однако сегодня скорость считывания и записи данных на флэш-память позволяет смотреть хранящийся в миниатюрной микросхеме полноформатный фильм или запускать "тяжёлую" операционную систему класса Windows XP.

Благодаря низкому энергопотреблению, компактности, долговечности и относительно высокому быстродействию, флэш-память идеально подходит для использования в качестве накопителя в таких портативных устройствах, как: цифровые фото- и видео камеры, сотовые телефоны, портативные компьютеры, MP3-плееры, цифровые диктофоны, и т.п.

История

Первоначально твердотельный жесткий диск разрабатывался для высокоскоростных серверов и использовался в военных целях, но как это обычно бывает, со временем их стали применять и для гражданских компьютеров и серверов.

Возникло два класса устройств: в одном случае жертвовали цепями стирания, получая память высокой плотности, а в другом случае делали полнофункциональное устройство с гораздо меньшей емкостью.

Соответственно усилия инженеров были направлены на решение проблемы плотности компоновки цепей стирания. Они увенчались успехом изобретением инженера компании Toshiba Фудзио Масуокой в 1984 году. Фудзио представил свою разработку на Международном семинаре по электронным устройствам (International Electron Devices Meeting), в Сан-Франциско, в Калифорнии. Компанию Intel заинтересовало данное изобретение и через четыре года в 1988 году она выпустила первый коммерческий флеш-процессор NOR-типа.
NAND-архитектура флеш-памяти была анонсирована спустя год компанией Toshiba в 1989 году на Международной конференции построения твердотельных схем (International Solid-State Circuits Conference). У NAND-чипа была больше скорость записи и меньше площадь схемы.

Иногда утверждают, что название Flash применительно к типу памяти переводится как "вспышка". На самом деле это не совсем так. Одна из версий его появления говорит о том, что впервые в 1989-90 году компания Toshiba употребила слово Flash в контексте "быстрый, мгновенный" при описании своих новых микросхем. Вообще, изобретателем считается Intel, представившая в 1988 году флэш-память с архитектурой NOR.

Преимущества флеш-карт USB над остальными накопителями очевидны:

- малые габариты,
- очень легкий вес,
- бесшумность работы,
- возможность перезаписи,
- хорошая устойчивость к механическим воздействиям, в отличие от компакт-дисков и дискет(в 5-10 раз превышающие предельно допустимые для обычных жёстких дисков),
- выдерживает серьезные перепады температуры,
- отсутствие подвижных частей, что сводит потребление электроэнергии к минимуму,
- отсутствие проблем с подключением – USB выходы есть практически в любом компьютере,
- большой объем памяти,
- запись информации в ячейки памяти,
- срок хранения информации до 100 лет.
- Flash-память потребляет значительно (примерно в 10-20 и более раз) меньше энергии во время работы.

Также следует отметить, что для работы с USB флешкой не требуются какие-либо сторонние программы, адаптеры и прочее. Распознавание устройства происходит автоматически.

Если записывать на флэшку в день 10 раз, то ее хватит примерно на 30 лет.

Принцип действия

Принцип работы полупроводниковой технологии флеш-памяти основан на изменении и регистрации электрического заряда в изолированной области (кармане) полупроводниковой структуры.

Изменение заряда («запись» и «стирание») производится приложением между затвором и истоком большого потенциала, чтобы напряженность электрического поля в тонком диэлектрике между каналом транзистора и карманом оказалась достаточна для возникновения туннельного эффекта. Для усиления эффекта тунеллирования электронов в карман при записи применяется небольшое ускорение электронов путем пропускания тока через канал полевого транзистора.

Схематическое представление транзистора с плавающим затвором.

Между управляющим затвором и каналом, по которому ток течёт от истока к стоку, мы помещаем тот самый плавающий затвор, окружённый тонким слоем диэлектрика. В результате, при протекании тока через такой «модифицированный» полевой транзистор часть электронов с высокой энергией туннелируют сквозь диэлектрик и оказываются внутри плавающего затвора. Понятно, что пока электроны туннелировали, бродили внутри этого затвора, они потеряли часть энергии и назад практически вернуться не могут. SLC и MLC приборы

1. 6 Лекция №6 (2 часа).

Тема: «**Структура машинных команд и способы адресации**»

1.6.1 Вопросы лекции:

1. Основные определения
2. Возможные структуры машинных команд
3. Способы адресации

1.6.2 Краткое содержание вопросов:

1. Основные определения

Машинная команда представляет собой код, определяющий как операцию, исполняемую вычислительной системой, так и данные, участвующие в этой операции, где *операция* – преобразование данных, выполняемое вычислительной системой под управлением одной команды в течение одного или более тактов сигналов синхронизации.

Данные, представленные в дискретной форме, при произвольном их кодировании подвергаются двум основным видам воздействий: преобразованию и передаче. Преобразование дискретных данных выполняется при помощи арифметических, логических и специальных действий над частью слова, над отдельным словом или над совокупностью слов. Действие передачи дискретных данных осуществляется в пространстве и во времени. Передача данных в пространстве осуществляется как между элементами, узлами и блоками одной вычислительной системы, так и между различными вычисли-тельными системами, а

передача во времени – представляется как хранение данных в запоминающих устройствах (ЗУ) различного типа.

В качестве ЗУ для хранения программы, выполняющейся в текущий момент времени, используется оперативное запоминающее устройство (ОЗУ) статического типа (энергозависимое) с наименьшей адресуемой единицей информации – один байт.

Микропроцессор(микроконтроллер) при помощи системы структурированных машинных команд, объединённых алгоритмом в программу, выполняет все основные виды воздействий над дискретными данными, помещёнными в вычислительную систему.

1. характеристики машинных команд

Машинная команда (более широкое определение) представляет собой код, кратный единице адресуемого пространства ОЗУ, приведенный в двоичной системе счисления и задающий единственно возможную последовательность действий управляющего блока микропроцессора – последовательность микрокоманд. В целом машинная команда определяет вид операции, исполняемой в данном цикле микрокоманд. Машинная команда, представленная последовательностью микрокоманд, содержит характеристику двух операндов, задающих типы узлов операционного блока микропроцессора, которые участвуют при исполнении данной команды. Кроме того, код машинной команды содержит также в явной или в неявной форме следующие указания:

- где расположены операнды (для выполнения данной команды);
- куда поместить результат, при его наличии, после выполнения данной команды;
- каким будет адрес следующей исполняемой команды.

Микрокоманда – с позиции кодирования машинных команд – элементарное действие микропроцессора, выполняющееся в течение одного или нескольких тактов сигналов синхронизации, которое невозможно разделить на более простое. Микрокоманда позволяет выполнять:

- вычисление адреса расположения операнда или машинной команды;
- преобразование или перемещение данных;
- формирование или проверку условия ветвления или состояния операндов;
- фиксацию состояния признаков результатов вычислений (при необходимости).

Операнд–двоичное число, подвергающееся операционному преобразованию, которое выполняется микропроцессором при исполнении одного кода машинной команды. При исполнении машинной команды выполняется преобразование одного операнда независимо от количества операндов, указанных в коде данной команды. В качестве операндов используются:

- одно или два числа, над которыми выполняется преобразование;
- параметр внутренней характеристики микропроцессора;
- константа для перемещения;
- результат преобразования.

Существует два способа кодирования машинных команд, объединяющая особенность которых – разбиение каждой машинной команды на последовательность микрокоманд.

При первом способе кодирования каждая машинная команда содержит в своём коде последовательность всех микрокоманд, реализованных в данном микропроцессоре. Уровень логической 1 определяет необходимость в исполнении каждой микрокоманды. Такое кодирование машинных команд получило название*горизонтальное*. При количестве микрокоманд, превышающем сотню, каждый код машинной команды при горизонтальном кодировании становится громоздким и малоэффективным и в настоящее время практически не используется.

*При втором способе кодирования*каждой машинной команде присваивается порядковый номер, и именно порядковый номер определяет требующиеся микрокоманды при выполнении. Такое кодирование машинных команд получило название*вертикальное*. При вертикальном способе кодирования существует возможность создания машинных команд различной длины, явно не указывая при этом используемые микрокоманды. Вертикальное кодирование широко используется в настоящее время.

Алгоритм–последовательность операций, которые необходимо выполнить над исходными и промежуточными данными для получения искомого результата.

Представление алгоритма в виде машинных команд, воспринимаемых вычислительной системой, называется *программой*.

Вывод.Использование программ, представленных алгоритмами машинных команд вертикального кодирования, позволяет достигать высокого быстродействия при исполнении за счёт значительного сокращения длины кода команд по сравнению с числом микроопераций, которые производит микропроцессор при выполнении этой программы.

2. Возможные структуры машинных команд

Машинная команда представляет собой закодированное по определенным правилам указание микропроцессору на выполнение некоторой операции или действия. Каждая команда содержит элементы, определяющие:

1) что делать? (Ответ на этот вопрос дает элемент команды, называемый кодом операции (КОП).);

2) объекты, над которыми нужно что-то делать (эти элементы называются операндами);

3) как делать? (Эти элементы называются типами операндов – обычно задаются неявно).

Приведенный на рисунке 20 формат машинной команды является самым общим. Максимальная длина машинной команды – 15 байт. Реальная команда может содержать гораздо меньшее количество полей, вплоть до одного – только КОП.

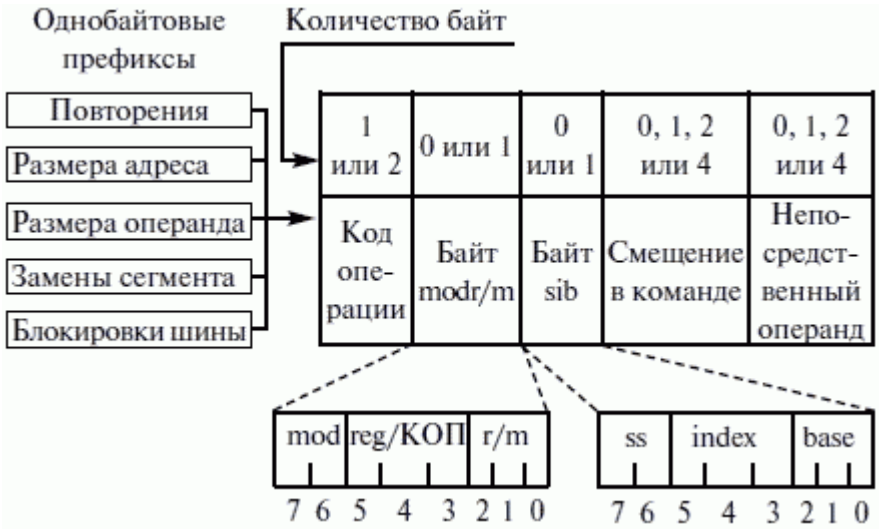


Рис. 20. Формат машинной команды

Опишем назначения полей машинной команды.

1. Префиксы.

Необязательные элементы машинной команды, каждый из которых состоит из 1 байта или может отсутствовать. В памяти префиксы предшествуют команде. Назначение префиксов – модифицировать операцию, выполняемую командой. Прикладная программа может использовать следующие типы префиксов:

1) префикс замены сегмента. В явной форме указывает, какой сегментный регистр используется в данной команде для адресации стека или данных. Префикс отменяет выбор сегментного регистра по умолчанию. Префиксы замены сегмента имеют следующие значения:

а) 2eh – замена сегмента cs;

б) 36h – замена сегмента ss;

в) 3eh – замена сегмента ds;

г) 26h – замена сегмента es;

д) 64h – замена сегмента fs;

е) 65h – замена сегмента gs;

2) префикс разрядности адреса уточняет разрядность адреса (32– или 16-разрядный). Каждой команде, в которой используется адресный операнд, ставится в соответствие разрядность адреса этого операнда. Этот адрес может иметь разрядность 16 или 32 бит. Если разрядность адреса для данной команды 16 бит, это означает, что команда содержит 16-разрядное смещение (рис. 20), оно соответствует 16-разрядному смещению адресного

операнда относительно начала некоторого сегмента. В контексте рисунка 21 это смещение называется эффективный адрес. Если разрядность адреса 32 бит, это означает, что команда содержит 32-разрядное смещение (рис. 20), оно соответствует 32-разрядному смещению адресного операнда относительно начала сегмента, и по его значению формируется 32-битное смещение в сегменте. С помощью префикса разрядности адреса можно изменить действующее по умолчанию значение разрядности адреса. Это изменение будет касаться только той команды, которой предшествует префикс;

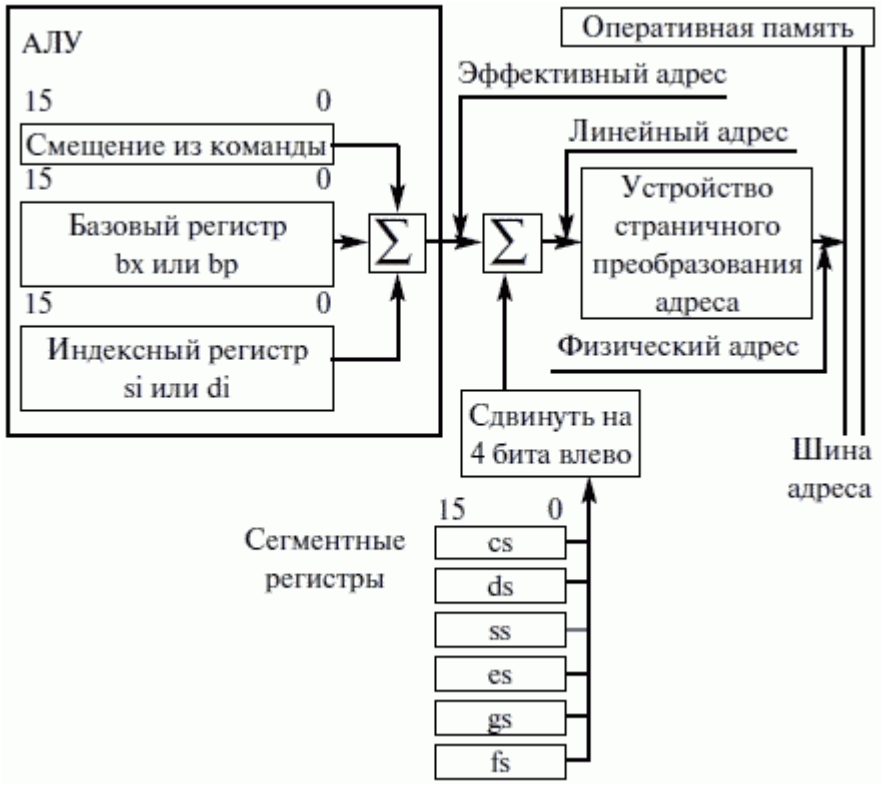


Рис. 21. Механизм формирования физического адреса в реальном режиме

3) префикс разрядности операнда аналогичен префиксу разрядности адреса, но указывает на разрядность операндов (32– или 16-разрядные), с которыми работает команда. В соответствии с какими правилами устанавливаются значения атрибутов разрядности адреса и операндов по умолчанию?

В реальном режиме и режиме виртуального 18086 значения этих атрибутов – 16 бит. В защищенном режиме значения атрибутов зависят от состояния бита D в дескрипторах исполняемых сегментов. Если D = 0, то значения атрибутов, действующие по умолчанию, равны 16 бит; если D = 1, то 32 бит.

Значения префиксов разрядности операнда 66h и разрядности адреса 67h. С помощью префикса разрядности адреса в реальном режиме можно использовать 32-разрядную адресацию, но при этом необходимо помнить об ограниченности размера сегмента величиной 64 Кбайт. Аналогично префиксу разрядности адреса вы можете использовать префикс разрядности операнда в реальном режиме для работы с 32-разрядными операндами (к примеру, в арифметических командах);

4) префикс повторения используется с цепочечными командами (командами обработки строк). Этот префикс «зацикливает» команду для обработки всех элементов цепочки. Система команд поддерживает два типа префиксов:

а) безусловные (rep – OOh), заставляющие повторяться цепочечную команду некоторое количество раз;

б) условные (repe/repz – OOh, repne/repnz – 0f2h), которые при зацикливании проверяют некоторые флаги, и в результате проверки возможен досрочный выход из цикла.

2. Код операции.

Обязательный элемент, описывающий операцию, выполняемую командой. Многим командам соответствует несколько кодов операций, каждый из которых определяет нюансы выполнения операции. Последующие поля машинной команды определяют местоположение операндов, участвующих в операции, и особенности их использования. Рассмотрение этих полей связано со способами задания операндов в машинной команде и потому будет выполнено позже.

3. Байт режима адресации modr/m.

Значения этого байта определяет используемую форму адреса операндов. Операнды могут находиться в памяти в одном или двух регистрах. Если операнд находится в памяти, то байт modr/m определяет компоненты (смещение, базовый и индексный регистры), используемые для вычисления его эффективного адреса (рисунок 21). В защищенном режиме для определения местоположения операнда в памяти может дополнительно использоваться байт sib (Scale-Index-Base – масштаб-индекс-база). Байт modr/m состоит из трех полей (рис. 20):

1) поле mod определяет количество байт, занимаемых в команде адресом операнда (рис. 20, поле смещение в команде). Поле mod используется совместно с полем r/m, которое указывает способ модификации адреса операнда «смещение в команде». К примеру, если mod = 00, это означает, что поле смещение в команде отсутствует, и адрес операнда определяется содержимым базового и (или) индексного регистра. Какие именно регистры будут использоваться для вычисления эффективного адреса, определяется значением этого байта. Если mod = 01, это означает, что поле смещение в команде присутствует, занимает 1 байт и модифицируется содержимым базового и (или) индексного регистра. Если mod = 10, это означает, что поле смещение в команде присутствует, занимает 2 или 4 байта (в зависимости от действующего по умолчанию или определяемого префиксом размера адреса) и модифицируется содержимым базового и (или) индексного регистра. Если mod =11, это означает, что операндов в памяти нет: они находятся в регистрах. Это же значение байта mod используется в случае, когда в команде применяется непосредственный операнд;

2) поле reg/коп определяет либо регистр, находящийся в команде на месте первого операнда, либо возможное расширение кода операции;

3) поле r/m используется совместно с полем mod и определяет либо регистр, находящийся в команде на месте первого операнда (если mod =11), либо используемые для вычисления эффективного адреса (совместно с полем смещение в команде) базовые и индексные регистры.

4. Байт масштаб – индекс – база (байт sib).

Используется для расширения возможностей адресации операндов. На наличие байта sib в машинной команде указывает сочетание одного из значений 01 или 10 поля mod и значения поля r/m = 100. Байт sib состоит из трех полей:

1) поля масштаба ss. В этом поле размещается масштабный множитель для индексного компонента index, занимающего следующие 3 бита байта sib. В поле ss может содержаться одно из следующих значений: 1, 2, 4, 8.

При вычислении эффективного адреса на это значение будет умножаться содержимое индексного регистра;

2) поля index. Используется для хранения номера индексного регистра, который применяется для вычисления эффективного адреса операнда;

3) поля base. Используется для хранения номера базового регистра, который также применяется для вычисления эффективного адреса операнда. В качестве базового и индексного регистров могут использоваться практически все регистры общего назначения.

5. Поле смещения в команде.

8-, 16– или 32-разрядное целое число со знаком, представляющее собой, полностью или частично (с учетом вышеприведенных рассуждений), значение эффективного адреса операнда.

6. Поле непосредственного операнда.

Необязательное поле, представляющее собой 8-, 16– или 32-разрядный непосредственный операнд. Наличие этого поля, конечно, отражается на значении байта modr/m.

3. Способы адресации

Перечислим и затем рассмотрим особенности основных видов адресации операндов в памяти:

1) прямую адресацию;

2) косвенную базовую (регистровую) адресацию;

3) косвенную базовую (регистровую) адресацию со смещением;

4) косвенную индексную адресацию со смещением;

5) косвенную базовую индексную адресацию;

6) косвенную базовую индексную адресацию со смещением.

Прямая адресация

Это простейший вид адресации операнда в памяти, так как эффективный адрес содержится в самой команде и для его формирования не используется никаких дополнительных источников или регистров. Эффективный адрес берется непосредственно из поля смещения машинной команды (см. рис. 20), которое может иметь размер 8, 16, 32 бит. Это значение однозначно определяет байт, слово или двойное слово, расположенные в сегменте данных.

Прямая адресация может быть двух типов.

Относительная прямая адресация

Используется для команд условных переходов, для указания относительного адреса перехода. Относительность такого перехода заключается в том, что в поле смещения машинной команды содержится 8-, 16– или 32-битное значение, которое в результате работы команды будет складываться с содержимым регистра указателя команд ip/eip. В результате такого сложения получается адрес, по которому и осуществляется переход.

Абсолютная прямая адресация

В этом случае эффективный адрес является частью машинной команды, но формируется этот адрес только из значения поля смещения в команде. Для формирования физического адреса операнда в памяти микропроцессор складывает это поле со сдвинутым на 4 бита значением сегментного регистра. В команде ассемблера можно использовать несколько форм такой адресации.

Но такая адресация применяется редко – обычно используемым ячейкам в программе присваиваются символические имена. В процессе трансляции ассемблер вычисляет и подставляет значения смещений этих имен в формируемую им машинную команду в поле «смещение в команде». В итоге получается так, что машинная команда прямо адресует свой операнд, имея, фактически, в одном из своих полей значение эффективного адреса.

Остальные виды адресации относятся к косвенным. Слово «косвенный» в названии этих видов адресации означает то, что в самой команде может находиться лишь часть эффективного адреса, а остальные его компоненты находятся в регистрах, на которые указывают своим содержимым байт modr/m и, возможно, байт sib.

Косвенная базовая (регистровая) адресация

При такой адресации эффективный адрес операнда может находиться в любом из регистров общего назначения, кроме sp/esp и bp/ebp (это специфические регистры для работы с сегментом стека). Синтаксически в команде этот режим адресации выражается заключением имени регистра в квадратные скобки []. К примеру, команда mov ax, [ecx] помещает в регистр ax содержимое слова по адресу из сегмента данных со смещением, хранящимся в регистре ecx. Так как содержимое регистра легко изменить в ходе работы программы, данный способ адресации позволяет динамически назначить адрес операнда для некоторой машинной команды. Это свойство очень полезно, например, для организации циклических вычислений и для работы с различными структурами данных типа таблиц или массивов.

Косвенная базовая (регистровая) адресация со смещением

Этот вид адресации является дополнением предыдущего и предназначен для доступа к данным с известным смещением относительно некоторого базового адреса. Этот вид адресации удобно использовать для доступа к элементам структур данных, когда смещение элементов известно заранее, на стадии разработки программы, а базовый (начальный) адрес структуры должен вычисляться динамически, на стадии выполнения программы. Модификация содержимого базового регистра позволяет обратиться к одноименным элементам различных экземпляров однотипных структур данных.

К примеру, команда mov ax,[edx+3h] пересылает в регистр ax слова из области памяти по адресу: содержимое edx + 3h.

Команда mov ax,mas[dx] пересылает в регистр ax слово по адресу: содержимое dx плюс значение идентификатора mas (не забывайте, что транслятор присваивает каждому идентификатору значение, равное смещению этого идентификатора относительно начала сегмента данных).

Косвенная индексная адресация со смещением

Этот вид адресации очень похож на косвенную базовую адресацию со смещением. Здесь также для формирования эффективного адреса используется один из регистров общего назначения. Но индексная адресация обладает одной интересной особенностью, которая очень удобна для работы с массивами. Она связана с возможностью так называемого масштабирования содержимого индексного регистра. Что это такое?

Посмотрите на рисунок 20. Нас интересует байт sib. При обсуждении структуры этого байта мы отмечали, что он состоит из трех полей. Одно из этих полей – поле масштаба ss, на значение которого умножается содержимое индексного регистра.

К примеру, в команде mov ax,mas[si*2] значение эффективного адреса второго операнда вычисляется выражением mas+(si)*2. В связи с тем, что в ассемблере нет средств для организации индексации массивов, то программисту своими силами приходится ее организовывать.

Наличие возможности масштабирования существенно помогает в решении этой проблемы, но при условии, что размер элементов массива составляет 1, 2, 4 или 8 байт.

Косвенная базовая индексная адресация

При этом виде адресации эффективный адрес формируется как сумма содержимого двух регистров общего назначения: базового и индексного. В качестве этих регистров могут применяться любые регистры общего назначения, при этом часто используется масштабирование содержимого индексного регистра.

Косвенная базовая индексная адресация со смещением

Этот вид адресации является дополнением косвенной индексной адресации. Эффективный адрес формируется как сумма трех составляющих: содержимого базового регистра, содержимого индексного регистра и значения поля смещения в команде.

К примеру, команда mov eax,[esi+5] [edx] пересылает в регистр eax двойное слово по адресу: (esi) + 5 + (edx).

Команда add ax,array[esi] [ebx] производит сложение содержимого регистра ax с содержимым слова по адресу: значение идентификатора array + (esi) + (ebx).

1. 7 Лекция №7 (2 часа).

Тема: «Принципы построения процессора**»**

1.7.1 Вопросы лекции:

1) Принципы построения простейшего процессора
2) Операционное устройство процессора
3) Управляющее устройство процессора

1.7.2 Краткое содержание вопросов:

1) Принципы построения простейшего процессора

Ранее, при рассмотрении обобщенной структуры ЭВМ, отмечалось, что основным устройством, непосредственно осуществляющим переработку поступающей в ЭВМ информации, является процессор (в больших ЭВМ – центральный процессор). Естественно, что конкретные типы ЭВМ содержат в своем составе процессоры, построенные по различным схемам, и процессоры больших ЭВМ существенно отличаются от процессоров мини- и микроЭВМ (о суперЭВМ и говорить не приходится). Однако основные принципы построения процессоров, в общем-то, одинаковые, причем наиболее наглядно их можно продемонстрировать на примере простейшего микропроцессора. Это оправдано и с той точки зрения, что инженер-разработчик радиоэлектронной аппаратуры или аппаратов автоматического управления имеет дело не с большими ЭВМ, а с микропроцессорными комплектами и построенными на их базе мини- и микроЭВМ. Ввиду этого рассмотрев общие вопросы построения ЭВМ, более подробно остановимся на обобщенной структуре гипотетического микропроцессора.

Ранее рассматривались действия над числами (сложение, вычитание, умножение), представленными в различной форме. Было подчеркнуто, что все эти действия осуществляются с помощью *элементарных* операций, выполняемых в определенной *последовательности*.

К таким элементарным операциям относятся:

- запись числа в регистр;

- инвертирование содержимого разрядов регистра;

- пересылка содержимого регистров;

- сдвиг содержимого регистра;

- сложение кодов;

- поразрядные логические операции или анализ разрядов;

- операция счета c+1 или c-1 (инкремент или декремент).

Пример.

Операция умножения реализуется с помощью:

- анализа разряда множителя;

- суммирования;

- сдвига.

Все эти действия выполняются в устройстве, называемом процессором, которое состоит из двух устройств – *операционного* (ОУ) и *управляющего* (УУ).

ОУ – выполняет указанные элементарные операции.

УУ – управляет ОУ, задавая необходимую последовательность выполнения этих операций.

Это соответствует принципу В.М. Глушкова, что в любом устройстве обработки цифровой информации можно выделить операционный и управляющий блоки.

В качестве узлов УУ и ОУ включают в себя регистры, счетчики, сумматоры, мультиплексоры, дешифраторы и т.д., т.е. устройства импульсной цифровой техники. Кроме того, нормальное функционирование процессора и всей ЭВМ возможно только при наличии высокостабильных импульсных последовательностей, формируемых, как правило, из одной импульсной последовательности, вырабатываемой кварцевым генератором. Эти *тактовые* импульсные последовательности синхронизируют работу узлов процессора, а иногда и всей ЭВМ.

Каждая элементарная операция, выполняемая в одном из узлов ОУ в течение одного тактового периода, называется *микрооперацией*.

В определенные тактовые периоды одновременно могут выполняться несколько микроопераций, например: $R_2 \neg 0$, Сч ¬ (Сч) – 1 и т.д. Такая совокупность непротиворечивых микроопераций называется *микрокомандой*, а набор микрокоманд, предназначенный для решения задачи, называется *микропрограммой*.

Если в ОУ предусмотрена возможность выполнения n различных микроопераций, то из УУ должно выходить n управляющих цепей $S_1,...,S_n$, каждая из которых соответствует своей микрооперации. В силу того что УУ определяет микропрограмму, т.е. какие и в какой временной последовательности должны выполняться микрооперации, оно получило название *микропрограммного автомата*. Соответственно ОУ часто называют*операционным автоматом*.

Формирование управляющих сигналов $S_1,...,S_n$ может зависеть как от внешних сигналов КОП (команды ассемблера), так и от состояния узлов ОУ, определяемого известительными сигналами признаков состояния $P_1,...,P_m$, поступающими с выхода ОУ на соответствующие входы УУ.

Как уже отмечалось, ОУ выполняет над исходными данными различные арифметические и логические операции, поэтому ОУ наиболее часто называют *арифметико-логическим устройством,* или АЛУ.

Деление любого процессора на программный и операционный автоматы достаточно очевидно и не вызывает особых трудностей в понимании. Однако структурные схемы даже простейших реальных процессоров, помимо АЛУ и УУ, содержат еще ряд узлов (регистров, счетчиков, дешифраторов), которые вроде бы не относятся ни к АЛУ, ни к УУ. Для устранения путаницы в дальнейшем материале необходимо сделать ряд замечаний:

1. В абсолютном большинстве случаев устройства обработки цифровой информации имеют многоуровневую структуру, т.е. построены по принципу "матрешки". Это означает, что УУ и ОУ могут сами распадаться на пары УУ' и ОУ', которые, в свою очередь, также могут распадаться на соответствующие УУ и ОУ. Все зависит от степени детализации рассмотрения данного цифрового устройства. Этот *принцип многоуровневости*справедлив для всех устройств ЭВМ.

Действительно, если рассматривать процессор в целом и делить его на УУ и ОУ, то совершенно безразлично, как выполняются арифметико-логические операции в ОУ – с помощью очень сложных логических схем или с помощью простой логики, работающей под управлением какого-либо вспомогательного УУ. Аналогичные рассуждения справедливы и для УУ.

Так, например, центральный процессор больших ЭВМ общего назначения середины 70-х годов разбивался на 4-5 уровней, на каждом из которых можно выделить свое УУ и ОУ. Современные процессоры имеют еще более сложную структуру.

Более того, эти рассуждения справедливы в целом для ЭВМ, которую можно разложить на ряд *виртуальных* (кажущихся) машин и с каждой работать на соответствующем уровне. В общем случае современные универсальные ЭВМ имеют шесть уровней:

- уровень проблемно-ориентированного языка;
- процедурно-ориентированный язык;
- ассемблерный уровень (язык ассемблера);
- уровень операционной системы (язык операционной системы);
- традиционный машинный уровень (язык машинных команд);
- микропрограммный уровень (язык микрокоманд).

Машинные языки двух нижних уровней являются цифровыми, и программы на них состоят из длинных числовых последовательностей, очень неудобных для человека, но понятных машине. Все более высокие уровни содержат слова и аббревиатуру, что более удобно для человека.

2. Из сказанного следует, что только самые простейшие процессоры имеют один уровень и могут быть в чистом виде разложены на УУ и ОУ, состоящие из комбинационных логических схем, способных выполнять элементарные арифметико-логические операции.

3. В настоящее время нет строгого определения АЛУ, что вызывает некоторую путаницу при пользовании различной литературой. АЛУ обычно обозначают так, как

показано на рис. 3.2. При этом одни авторы подразумевают под АЛУ только комбинационные логические схемы, способные выполнять операции двоичного суммирования (т.е. фактически двоичный сумматор), другие – целый комплекс схем для выполнения арифметико-логических операций, который сам может быть разложен на УУ и ОУ.

4. Из сказанного следует вывод, что в общем случае понятия микрооперации и микропрограммы *относительны* и требуют конкретизации уровня рассмотрения процессора, поскольку один такт верхнего уровня может включать в себя несколько тактов нижнего уровня.

5. Для устранения путаницы при изучении основных принципов построения элементарных процессоров будем считать:

- процессор имеет один уровень;

- процессор пользуется одной тактовой последовательностью;

- значок АЛУ обозначает комплекс комбинационных схем, способных выполнять двоичное суммирование, сдвиг двоичного числа, простейшие поразрядные логические операции;

- узлы микропроцессора, не относящиеся непосредственно к схеме управления, будем считать вспомогательными узлами АЛУ, или, точнее, узлами, обеспечивающими нормальное функционирование АЛУ.

2) Операционное устройство процессора

В целом ОУ выполняет операции, определяемые командами, и формирует эффективные адреса.

УУ вырабатывает управляющие сигналы, поступающие во все блоки вычислительной машины. В составе УУ можно выделить следующие функциональные блоки:

1) *регистр команд* – запоминающий регистр, в котором хранится код команды: код операции и адреса операндов (расположен в интерфейсной части процессора);

2) *дешифратор операций* – логический блок, который в соответствии с поступающим из регистра команд кодом операции выбирает один из множества имеющихся у него выходов;

3) *постоянное запоминающее устройство (ПЗУ) микропрограмм*хранит управляющие импульсы для выполнения в блоках вычислительной машины процедур обработки информации; импульс по выбранному дешифратором операций проводу считывает из ПЗУ микропрограмм необходимую последовательность управляющих сигналов;

4) *узел формирования адреса* (располагается в ШИ) – устройство для вычисления полного адреса ячейки памяти (регистра) по реквизитам, поступающим из микропроцессорной памяти или регистра команд;

5) *кодовые шины данных, адреса и инструкций* – часть внутренней интерфейсной шины процессора.

Таким образом, УУ формирует управляющие сигналы для выполнения процессором своих функций, рассмотренных в предыдущем вопросе.

АЛУ предназначено для выполнения арифметических и логических операций преобразования информации. Функционально в простейшем варианте АЛУ состоит из следующих компонент:

1) *сумматор* выполняет процедуру сложения двоичных кодов, имеет разрядность двойного машинного слова (32 бита);

2) *регистры* – быстродействующие ячейки памяти различной длины: регистр 1 имеет разрядность 32 бита, регистр 2 - 16 бит; при сложении в регистр 1 помещается первое слагаемое, а потом результат, в регистр 2 – второе слагаемое;

3) *схема управления* принимает по кодовым шинам инструкций управляющие сигналы от УУ и преобразует в сигналы для управления работой регистров и сумматора.

АЛУ выполняет арифметические операции только над двоичными числами с фиксированной точкой. Для обработки чисел с плавающей точкой привлекается математический сопроцессор или специально составленные программы.

Более подробные сведения об устройстве и функционировании УУ и АЛУ можно найти в /2, 4, 5/.

Регистры ОУ – часть микропроцессорной памяти. Рассмотрим регистры на примере базового процессора Intel 8086, который содержит всего 14 двухбайтовых регистра. В современных процессорах их гораздо больше и большей разрядности. Однако в качестве базовой модели, в частности для языка Ассемблера, используется 14-регистровая память процессора.

В состав ОУ входят следующие регистры:

1) *регистры общего назначения (РОН)* или *универсальные*: AX - (AH, AL), BX - (BH, BL), CX - (CH, CL), DX - (DH, DL) могут использоваться для временного хранения любых данных, при этом можно работать с каждым регистром целиком, а можно отдельно, с каждой его половиной; но каждый из РОН может использоваться и как специальный при выполнении некоторых конкретных команд;

2) *регистры смещений*: SP, BP, SI, DI являются неделимыми и предназначены для хранения относительных адресов ячеек памяти внутри сегментов (смещений относительно начала сегментов);

2.1) *SP (Stack Pointer)* – смещение вершины стека;

2.2) *BP (Base Pointer)* – смещение начального адреса поля памяти, непосредственно отведённого под стек;

2.3) *SI (Source Index), DI (Destination Index)* предназначены для хранения адресов индекса источника и приёмника данных при операциях над строками и им подобных.

Слово состояния процессора (PSW – Processor State Word) или регистр *флагов* – имеет размер 2 байта и содержит одноразрядные признаки или флаги. Всего в регистре 9 флагов: 6 из них *условные* или *статусные*, отражают результаты операций, выполненных ОУ, 3 других – *управляющие*, определяют режим исполнения программы.

1) Статусные флаги.

1.1) *CF (Carry Flag)* – флаг переноса. Устанавливается в 1, если при выполнении арифметических и некоторых операций сдвига возникает «перенос» из старшего разряда.

1.2) *PF (Parity Flag)* – флаг чётности. Проверяет младшие 8 битов результатов над данными. Чётное число единиц приводит к установке этого флага в 1, нечётное – в 0.

1.3) *AF (Auxiliary Carry Flag)* – флаг логического переноса в двоично-десятичной арифметике. Устанавливается в 1, если арифметическая операция приводит к переносу или займу четвёртого справа бита однобайтового операнда. Используется при арифметических операциях над двоично-десятичными кодами и кодами ASCII.

1.4) *ZF (Zero Flag)* – флаг нуля. Устанавливается в 1, если результат операции равен 0, в противном случае ZF обнуляется.

1.5) *SF (Sign Flag)* – флаг знака. Устанавливается в 1, если результат арифметической операции является отрицательным, в 0, если результат положительный.

1.6) *OF (Overflow Flag)* – флаг переполнения. Устанавливается в единицу при арифметическом переполнении, когда результат выходит за пределы разрядной сетки.

2) Управляющие флаги.

2.1) *TF (Trap Flag)* – флаг трассировки. Единичное состояние этого флага переводит процессор в режим пошагового выполнения программы.

2.2) *IF (Interrupt Flag)* – флаг прерываний. При нулевом состоянии этого флага прерывания запрещены, при единичном – разрешены (о механизме прерываний речь пойдёт в следующей лекции).

2.3) *DF (Direction Flag)* – флаг направления. Используется в строковых операциях для задания направления обработки данных; при единичном состояния строки обрабатываются «справа налево», при нулевом – «слева направо».

Расположение флагов в регистре PSW показано на рисунке 4.2. Свободные биты отведены для использования в будущем.

Рисунок 4.2 – Схема расположения флагов в регистре PSW

3) Управляющее устройство процессора

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.

Особенности программного и микропрограммного управления

В микропроцессорах используют два метода выработки совокупности функциональных управляющих сигналов: программный и микропрограммный.

Микропрограммная схема управления

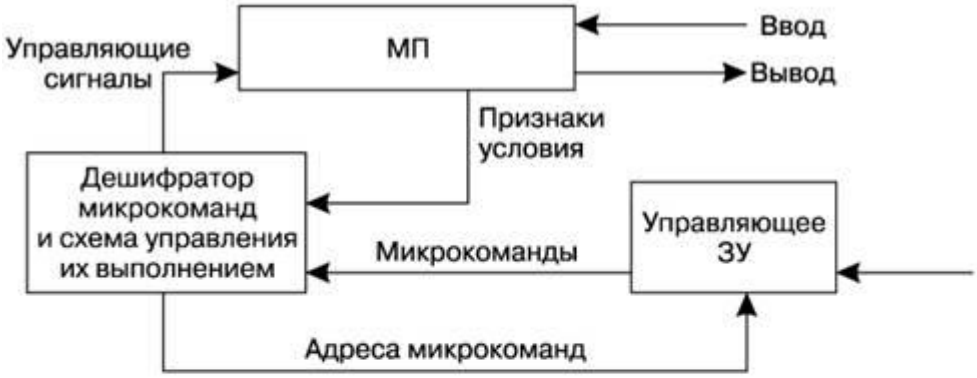


Схема управления с фиксированной системой команд

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:

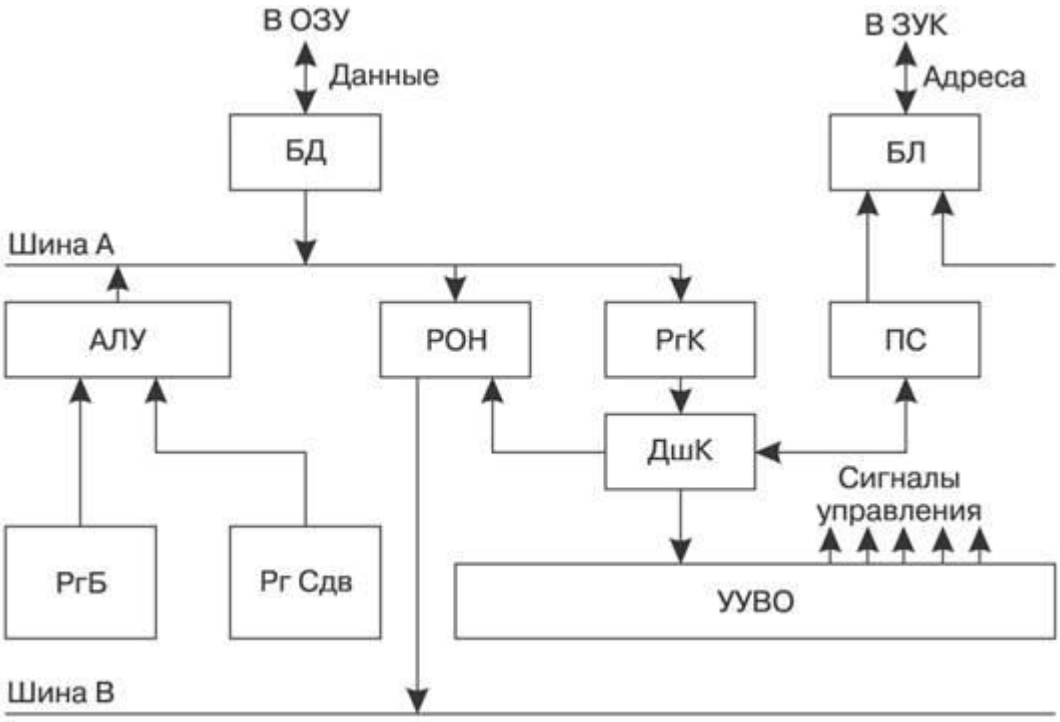
1) любая операция, реализуемая устройством, является последовательностью элементарных действий - микроопераций;

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.



Команды, составляющие программу, размещаются во внешнем запоминающем устройстве команд (ЗУК). МП обеспечивает выборку команд в нужной последовательности, их дешифрацию и т.д. Для выполнения этих функций МП должны иметь: программный счетчик (ПС), хранящий текущий адрес команды, регистр команд (РК), предназначенный для хранения команд, поступающих из ЗУК. Адрес команды из ПС поступает через блок усилителей буфера адреса (БА) на шину адреса, а затем на адресные входы ЗУК. Команда, выбранная из ЗУК, поступает на буфер данных (БД), а затем, на регистр команд (РгК). Код команды расшифровывается дешифратором команд (ДшК), который передает код выполняемой операции в устройство управления выполнением операций (УУВО), вырабатывающее последовательность управляющих сигналов (рис. 1.3). В настоящее время существуют МП с одной, двумя (шины А и Б на рис. 1.3) и тремя внутренними шинами. При трехшинной организации МП возможно выполнение логических, арифметических операций за один такт работы МП. Основной

недостаток трехшинной архитектуры - большая площадь, занимаемая шинами на кристалле ИС (до 25%). При одношинной организации МП площадь, занимаемая шиной, не превышает 10% всей площади кристалла, однако выполнение операций усложняется прежде всего за счет введения дополнительных регистров.

2. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

2.1 Практическое занятие №1 (2 часа).

Тема: «Выполнение операций в двоичном коде**»**

2.1.1 Задание для работы:

1. Выполнение операции сложения
2. Выполнение операции вычитания
3. Выполнение операции умножения
4. Выполнение операции деления

2.1.2 Краткое описание проводимого занятия:

Выполнение операции сложения в двоичном коде. Выполнение операции вычитания в двоичном коде. Выполнение операции умножения в двоичном коде. Выполнение операции деления в двоичном коде.

2.1.3 Результаты и выводы:

Даны два числа 1111_2и 110_2

1. Выполним сложение чисел:

+	1	1	1	1
		1	1	0
1	0	1	0	1

2. Выполним вычитание чисел:

-	1	1	1	1
		1	1	0
	1	0	0	1

3. Выполним умножение чисел:

	*	1	1	1	1	
				1	1	0
+		1	1	1	1	
	1	1	1	1		
1	0	1	1	0	1	0

4. Выполним деление чисел:

-	1	1	1	1		1	1	0
	1	1	0			1	0,	1
		-	1	1	0			
			1	1	0			
					0			

Задачи для решения.

1. Дано A=$A71_6$, B=251_8. Какое из чисел C, записанных в двоичной системе, отвечает условию A<C<B?
   1) 10101100_2
   2) 10101010_2
   3) 10101011_2
   4) 10101000_2
2. Вычислите сумму чисел X и Y, если X=110111_2, Y=135_8. Результат представьте в двоичном виде.

1) 11010100_2

2) 10100100_2

3) 10010011_2

4) 10010100_2

3. Вычислите значение выражения 206_8+ AF_{16}− 11001010_2. Вычисления производите в двоичной с/с. Переведите ответ в десятичную с/с.

2.2 Практическое занятие №2 (2 часа).

Тема: «**Минимизация логических функций**»

2.2.1 Задание для работы:

1. Представление логических функций в СКНФ, СДНФ.
2. Минимизация логических функций

2.2.2 Краткое описание проводимого занятия:

Научиться минимизировать совершенные дизъюнктивные нормальные формы логической функции**.**

2.2.3 Результаты и выводы:

Основные понятия.

1 Сложность логической функции, а отсюда сложность и стоимость реа-лизующей ее схемы (цепи), пропорциональны числу логических операций и числу вхождений переменных или их отрицаний. В принципе любая логическая функция может быть упрощена непосредственно с помощью аксиом и теорем логики, но, как правило, такие преобразования требуют громоздких выкла-док. Поэтому более целесообразно использовать специальные алгоритмические методы минимизации, позволяющие проводить упрощение функции более

про-сто, быстро и безошибочно.

2 Минимизация — представление функции в ДНФ или КНФ с мини-мальным количеством членов и минимальным набором переменных.

3 К методам минимизации относятся:

- метод Квайна;
- метод карт Карно;
- метод испытания импликант;
- метод импликантных матриц;
- метод Квайна-Мак-Класки и др.

4 Преобразование функции методом Квайна можно разделить на два эта-па:

- на первом этапе осуществляется переход от канонической формы (СДНФ или СКНФ) к так называемой сокращѐ нной форме;
- на втором этапе — переход от сокращѐ нной формы к минимальной форме.

Задание

Для заданной булевой функции *f (x, y, z):*

- найти двоичную форму булевой функции;
- составить СДНФ функции;
- минимизировать СДНФ функции.

Пример выполнения: $f = (x \vee z) \leftrightarrow (x \to y)$.

Решение:

1 Для того, чтобы найти двоичную форму булевой функции, составим таблицу истинности функции $f = (x \vee z) \leftrightarrow (x \to y)$:

x	Y	z	*∂1*	*∂2*	*∂3*	*∂4*	*∂5*
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1
0	1	1	1	1	0	1	1
1	0	0	0	0	1	1	0
1	0	1	0	1	1	1	1
1	1	0	0	0	0	0	1
1	1	1	0	1	0	0	0

По результатам в последней колонке $f(x, y, z) = (11110110)$

2 Составим СДНФ функции. Функция принимает значение 1 на наборах 000, 001, 010, 011, 101, 110. Нулю соответствует переменная с отрицанием, единице – без отрицания. Получим СДНФ:

$$f(x, y, z) = \bar{x}\bar{y}\bar{z} \vee \bar{x}\bar{y}z \vee \bar{x}y\bar{z} \vee \bar{x}yz \vee x\bar{y}z \vee xy\bar{z}$$

3. Минимизируем СКНФ функции, для этого:

- перегруппируем элементарные конъюнкции так, чтобы между дву-мя членами, содержащими одинаковые переменные, вхождения которых (пря-мые и инверсные) совпадали для всех переменных, кроме одной

$$f(x, y, z) = \underline{\underline{\bar{x}\bar{y}\bar{z} \vee \bar{x}\bar{y}z}} \vee \underline{\underline{\bar{x}y\bar{z} \vee \bar{x}yz}} \vee x\bar{y}z \vee xy\bar{z};$$

- последние два члена нельзя сгруппировать, но, используя закон идемпотентности ($A \vee A = A$), продублируем подходящие конъюнкции:

$$f(x, y, z) = \underline{\underline{\bar{x}\bar{y}\bar{z} \vee \bar{x}\bar{y}z}} \vee \underline{\underline{\bar{x}y\bar{z} \vee \bar{x}yz}} \vee \underline{\underline{x\bar{y}z \vee x\bar{y}z}} \vee \underline{\underline{xy\bar{z} \vee xyz}};$$

- в этом случае все переменные в паре, кроме одной, можно вынести за скобки

$$f(x, y, z) = \underline{\underline{\bar{x}\bar{y}}}\left(\underline{\underline{\bar{z} \vee z}}\right) \vee \underline{\underline{\bar{x}y}}\left(\underline{\underline{\bar{z} \vee z}}\right) \vee \underline{\underline{y\bar{z}}}\left(\underline{\underline{\bar{x} \vee x}}\right) \vee \underline{\underline{y\bar{z}}}\left(\underline{\underline{\bar{x} \vee x}}\right);$$

- а оставшиеся в скобках прямое и инверсное вхождение одной переменной подвергнуть склейке $x \vee \bar{x} = 1$

$$f(x, y, z) = \bar{x}\bar{y} \vee \bar{x}y \vee y\bar{z} \vee y\bar{z};$$

- при необходимости можно повторить действия:

$$f(x, y, z) = \bar{x}\bar{y} \vee \bar{x}y \vee y\bar{z} \vee y\bar{z} = \bar{x}\left(\bar{y} \vee y\right) \vee y\bar{z} \vee y\bar{z} = \bar{x} \vee y\bar{z} \vee y\bar{z}.$$

Задание к практической работе:

№	f(x, y, z)	№	f(x, y, z)	№	f(x, y, z)
1	$(x \to (y \downarrow z)) \oplus y$	11	$(x \vee \bar{z}) \to (\bar{y} \downarrow z)$	21	$((x \downarrow \bar{y}) \leftrightarrow z) \oplus y$
2	$((x \mid z) \to y) \oplus \bar{z}$	12	$(\bar{x} \vee y) \leftrightarrow (z \oplus x)$	22	$(y \vee z) \to (x \leftrightarrow \bar{y})$
3	$((x \downarrow y) \to z) \leftrightarrow y$	13	$(y \vee z) \to (x \leftrightarrow \bar{y})$	23	$(\bar{y} \to z) \leftrightarrow (x \downarrow y)$
4	$((y \mid z) \to \bar{x}) \leftrightarrow z$	14	$(x \vee \bar{y}) \to (\bar{z} \mid y)$	24	$(\bar{x} \wedge y) \to (\bar{z} \leftrightarrow x)$
5	$((x \downarrow y) \to \bar{z}) \oplus x$	15	$(z \to \bar{x}) \leftrightarrow (x \downarrow y)$	25	$(x \downarrow \bar{y}) \to (y \leftrightarrow \bar{x})$
6	$(y \vee z) \to (x \leftrightarrow \bar{y})$	16	$(x \vee \bar{z}) \downarrow (y \vee z)$	26	$(z \to x) \leftrightarrow (y \vee x)$
7	$(x \downarrow y) \oplus (\bar{y} \to z)$	17	$(x \wedge \bar{y}) \leftrightarrow (\bar{z} \vee y)$	27	$(x \leftrightarrow \bar{y}) \vee (z \to x)$
8	$((x \mid y) \to z) \leftrightarrow y$	18	$(\bar{x} \wedge z) \to (y \leftrightarrow x)$	28	$(x \to y) \wedge (\bar{y} \leftrightarrow z)$
9	$(\bar{x} \vee y) \to (y \leftrightarrow \bar{z})$	19	$(x \downarrow y) \oplus (\bar{y} \leftrightarrow z)$	29	$(x \vee \bar{z}) \downarrow (y \to z)$
10	$((x \leftrightarrow \bar{y}) \to z) \downarrow y$	20	$(\bar{x} \vee y) \leftrightarrow (z \to x)$	30	$(x \leftrightarrow y) \vee (\bar{z} \to x)$

2.3 Практическое занятие №3 (2 часа).

Тема: «**Аудиосистема ПК**»

2.3.1 Задание для работы:

1. Оцифровка звуковых сигналов
2. Методы синтеза звука
3. Трёхмерный звук

2.3.2 Краткое описание проводимого занятия:

Общие сведения. Принцип оцифровки звуковых сигналов. Метод синтеза звука. Трёхмерный звук. Достоинства и недостатки методов синтеза звука. Сфера применения объёмного звука.

2.3.3 Результаты и выводы:

Аналоговый сигнал с помощью специального процесса может быть представлен в виде **цифрового сигнала** – некоторой последовательности чисел. Таким образом, аналоговый звуковой сигнал может быть «введен» в компьютер, обработан цифровыми методами и сохранен на цифровом носителе в виде некоторого набора описывающих его дискретных значений.

Аналоговый или цифровой аудио сигнал – это лишь формы представления звуковых колебаний материи, придуманная человеком для того, чтобы иметь возможность анализировать и обрабатывать звук. Непосредственно аналоговый или цифровой сигнал в его исходном виде не может быть «услышан». Чтобы воссоздать закодированное в цифровых данных звучание, необходимо вызвать соответствующие колебания воздуха, потому что именно эти колебания и есть звук. Это можно сделать лишь путем организации вынужденных колебаний некоторого предмета, расположенного в воздушном пространстве (например, диффузора громкоговорителя). Колебания предмета вызывают колебаниями напряжения в

электрической цепи. Эти самые колебания напряжения и есть аналоговый сигнал. Таким образом, чтобы «прослушать» цифровой сигнал, необходимо вернуться от него к аналоговому сигналу. А чтобы «услышать» аналоговый сигнал нужно с его помощью организовать колебания диффузора громкоговорителя.

Компьютер – это цифровое устройство, то есть электронное устройство, в котором рабочим сигналом является дискретный сигнал. Сегодняшние компьютеры оперируют дискретными сигналами, несущими двоичные значения, условно обозначаемые как «да» и «нет» (на электрическом уровне: 0 вольт и V вольт, для некоторого ненулевого значения V). С помощью одного двоичного сигнала за один шаг можно передать информацию об одном из всего двух положений: 0 («да») или 1 («нет»). С помощью N двоичных сигналов за один шаг можно передать информацию об одном из 2^N положений (2^N – это число комбинаций нулей и единиц для N сигналов). Взаимодействие всех составляющих компьютер блоков происходит путем обмена и обработки одним или одновременно несколькими двоичными сигналами. Все коды управления, а также сама обрабатываемая информация, представляется в компьютере в виде чисел. По этой причине и аудио сигналы в цифровой аппаратуре представляют в виде чисел.

Создание (синтез) звука в основном преследует две цели: имитация различных естественных звуков (шум ветра и дождя, звук шагов, пение птиц и т. п.), а также акустических музыкальных инструментов (имитационный синтез), и получение принципиально новых звуков, не встречающихся в природе (чистый синтез).

Обработка звука обычно направлена на получение новых звуков из уже существующих (например, «голос робота»), либо придание им дополнительных качеств или устранение существующих (например, добавление эффекта хора, удаление шума или щелчков). Каждый из методов синтеза и обработки имеет свою математическую и алгоритмическую модель, что позволяет любой из них реализовать на компьютере; однако, многие методы, будучи реализованы точно, требуют слишком большого объема вычислений, отчего их обычно реализуют с какой-либо степенью допущения.

Рассмотрим основные методы синтеза звука.

1. Аддитивный (*additive*). Основан на утверждении Фурье о том, что любое периодическое колебание можно представить в виде суммы чистых тонов (синусоидальных колебаний с различными частотами и амплитудами).

Для этого нужен набор из нескольких синусоидальных генераторов с независимым управлением, выходные сигналы которых суммируются для получения результирующего сигнала. На этом методе основан принцип создания звука в духовом органе.

Достоинства метода: позволяет получить любой периодический звук, и процесс синтеза хорошо предсказуем (изменение настройки одного из генераторов не влияет на остальную часть спектра звука). Основной недостаток – для звуков сложной структуры могут потребоваться сотни генераторов, что достаточно сложно и дорого реализовать. Для снижения стоимости реализации вместо набора отдельных генераторов (реальных или математических) применяется обратное преобразование Фурье.

2. Разностный (*subtractive*). Идеологически противоположен первому. В основу положена генерация звукового сигнала с богатым спектром (множеством частотных составляющих) с последующей фильтрацией (выделением одних составляющих и ослаблением других) – по этому принципу работает речевой аппарат человека. В качестве исходных сигналов обычно используются меандр (прямоугольный,*square*), с переменной скважностью (отношением всего периода к положительному полупериоду), пилообразный (*saw*) – прямой и обратный, и треугольный (*triangle*), а также различные виды шумов (случайных непериодических колебаний). Основным органом синтеза в этом методе служат управляемые фильтры: резонансный (полосовой) – с изменяемым положением и шириной полосы пропускания (*band*) и фильтр нижних частот (ФНЧ) с изменяемой частотой среза (*cutoff*). Для каждого фильтра также регулируется добротность (*Q*) – крутизна подъема или спада на резонансной частоте.

Достоинства метода – относительно простая реализация и довольно широкий диапазон синтезируемых звуков. На этом методе построено множество студийных и концертных синтезаторов.

Недостаток – для синтеза звуков со сложным спектром требуется большое количество управляемых фильтров, которые достаточно сложны и дороги.

3. Частотно-модуляционный (*frequency modulation – FM*). В основу положена взаимная модуляция по частоте между несколькими синусоидальными генераторами. Каждый из таких генераторов, снабженный собственными формирователем амплитудной огибающей, амплитудным и частотным вибрато, именуется оператором. Различные способы соединения нескольких операторов, когда сигналы с выходов одних управляют работой других, называются алгоритмами синтеза. Алгоритм может включать один или больше операторов, соединенных последовательно, параллельно, последовательно-параллельно, с обратными связями и в прочих сочетаниях – все это дает практически бесконечное множество возможных звуков.

Благодаря простоте цифровой реализации, метод получил широкое распространение в студийной и концертной практике. Однако практическое использование этого метода достаточно сложно из-за того, что большая часть звуков, получаемых с его помощью, представляет собой шумоподобные колебания, и достаточно лишь слегка изменить настройку одного из генераторов, чтобы чистый тембр превратился в шум. Однако метод дает широкие возможности по синтезу разного рода ударных звуков, а также – различных звуковыхэффектов, недостижимых в других методах разумной сложности.

Сферы применения объёмного звука.

Технологии многоканального пространственного звучания могут использоваться при производстве самого разнообразного содержимого: музыки, речи, натуральных или искусственных звуков для кино, телевидения, радиовещания или компьютерных игр и приложений. В музыкальных представлениях могут использоваться многоканальные технологии на открытых площадках, стадионах, концертных залах, в музыкальных театрах и для целей вещания. В кинематографе подобные технологии используются в кинотеатрах или дома (так называемый домашний кинотеатр). Помимо киноиндустрии объёмный звук может применяться на выставках, постановках в театрах и образовательных целях. Другая область применения включает игровые приставки, персональные компьютеры и другие платформы.

2.4 Практическое занятие №4 (2 часа).

Тема: «**Коммуникационные устройства**»

2.4.1 Задание для работы:

1. Модемы и факс-модемы
2. Подключение к проводным сетям

2.4.2 Краткое описание проводимого занятия:

Стандарты факс-модемов. Модем. Проводные сети.

2.4.3 Результаты и выводы:

Модем (модулятор-демодулятор) служит для передачи информации на большие расстояния, недоступные локальным сетям, с использованием выделенных и коммутируемых телефонных линий.

Fax-modem (факс-модем) позволяет передавать и принимать факсимильные изображения, совместимые с обычными факс-машинами. Совместимость средств факсимильной связи в глобальных масштабах обеспечивается стандартами CC1TT. Существуют следующие стандарты:

Fax Group 1, 11 — устаревшие стандарты аналоговой передачи изображений. *Fax: Group III* — современный стандарт, использующий алгоритмы цифрового сжатия данных, передаваемых по аналоговым телефонным линиям. Скорость передачи 14,400 или 9,600 бод (может снижаться при ухудшении качества связи до 4,800 бод).*Fax Group IV* — стандарты для передачи изображений по каналам цифровой связи (сети ISDN).

Сегодня используется два типа соединения: **проводной и беспроводной.**

В качестве проводной связи может служить практически любой вид кабеля. Как правило, это коаксиальный, оптоволоконный или кабель на основе витой пары. Из более экзотических – электрический и телефонный кабели.

2.5 Практическое занятие №5 (2 часа).

Тема: «**Организация системы прерываний**»

2.5.1 Задание для работы:

1. Общие сведения о системах прерывания
2. Характеристики систем прерывания
3. Возможные структуры систем прерывания

2.5.2 Краткое описание проводимого занятия:

Общие сведения о системах прерывания. Характеристики систем прерывания. Возможные структуры систем прерывания.

2.5.3 Результаты и выводы:

Организация системы прерываний.

Назначение системы прерываний – это быстрая реакция на события, происходящие как внутри ЭВМ, так и за ее пределами. Прерывания, вызванные вне ЭВМ, вызываются внешними устройствами, периферийными устройствами (напр. клава). Под операцией прерывания понимается прекращение процесса выполнения текущей проги и переключение процессора на выполнение другой проги, обслуживающей причину прерывания.Основное назначение прерываний – это реагирование на критические события. По завершению обслуживания прерывания, прерванный процесс необходимо восстановить и продолжить (с того места, на котором он был прерван). С целью запоминания процесса необходимо собрать всю информацию о текущем состоянии. Данную информацию обычно называют вектор состояния процесса. Он включает слово состояния процессора (ССП), которое включает адрес следующей команды и признаки состояния (различные флаги, значения переменных, значение регистров общего назначения (РОН) и т.д.). Сам вектор состояния процесса располагается в области оперативной памяти, которая обычно организована как стек.Операция прерывания сводится к выполнению следующих действий:

1) выполнение текущей проги прерывается. Прерывание осуществляется по завершении i-той команды проги. Выполнение команды i+1 не начинается.
2) запоминается информация о состоянии процессора в виде ССП (адрес следующей команды i+1, признаки результата, различные флаги, маски и т.д.).
3) управление передается специальной проге (обработчику), обслуживающей данную причину прерывания. Данная прога выполняет все действия, необходимые для данного прерывания.

Обработчик прерываний.Типовая структура обработчика прерываний состоит в следующем:

1) сохранение используемых в регистре общего назначения данных в ОП.
2) обработка прерывания.
3) восстановление регистров общего назначения (из ОП в РОН).

Причины формирования прерываний.

Все причины делят на внутренние и внешние. К внутренним причинам относят два типа:

1) это события, при возникновении которых нормальное продолжение выполняемого процесса невозможно (или бессмысленно).
2) причины, естественные для выполняемого процесса.

К внешним причинам (события которые возникают вне вычислительной системы) относят:

1) от неких внешних устройств ввода/вывода (напр. клавиатурное прерывание).
2) от различных таймеров.
3) другие вычислительные системы (ПК, микро ЭВМ и т.д.).

2.6 Практическое занятие №6 (2 часа).

Тема: «**Организация перехода к прерывающей программе**»

2.6.1 Задание для работы:

1. Процедуры перехода к прерывающей программе
2. Реализация фиксированных приоритетов
3. Реализация программно-управляемых приоритетов

2.6.2 Краткое описание проводимого занятия:

Процедуры перехода к прерывающей программе. Реализация фиксированных приоритетов. Реализация программно- управляемых приоритетов.

2.6.3 Результаты и выводы:

Организация перехода к. прерывающей программе. Приоритетное обслуживание запросов прерывания. Назовем *вектором прерывания* вектор начального состояния прерывающей программы. Вектор прерывания содержит всю необходимую информацию для перехода к прерывающей программе, в том числе ее начальный адрес. Каждому запросу (уровню) прерывания, а в ряде случаев, например в малых и микроЭВМ и микропроцессорах, каждому периферийному устройству соответствует свой вектор прерывания, способный инициировать выполнение соответствующей прерывающей программы. Векторы прерывания обычно находятся в специально выделенных фиксированных ячейках памяти.

Главное место в процедуре перехода к прерывающей программе занимают передача из соответствующего регистра (регистров) процессора в память (в частности, в стек) на сохранение текущего вектора состояния прерываемой программы (чтобы можно было вернуться к ее исполнению) и загрузка в регистр (регистры) процессора вектора прерывания прерывающей программы, к которой при этом переходит управление процессором.

Процедура организации перехода к прерывающей программе включает в себя выделение из выставленных запросов такого, который имеет наибольший приоритет.

Различают *абсолютный* и, *относительный приоритеты*. Запрос, имеющий абсолютный приоритет, прерывает выполняемую программу и инициирует выполнение соответствующей прерывающей программы. Запрос с относительным приоритетом является первым кандидатом на обслуживание после завершения выполнения текущей программы.

Если наиболее приоритетный из выставленных запросов прерывания не превосходит по уровню приоритета выполняемую процессором программу, то запрос прерывания игнорируется или его обслуживание откладывается до завершения выполнения текущей программы.

Простейший способ установления приоритетных соотношений между запросами (уровнями) прерывания состоит в том, что приоритет определяется порядком присоединения линий сигналов запросов ко входам системы прерывания. При появлении нескольких запросов прерывания первым воспринимается запрос, поступивший на вход с меньшим номером. В этом случае приоритет является жестко фиксированным. Изменить приоритетные соотношения можно лишь пересоединением линий сигналов запросов на входах системы прерывания.

2.7 Практическое занятие №7 (2 часа).

Тема: «**Принципы организации ввода-вывода**»

2.7.1 Задание для работы:

1. Общие принципы организации ввода-вывода
2. Программный ввод-вывод
3. Ввод-вывод по прерываниям
4. Ввод-вывод в режиме прямого доступа к памяти

2.7.2 Краткое описание проводимого занятия:

Общие принципы организации ввода-вывода. Программный ввод-вывод. Ввод-вывод по прерываниям. Ввод-вывод в режиме прямого доступа к памяти.

2.7.3 Результаты и выводы:

Основные принципы обслуживания запросов прерывания от периферийных устройств.

Периферийные устройства, при необходимости передачи информации в память компьютера, или ее приема от него, посылают в контроллер прерывания запрос прерывания в виде электрического сигнала. Эти запросы фиксируются в регистре запросов прерываний контроллера, а в процессор посылается общий запрос прерывания – сигнал INTR, который определяет наличие в данный момент хотя бы одного незамаскированного запроса прерывания от периферийных устройств. Процессор, в конце цикла выполнения каждой очередной команды опрашивает этот бит INTR и, в случае наличия разрешения приема запросов прерывания (IF = 1), приостанавливает процесс выполнения текущей программы и запоминает состояние процессора на данный момент времени (как минимум содержимое регистров CS, IP и регистра флагов Flags). После этого процессор посылает контроллеру прерывания сигнал готовности выполнять процесс обслуживания прерывания – INTA (InterruptAcknowledge). Получив этот сигнал, контроллер прерываний выдает процессору код типа прерывания от источника, который имеет наивысший приоритет изо всех принятых к этому времени запросов на прерывания. Этот код представляет собой 8-ми разрядное слово, определяющее логический адрес (вектор) процедуры обработки данного прерывания, хранящийся в таблице векторов прерывания, расположенной в первом килобайте адресного пространства оперативной памяти.

Получив код типа прерывания, процессор по указанному коду извлекает логический адрес процедуры, формирует из него физический адрес и передает управление процедуре обслуживания данного прерывания. А эта процедура заключается в передаче в периферийное устройство, запросившее прерывание, или приеме из периферийного устройства адресуемых данных. Таким образом, ввод/вывод по прерываниям позволяет существенно сократить непроизводительное время процессора, затрачиваемое на опрос и ожидание готовности периферийных устройств к приему или выдачи информации. Однако сам процесс ввода или вывода происходит под управлением процессора и, следовательно, при операциях ввода/вывода процессор все равно отвлекается от своей основной обязанности – обработки информации. Поэтому, когда приходится вводить или выводить большие объемы данных, например, при обмене информации с дисками при свопинге, или при одновременном вводе больших объемов данных при проведении научно-технических

экспериментов, то отвлечение процессора от вычислительных задач, особенно при мультипрограммном режиме работы процессора, крайне нежелательно. При этом следует учитывать, что обмен большими объемами информации осуществляется всегда блоками между периферийными устройствами и оперативной памятью компьютера.

Поэтому, в таких случаях используется более эффективная система ввода/вывода информации путем прямого доступа к памяти.

Ввод вывод в режим прямого доступа к памяти используют канал прямого доступа к памяти по которому массивы данных передаются непосредственно между периферийным устройством и АЗУ, минуя микропроцессор. Это позволяет достичь наибольшей скорости передачи, но требует специального контроллера прямого доступа к памяти. Периферийное устройство посылает в контроллер прямого доступа к памяти запрос, который транслируется контроллером в микропроцессор, на что микропроцессор отвечает сигналом подтверждения. При этом микропроцессор прекращает работу по выполнению текущей программы, переводит свои буферные регистры, подключенные к шине адреса и шине данных, в высоко эмпедансное состояние и прекращает выработку управляющих сигналов. В контроллер прямого доступа к памяти в его счетчики заносится адрес ячейки АЗУ с которой начинается массив данных и число слов в нем, а также взводится триггер прямого доступа к памяти. Адрес ячейки выдается затем на шину адреса и начинается обмен. При передаче каждого слова содержимое счетчиков в контроллере изменяется на единицу и обмен данными производиться автоматически пока триггер прямого доступа к памяти не сбросится сигналом переполнения или обнуления счетчика. Запрос снимается и микропроцессор возобновляет приостановленную программу.