МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ

Методические указания к лабораторным работам для студентов направления **“Информатика и вычислительная техника”**

(уровень бакалавриата)

2018

Замятин Николай Владимирович

Организация ЭВМ и систем: методические указания к лабораторным работы для студентов направления подготовки “ **Информатика и вычислительная техника**” (уровень бакалавриата / Н.В. Замятин. – Томск, 2018- с 90.

Содержание

1. Введение

Методические указания к лабораторному практикуму по дисциплине " Организация ЭВМ и систем" содержат описания 8 четырехчасовых лабораторных работ.

Лабораторные работы расположены последовательно, начиная от аппаратных реализаций элементов ЭВМ и затем программное управление, а также представлены современные параллельные системы в виде нейронных сетей. Включены работы по управлению таймером и звуком, управлению видеоадаптером в текстовом режимах, синтезу персептрона, сетевым технологиям

Лабораторные работы ориентированы на использование ПЭВМ, совместимых с микропроцессорами семейства 8086.

Цель лабораторного курса

Целью лабораторных работ является закрепление практических навыков управления вычислительными системами на низком уровне путем программирования адаптеров устройств на уровне портов с непосредственной адресацией к видеопамяти.

Организация и проведение лабораторных работ

Студенты группы обьединяются в бригады по 2-3 человека, работающие на закрепленном компьютере. Каждый студент получает индивидуальное задание в соответствии с номером в журнале и оформляет отчет по лабораторной работе.

Выполнение лабораторной работы предполагает предварительное изучение соответствующего раздела курса и методических указаний к очередной работе.

Для допуска к выполнению лабораторной работы студент должен ознакомиться с темами для проработки и предварительно написать текст программы, в соответствии с индивидуальным заданием.

Для некоторых лабораторных работ текст программы составляется на ассемблере по указанию преподавателя и с учетом глубины знаний студентами этого языка.

В течение выполнения лабораторной работы студент должен ответить на контрольные вопросы по предыдущей лабораторной работе. К лабораторной работе не допускаются студенты, не сдавшие более двух лабораторных работ.

Пропущенные лабораторные работы выполняются в конце семестра.

В процессе выполнения лабораторных работ следует ограничить перемещения в лаборатории.

Лабораторная работа №1. Синтез операционного автомата

Цель работы:

Понять, каким образом выполняются арифметическо - логические операции в микропроцессоре. Научиться синтезировать операционный автомат (ОА) для выполнения арифметических операций сложения и умножения.

Основные понятия

Любой универсальный преобразователь информации состоит из следующих операционных устройств (ОУ):

- арифметическо-логического устройства,
- памяти
- устройств ввода-вывода
- шин передачи сигналов.

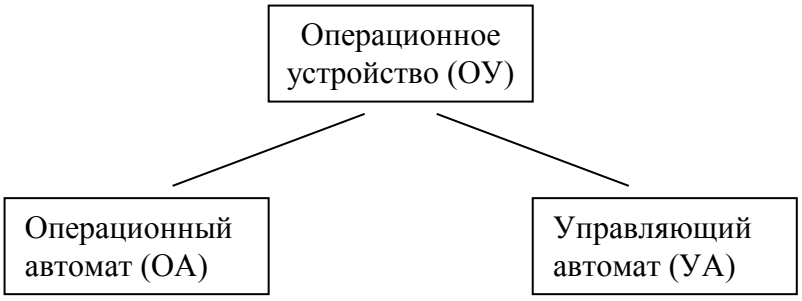
Согласно концепции академика Глушкова В.М. любое операционное устройство декомпозируется на две части - пассивную исполнительную (ОА) и активную управляющую (УА).

АЛУ часть узлов (сумматор и др. КС) являются исполнителями, а часть узлов (распределитель сигналов) являются управляющими. У этих узлов разные принципы построения, организации. Поскольку принципы организации разные, то и их проектирование тоже раздельное.

Операционное устройство представляет собой конечный автомат, состоящий из двух частей:

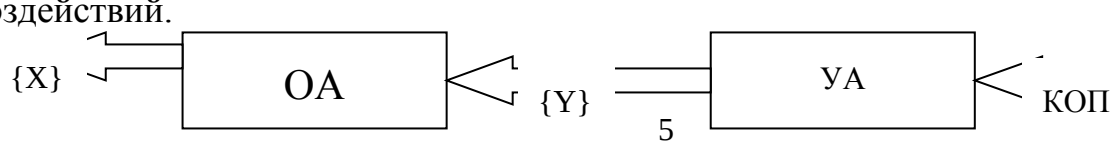
1.) операционный автомат;

2.) управляющий автомат;



Операционный автомат (ОА) – предназначен для арифметических и логических операций под управлением управляющего автомата.

Управляющий автомат (УА) – предназначен для формирования управляющих воздействий.



КОП – код операции;
{X} – входные сигналы;
{Y} – управляющие команды;
{Z} – выходные сигналы;
{x} – множество осведомительных сигналов (количество, переполнение и т. п.);
Основа функционирования – принцип микропрограммного управления

1. Любая операция это сложное действие, состоящее из совокупности элементарных действий, называемых микрооперациями (МО). Выполнение каждой МО осуществляется специальной комбинационной схемой (КС) за один такт машинного времени.
2. Порядок выполнения МО определяется алгоритмом операции и зависит от значений логических условий x. ЛУ принимают значения истина или ложь в зависимости от значений операндов.
3. Алгоритм, представленный, записанный в терминах МО и ЛУ, называется микропрограммой (МП). МП задает порядок выполнения МО и проверки ЛУ во времени.
4. Совокупность микропрограмм $МП_1$, …, $МП_g$ задает функцию ОУ.

Все команды, формируемые УА называются микрокомандами (запись в триггер, считывание кода, сдвиг).
Rg A, Rg B – входные регистры;
S – сумматор;
Rg C – выходной регистр;
Перечень микроопераций
y_1 – запись через управляющую шину во входные регистры A и B;
y_2 – проверка знака числа (сдвиг для определения знакового разряда);
y_3 – считывание кодов из регистров;
y_4 – сложение;
y_5 – проверка знака суммы;
y_6 – выдача результата на выходную шину;
Совокупность микрокоманд образует микропрограмму действия ОА под воздействием микрокоманды, называемой микрооперацией.
Любая математическая илм логическая операция в машинном исполнении состоит из набора простейших микроопераций: сложения и сдвиога. Следовательно и ОА предназначен для выполнения двух простейших операций (сложения и сдвига). Поэтому в нём должны присутствовать два основных устройства – сумматор и регистры. Структурная схема операционного автомата приведена ниже.

По управляемой входной шине поступают операнды A и B, которые записываются соответственно в регистры Rg A и Rg B. Вэтих регистрах над операндами могут выполняться микрооперации сдвига и инвертирования

В определенный момент времени содержимое регистров поступает на сумматор, где выполняется операция суммирования. Результат операции записывется в регистр Rg C и затем поступает на выходную шину. Для формирования условий (признаков) в ОА имеется триггер и счетчик циклов.

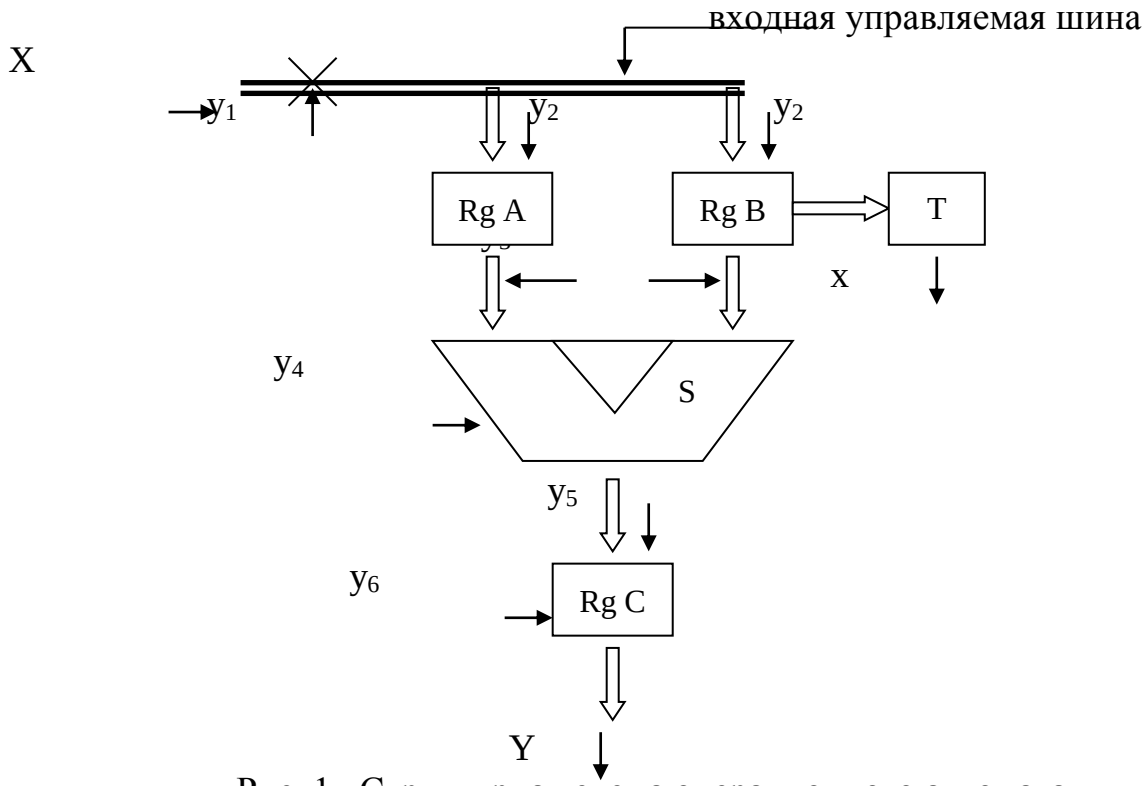
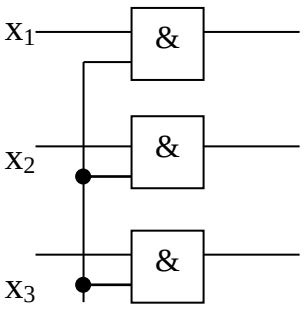


Рис. 1. Структурная схема операционного автомата

Элементы базиса построения операционного автомата:
Управляемая шина



1 – управляющая команда

Микрооперации y_1, y_2, y_3, y_4, y_5, y_6 представляющие одноразрядные сигналы, и поступающие на входы регистров и сумматоров, формирует УА

Обработка информации с помощью ОУ осуществляется путем выполнения операций из списка F в последовательности, которая задается алгоритмом (программой) решения задачи: ОА, выполняя программу, распределяет выполнение операций, предписанных командами программы, между различными ОУ - АЛУ, контроллерами ПУ.

Описание входных и выходных слов и массивов. Слово описывается своим именем и длиной: $C(n_1{:}n_2)$. Здесь С - имя слова, n_1, n_2 - номера старшего и младшего разрядов слова соответственно. Часть слова называется **полем** и описывается аналогично словам: A(0:7), A(0:15), B(15), A(0) и т.п.

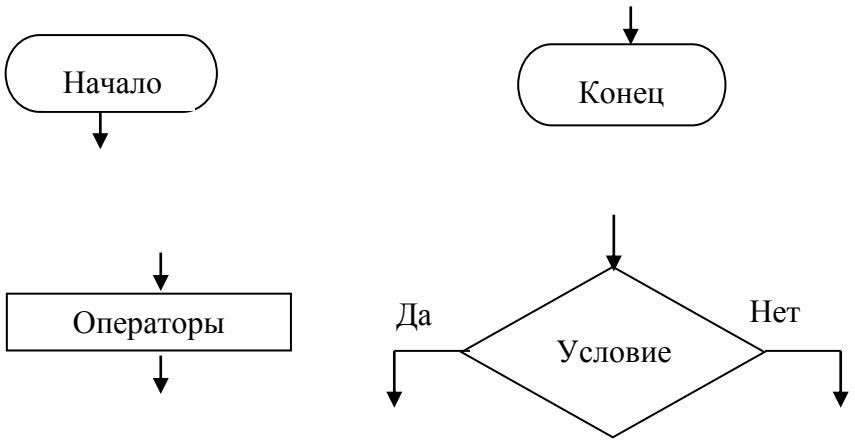
Массивы слов (например, запоминающее устройство) описывается в виде: $M[m_1{:}m_2](n_1{:}n_2)$. Здесь М - имя массива, m_1,m_2 - номера первого и последнего элемента (ячейки) массива соответственно, n_1, n_2 - определены выше. Пример описания локальной памяти как массива регистров: ЛП[0:15](0:31) - 16 тридцатидвухразрядных регистров.

Описание МО. Для описания МО используется оператор присваивания «:=» (или «←»). Слева от оператора указывается слово, поле, составное слово или элемент массива. Справа - двоичное выражение, которое описывает правило получения результата МО. Запись двоичного выражения осуществляется при помощи символов, обозначающих различные операции над операндами (словами), представленными в двоичной форме. Например, + - сложение кодов, $\vee$ - логическая операция или и т.п. Примеры - в МП умножения.

Описание ЛУ. Для описания ЛУ используется различного рода отношения: ”<” - меньше, ”>” - больше, ”=0” - равно нулю, “≠0” - не равно нулю и т.п. Примеры: A<0, B≥0, C=0 и т.п. (смотри МП умножения).

Порядок выполнения МО. Порядок выполнения МО и проверки ЛУ задается в графической форме - в виде так называемой **граф-схемы алгоритма** (ГСА). ГСА строится с использованием вершин четырех типов: начальной, конечной, операторной и условной и дуг, связывающих эти вершины (рисунок 5.3).

Начальная вершина имеет одну выходящую дугу. Конечная вершина имеет одну входящую дугу. Операторная вершина имеет одну входящую и одну выходящую дугу. В ней записывается один или несколько операторов присваивания, описывающих МО.



Условная вершина имеет одну входящую и две исходящих, отмеченных символами «да» (1) и «нет» (0). Выход по дуге «да» осуществляется в случае, если ЛУ принимает истинное значение (1), и по дуге «нет» - если ложное значение (0).

Назначение ОА - выполнение микроопераций из списка $Y=\{y_1,\ldots,y_M\}$ под воздействием управляющих сигналов $y_m \in Y$ и формирование значений логических условий (осведомительных сигналов) $X=\{x_1, \ldots, x_L\}$.

С каждым сигналом $y_m \in Y$ в ОА отождествляется определенная МО. Поступление сигнала y_3 в ОА приводит к выполнению этой МО и записи её результата в С.

С каждым логическим условиям $x_l \in X$ в ОА отождествляется значение осведомительного сигнала, который принимает значение истина (единица) или ложь (ноль).

Ход работы

Для синтеза ОА в лабораторной работе необходимо выполнить следующую последовательность действий:

1. Синтезировать RS –триггер,
2. Синтезировать параллельный регистр,
3. Синтезировать сумматор,
4. Собрать триггер, регистр и сумматор в среде Electronics Workbench, промоделировать их работу. Привести процедуры синтеза, схемы и эпюры в отчете,
5. Выполнить процедуру операции сложения двух трехразрядных чисел без знака,
6. Собрать операционный автомат в среде Electronics Workbench, используя модели регистров и сумматора, имеющиеся в библиотеке системы.
7. Промоделировать выполнение арифметических процедур на операционном автомате, путем подачи управляющих воздействий на соответствующие входы регистров и сумматора,
8. Подготовить отчет с описанием результатов выполненной работы и ответы на вопросы.

Пример синтеза RS-триггера.

Синтезировать логическую схему триггера.

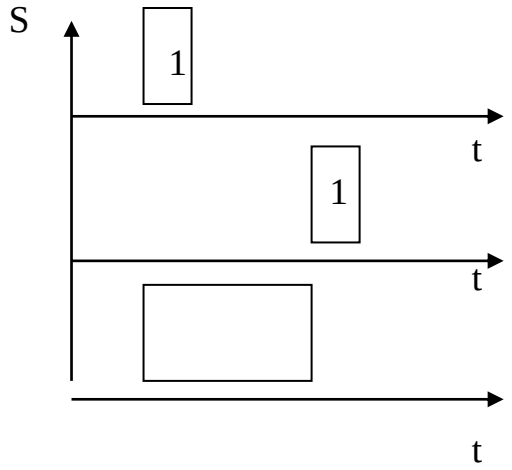
последовательность действий:

- уяснить задачу
- составить таблицу истинности
- получить булеву функцию
- минимизировать булеву функцию
- взять заданный базис (и - не либо или - не)
- составить логическую схему триггера
- произвести проверку
- изобразить эпюры сигналов

асинхронный RS – триггер.

S – SET (установка);
R – RESET (сброс);
Q(t) – состояние триггера;
Таблица истинности.

R	S	Q(t)	Q(t+1)	
0	0	0	0	} т. к. на входе ничего нет.
0	0	1	1	
0	1	0	1	} установка единицы.
0	1	1	1	
1	0	0	0	} установка нуля.
1	0	1	0	
1	1	0	-	} не используются.
1	1	1	-	



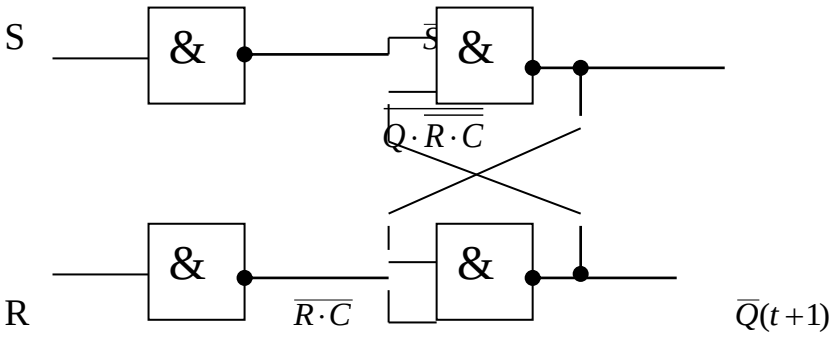
Эпюры выходных сигналов

Методом карт Карно определяем булеву функцию:

S \ R	$\overline{R}$	$\overline{R}$	R	R
	6	7	3	2
$\overline{Q}$	0	0	0	1
	4	5	1	0
Q	-	-	1	1

$$Q(t+1) = S + \overline{R} \cdot \overline{Q};$$

Выберем базис на элементах И – НЕ:

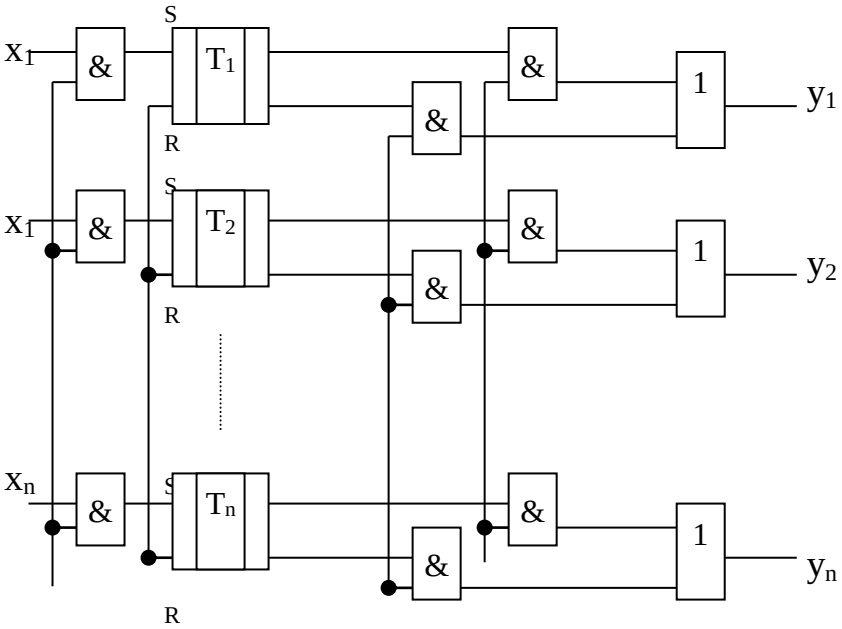


Проверка работы схемы:

C=0; S=0; R=0; Q(t)=0; Q(t+1)=0.

Синтезировать параллельный регистр, схема которого представлена на рисунке

Параллельный регистр



сброс считывание в прямом коде

запись считывание в обратном коде

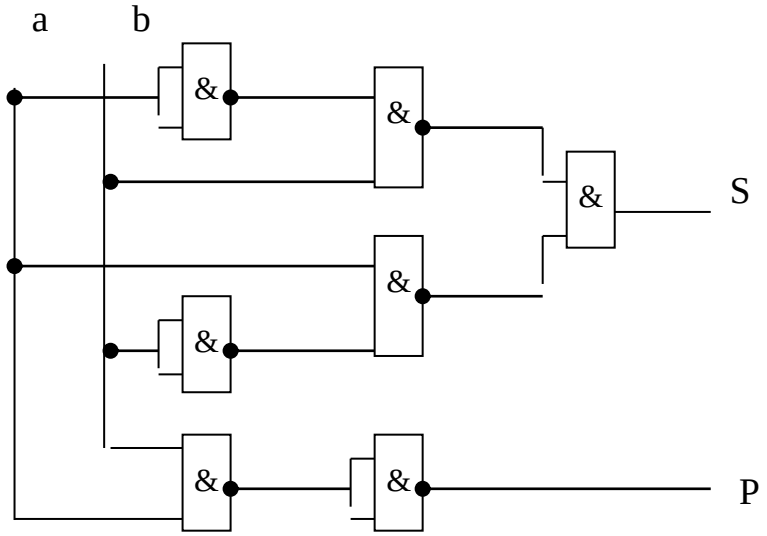
T_n – однотактный RS - триггер.

) Сумматор

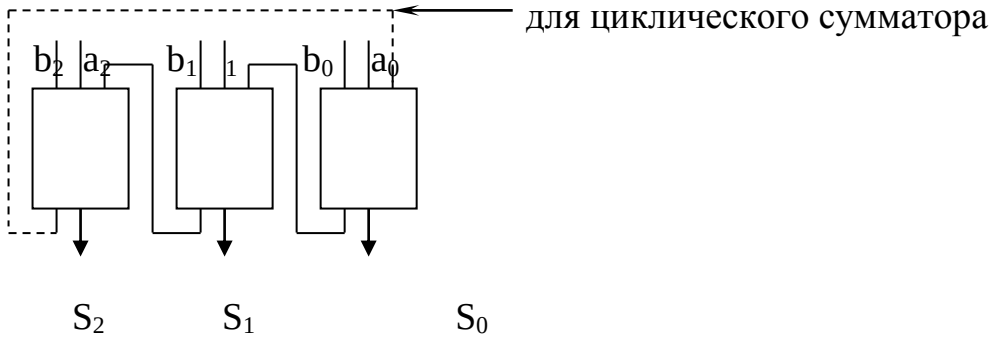
a	b	S	P
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = \overline{\overline{\overline{a} \cdot b + a \cdot \overline{b}}} = \overline{\overline{\overline{a} \cdot b} \cdot \overline{a \cdot \overline{b}}};$$
$$P = a \cdot b;$$

Одноразрядный сумматор



Трёхразрядный сумматор



Выполнение арифметических операций

Двоичная арифметика

Правила выполнения арифметических действий над двоичными числами определяются арифметическими действиями над одноразрядными двоичными числами.

Сложение	Вычитание	Умножение
$0+0=0$	$0-0=0$	$0*0=0$
$0+1=1$	$1-0=0$	$1*0=0$
$1+0=1$	$1-1=0$	$0*1=0$
$1+1=10$	$10-1=1$	$1*1=1$

↑↵

перенос в старший разряд

Правила выполнения арифметических действий во всех позиционных системах счисления аналогичны.

Сложение

Как и в десятичной системе счисления, сложение двоичных чисел начинается с правых (младших) разрядов. Если результат сложения цифр МЗР обоих слагаемых не помещается в этом же разряде результата, то происходит перенос. Цифра, переносимая в соседний разряд слева, добавляется к его содержимому. Такая операция выполняется над всеми разрядами слагаемых от МЗР до СЗР.

Вычитание

Операция вычитания двоичных чисел аналогична операции в десятичной системе счисления. Операция вычитания начинается, как и сложение, с МЗР. Если содержимое разряда уменьшаемого меньше содержимого одноименного разряда вычитаемого, то происходит заем 1 из соседнего старшего разряда. Операция повторяется над всеми разрядами операндов от МЗР до СЗР.

Второй вариант операции вычитания - когда уменьшаемое меньше вычитаемого. В этом случае отрицательное число представляется в дополнительном коде.

Умножение

Как и в десятичной системе счисления, операция перемножения двоичных многоразрядных чисел производится путем образования частичных произведений и последующего их суммирования. Частичные произведения формируются в результате умножения множимого на каждый разряд множителя, начиная с МЗР. Каждое частичное произведение смещено относительно предыдущего на один разряд. Поскольку умножение идет в двоичной системе счисления, каждое частичное произведение равно либо 0 (если в соответствующем разряде множителя стоит 0), либо является копией множимого, смещенного на соответствующее число разрядов влево (если в разряде множителя стоит 1). Поэтому умножение двоичных чисел идет путем сдвига и сложения. Таким образом, количество частичных произведений определяется количеством единиц в множителе, а их сдвиг - положением единиц (МЗР частичного произведения совпадает с положением соответствующей единице в

множителе). Положение точки в дробном числе определяется так же, как и при умножении десятичных чисел.

Операция умножения состоит в формировании суммы частичных произведений, которые суммируются с соответствующими сдвигами друг относительно друга. Этот процесс суммирования можно начинать либо с младшего, либо со старшего частичного произведения. В ЭВМ процессу суммирования придают последовательный характер, т.е. формируют одно частичное произведение, к нему с соответствующим сдвигом прибавляют следующее и т.д. (т.е. не формируют все частичные произведения, а потом их складывают). В зависимости от того, с какого частичного произведения начинается суммирование (старшего или младшего), сдвиг текущей суммы осуществляется влево или вправо. При умножении целых чисел для фиксации результата в разрядной сетке число разрядов должно равняться сумме числа разрядов в X и Y.

Содержание отчета:

1. Титульный лист
2. Цель работы
3. Ход работы

RS-триггер

Многоразрядный сумматор

Регистр

Описание арифметической операции, согласно задания,

Схема операционного автомата.

4. Ответы на вопросы

Вопросы к лабораторной работе

1. Принцип академика Глушкова В.М.
2. Какова разрядность управляющих сигналов, поступающих на ОА
3. Основные функции ОА
4. Какие основные микрооперации реализуются в ОА
5. В чем заключается принцип микропрограммирования
6. Каким образом меняется структура ОА для выполнения различных арифметическо-логических операций.

Лабораторная работа № 2 Синтез управляющего автомата

Цель работы:

Понять, каким образом выполняются арифметическо - логические операции в микропроцессоре. Научиться синтезировать управляющий автомат (УА) для выполнения арифметических операций сложения и умножения.

Основные понятия

Типы УА:

1.) УА с жёсткой логикой;

2.) УА с программируемой логикой;

УА с жёсткой логикой вырабатывает управляющие воздействия, путем срабатывания триггеров, число которых определяется числом устойчивых состояний в граф-схеме алгоритма, и входными условиями.

В УА с программируемой логикой коды микроопераций выбирается из ячеек ПЗУ, исходя из входных условий,

Пример схемы алгоритма (ГСА) – представлен на рис.2. Здесь Сч - счетчик циклов.

Операция умножения разделяется на 7 МО, основные из которых сложение (реализуется за один такт комбинационным двоичным сумматором) и сдвиг (реализуется регистром сдвига). Порядок выполнения МО зависит от значений двух ЛУ (осведомительных сигналов): B(0), СЦ=0. ГСА умножения задает порядок выполнения МО и проверки ЛУ во времени. Например, в зависимости от B(0) в следующий момент времени, в следующем такте будет выполняться либо МО сложения C:=C+A (если B(0)=1), либо МО сдвига, если B(0)=0.

Обработка информации с помощью ОУ осуществляется путем выполнения операций из списка F в последовательности, которая задается алгоритмом (программой) решения задачи: ОА, выполняя программу, распределяет выполнение операций, предписанных командами программы, между различными ОУ - АЛУ, контроллерами ПУ.

Запуск (инициализация) операции $f_g \in F$ осуществляется путем подачи кода операции в ОУ из УУ. Реализация операции f_g осуществляется путем выполнения МО в порядке, заданном микропрограммой, хранимой внутри ОУ (т.е. без участия УУ). Работа (функционирование) ОУ во времени осуществляется тактами. Реализация МПg в общем случае занимает различное количество тактов n, т.е. время выполнения операции $t_g = nT$, где T-продолжительность такта.

Принцип микропрограммного управления является основой организации (построения) ОУ. Эти процедуры достаточно схожи с фон Неймановскими принципами функционирования. В основе управления лежит алгоритм. Только у

Неймана он представляется в виде макропрограммы и поступает в процессор извне (из ОЗУ извлекается процессором). Здесь алгоритм в виде МП уже находится внутри ОУ. При выполнении программы ЦП генерирует определенную последовательность операций, реализуемых ОУ. При выполнении операции f_g ОУ генерирует последовательность МО, реализуемых комбинационными схемами КС.

Отличия между этими принципами: 1)операция - сложное действие, для реализации которого необходимо ОУ. МО - элементарное действие, для реализации которого достаточно КС. 2)Операция выполняется за n тактов: $t_{опер}=nT$. МО выполняется за один такт (алгоритм – в структуре КС). КС управляется данными на ее входах.

Функция ОУ определяется совокупностью микропрограмм $МП_1$, …, $МП_G$, описывающих алгоритмы операций f_1, …, f_g. Для описания МП в языке используются различные средства, обеспечивающие описание слов, МО, ЛУ, а также средства, описывающие порядок их выполнения во времени.

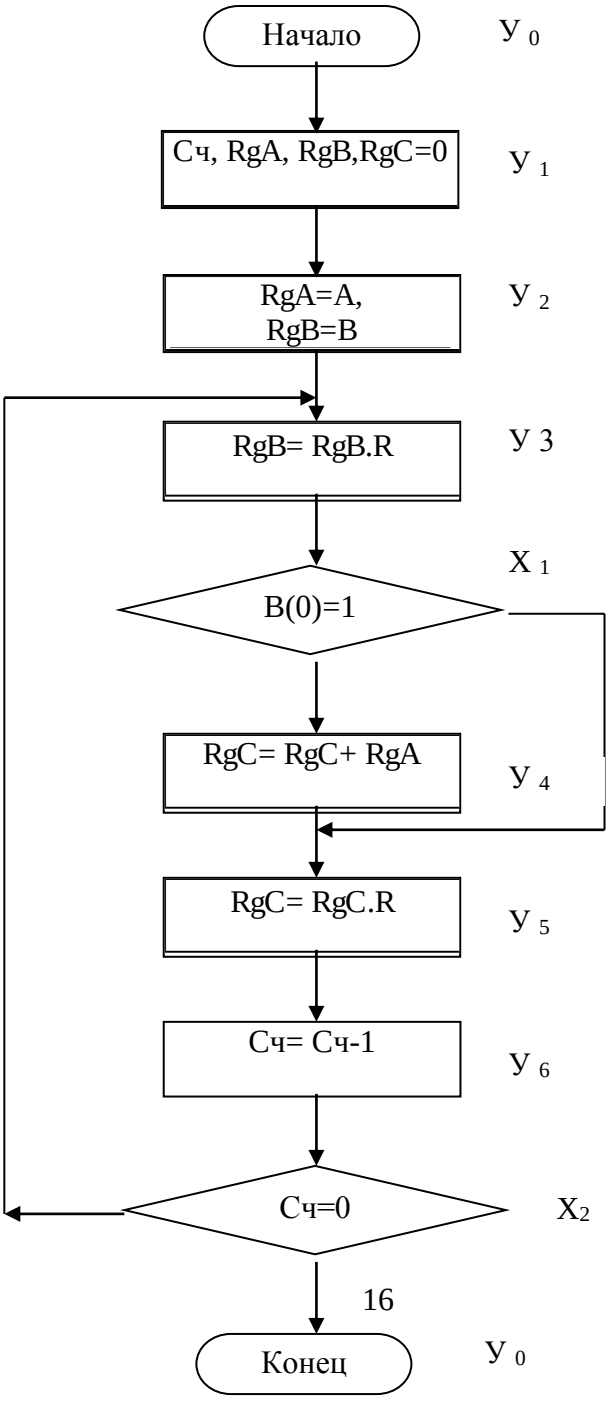


Рис.2. Граф-схема алгоритма

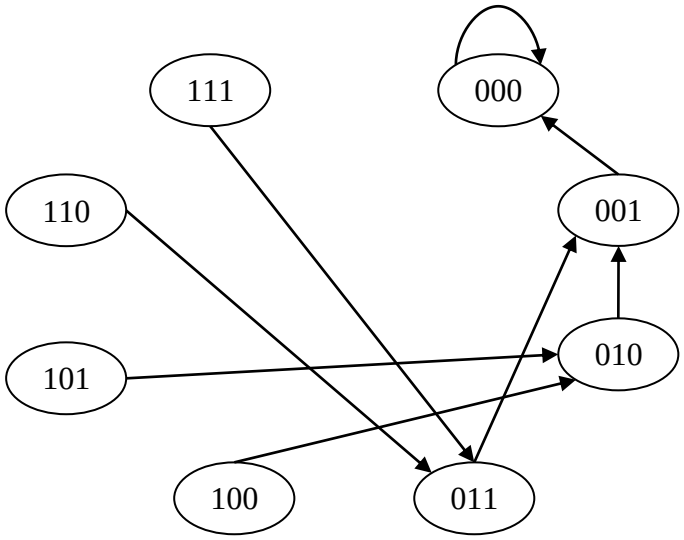
Пример синтеза регистра со сдвигом.
синтезировать логическую схему регистра;

- уяснить задачу;
- составить граф переходов;
- составить таблицу истинности;
- получить булевы функции;
- минимизировать булевы функции;
- взять заданный базис в виде заданных триггеров;
- составить логическую схему;
- произвести проверку.

Наименование регистра	Вид сдвига
D-триггер	Сдвиг вправо

Регистр – логический узел с памятью, состоящий из группы триггеров. Число триггеров соответствует количеству разрядов в слове, которое обрабатывается или запоминается в регистре. Рассмотрим 3-рязрядный регистр со сдвигом вправо. Он состоит из трёх D-триггеров, соединенных таким образом, что каждый тактовый импульс вызывает перемещение содержимого триггера в следующий за ним триггер.

Составляем граф переходов.



Граф переходов

Составляем таблицу истинности.

	x	$Q_2(t)$	$Q_1(t)$	$Q_0(t)$	$Q_2(t+1)$	$Q_1(t+1)$	$Q_0(t+1)$	D_2	D_1	D_0
8	1	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	0	0
10	1	0	1	0	0	0	1	0	0	1
11	1	0	1	1	0	0	1	0	0	1
12	1	1	0	0	0	1	0	0	1	0
13	1	1	0	1	0	1	0	0	1	0
14	1	1	1	0	0	1	1	0	1	1
15	1	1	1	1	0	1	1	0	1	1

По таблице истинности получим булевы функции

$$D_2 = 0$$

$$D_1 = XQ_2\overline{Q_1}\,\overline{Q_0} + XQ_2\overline{Q_1}Q_0 + XQ_2Q_1\overline{Q_0} + XQ_2Q_1Q_0$$

$$D_0 = X\overline{Q_2}Q_1\overline{Q_0} + X\overline{Q_2}Q_1Q_0 + XQ_2Q_1\overline{Q_0} + XQ_2Q_1Q_0$$

С помощью карт Карно минимизируем булевы функции

x

Q_2 \ Q_1				
	1_{12}	1_{14}	0_{6}	0_{4}
	1_{13}	1_{15}	0_{7}	0_{5}
	0_{9}	0_{11}	0_{3}	0_{1}
	0_{8}	0_{10}	0_{2}	0_{0}

Q_0

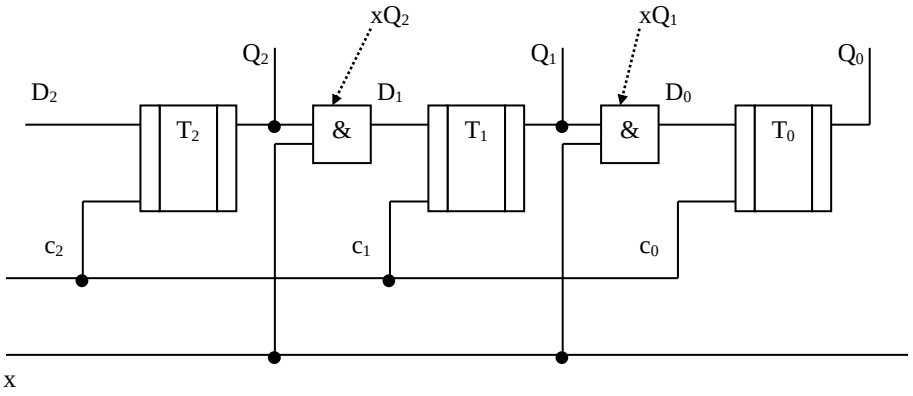
$D_1 = XQ_2$

Q_2 \ x, Q_1				
	0_{12}	1_{14}	0_{6}	0_{4}
	0_{13}	1_{15}	0_{7}	0_{5}
	0_{9}	1_{11}	0_{3}	0_{1}
	0_{8}	1_{10}	0_{2}	0_{0}

Q_0

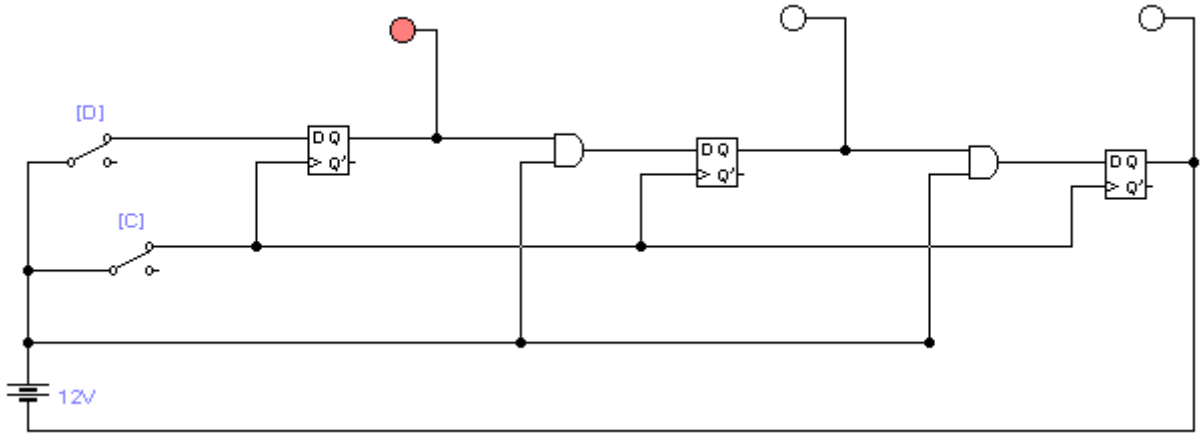
$D_0 = XQ_1$

Составим логическую схему



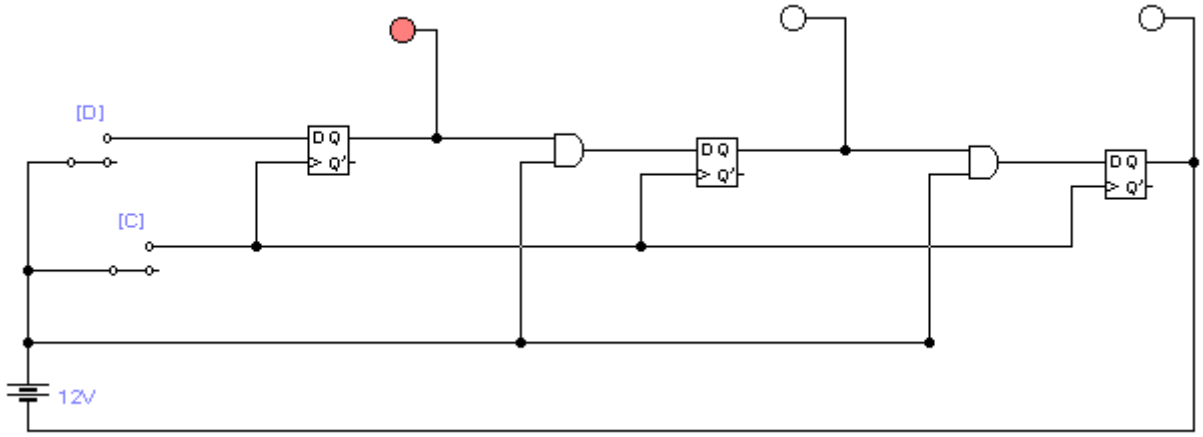
Реализуем логический узел в среде EWB и докажем его работоспособность записыванием и считыванием трехразрядных кодов.

Шаг 1. $Q_2=Q_1=Q_0$.Запишем в регистр трехразрядный код – 100, включив оба ключа D и C ($D_2=1$, $D_1=x\&Q_2=1\&0=0$, $D_0=x\&Q_1=1\&0=0$).



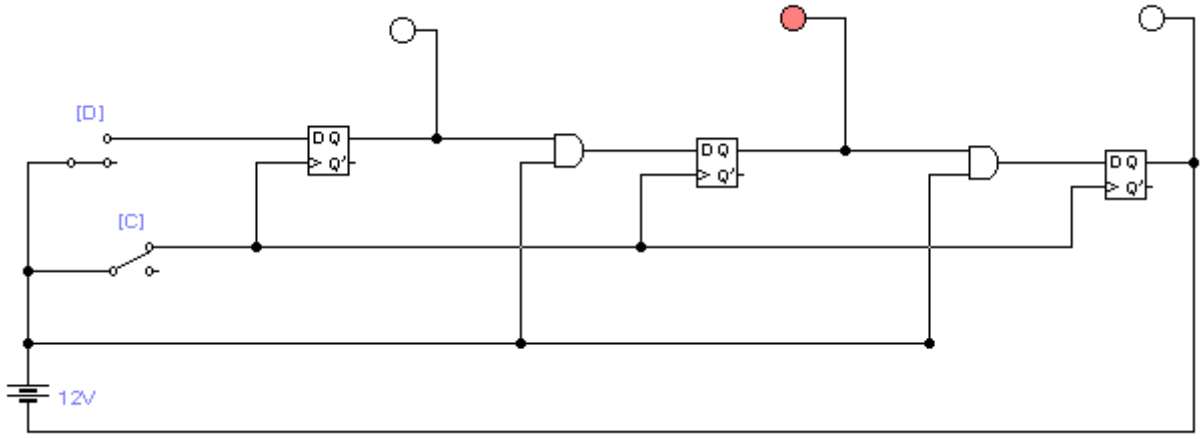
$Q_2(t+1)=1$, при $D_2=1$ и $C_2=1$; $Q_1(t+1)=0$, при $D_1=0$ и $C_1=1$; $Q_0(t+1)=0$, при $D_0=0$ и $C_0=1$.

Шаг 2. Выключим оба ключа (D и C).



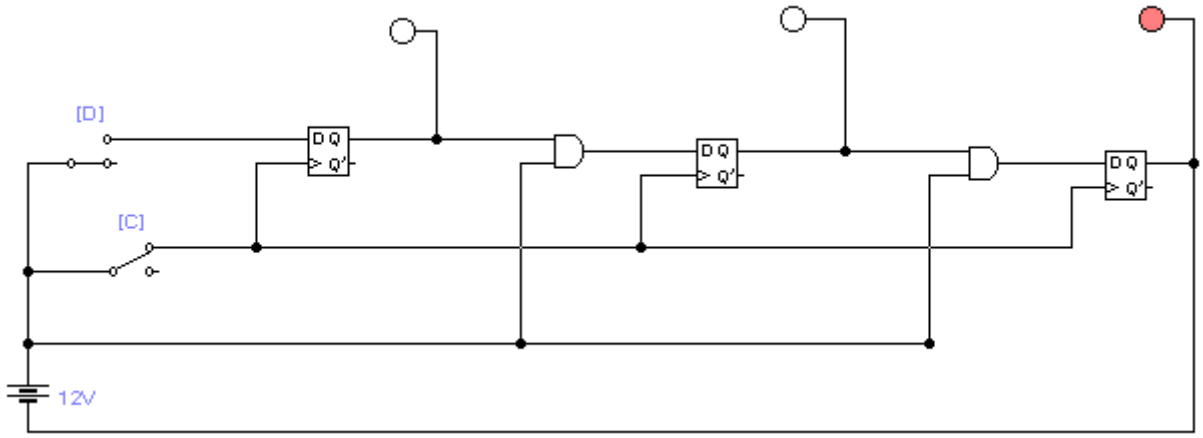
Произошла задержка сигнала, т.е. в регистре сохранился трехразрядный код – 100.

Шаг 3. Включим ключ C, что соответствует тактовому импульсу, вызывающему перемещение содержимого триггера в следующий за ним триггер, при выключенном ключе D ($D_2=0$, $D_1=x\&Q_2=1\&1=1$, $D_0=x\&Q_1=1\&0=0$).



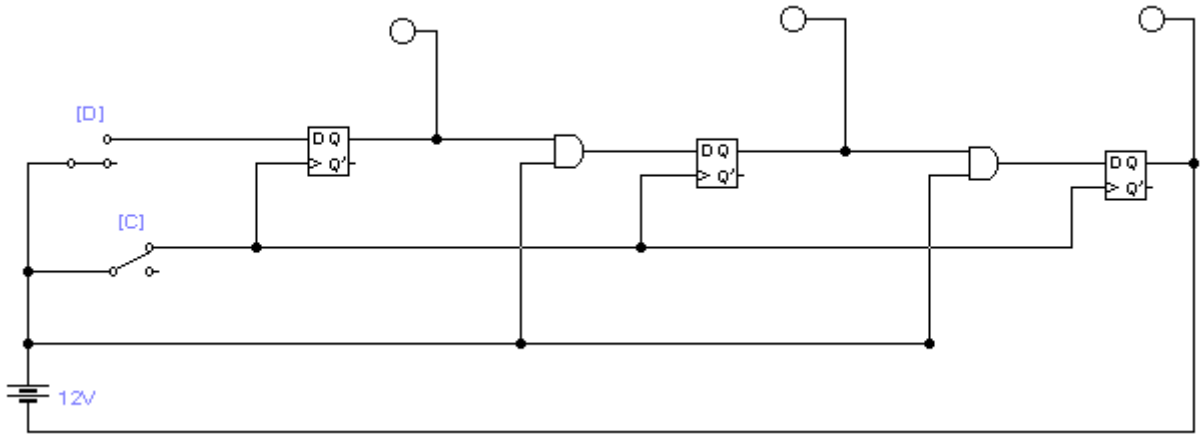
Получили трехразрядный код – 010. $Q_2(t+1)=0$, при $D_2=0$ и $C_2=1$; $Q_1(t+1)=1$, при $D_1=1$ и $C_1=1$; $Q_0(t+1)=0$, при $D_0=0$ и $C_0=1$.

Шаг 4. Выключим и снова включим ключ C, тем самым подав следующий тактовый импульс ($D_2=0$, $D_1=x\&Q_2=1\&0=0$, $D_0=x\&Q_1=1\&1=1$).



Получили трехразрядный код – 001. $Q_2(t+1)=0$, при $D_2=0$ и $C_2=1$; $Q_1(t+1)=0$, при $D_1=0$ и $C_1=1$; $Q_0(t+1)=1$, при $D_0=1$ и $C_0=1$.

Шаг 5. Повторим предыдущую операцию ($D_2=0$, $D_1=x\&Q_2=1\&0=0$, $D_0=x\&Q_1=1\&0=0$).



Получили трехразрядный код – 000. $Q_2(t+1)=0$, при $D_2=0$ и $C_2=1$; $Q_1(t+1)=0$, при $D_1=0$ и $C_1=1$; $Q_0(t+1)=0$, при $D_0=0$ и $C_0=1$.

В результате выполнения работы должна быть синтезирована логическая схема последовательного регистра со сдвигом вправо на D-триггерах. Функционирование синтезированной схемы подтверждает правило изменения трехразрядного кода занесенного в регистр, при подачи тактовых импульсов ($100 \rightarrow 010 \rightarrow 001 \rightarrow 000$), что соответствует построенному графу переходов.

Аналогичным образом необходимо синтезировать управляющий автомат.

Последовательность действий при синтезе УА

1. Уяснить задачу
2. Ознакомиться со способом выполнения арифметической операции

3. Составить ГСА
4. Разметить ГСА
5. Составить граф переходов
6. Выбрать триггер
7. Составить таблицу истинности
8. Получить булевы функции;
9. Минимизировать булевы функции;
10 Взять заданный базис в виде заданных триггеров;
11. Составить логическую схему;
12. Произвести проверку.

Содержание отчета:

4. Титульный лист
5. Цель работы
6. Ход работы

Триггер со сдвигом
Описание арифметической операции, согласно задания,
ГСА арифметической операции
Граф переходов
Схема управляющего автомата.
Результаты проверки

4. Ответы на вопросы

Вопросы к лабораторной работе

1. В чем отличие УА с жесткой логикой от УА с программируемой логикой.
2. В какой форме представляются сигналы из УА для управления в ОА.
3. Какую роль играет разметка в ГСА
4. Почему D- триггер предпочтительнее применять в УА с жесткой логикой.
5. Чем конечный автомат Мили отличается от автомата Мура.
6. Чем определяется количество триггеров в УА с жесткой логикой.

Лабораторная работа 3. Программирование ЭВМ в машинных кодах

Цель работы

Целью настоящей работы является изучение программирования компьютера IBM в машинных кодах с использованием возможностей системного отладчика DEBUG. Для выполнения работы необходимо предварительно ознакомиться с описанием DEBUG.

Основные понятия

Основой лабораторной работы является использование системной программы DEBUG, позволяющей вводить программы, осуществлять трассировку их выполнения, просматривать память. В этой работе изучается процесс ввода программы непосредственно в память в область сегмента кодов и объясняется каждый шаг ее выполнения. Изучаются команды отладчика DEBUG.

Для запуска этой программы наберите td.exe и нажмите Enter, в результате программа DEBUG должна загрузиться с диска в память. После окончания загрузки на экране появится главное окно, что свидетельствует о готовности программы DEBUG для приема команд.

1. Проверка размер доступной для работы памяти. В зависимости от модели компьютера это значение связано с установкой внутренних переключателей и может быть меньше, чем реально существует. Данное значение находится в ячейках памяти 413h и 414h и его можно просмотреть из DEBUG по адресу, состоящему из двух частей:
400 это адрес сегмента, который записывается как 40 (последний нуль подразумевается) и 13 это смещение от начала сегмента.

Таким образом, можно ввести следующий запрос:
D 40:13 (и нажать Enter).

Первые два байта, появившиеся в результате на экране, содержат объем памяти в килобайтах и в шестнадцатеричном представлении, причем байты располагаются в обратной последовательности.

Несколько следующих примеров показывают шестнадцатеричное обратное, шестнадцатеричное нормальное и десятичные представления.

Шестн. обратн.	Шестн. норм.	Десятичное
80 00	00 80	128
00 01	01 00	256
80 01	01 80	384
00 02	02 00	512
80 02	02 80	640

2. Серийный номер компьютера "зашит" в ROM по адресу FEOOOh. Для его выделения следует ввести:

D FEOO:0
(и нажать Enter)
В результате на экране появится семизначный номер компьютера и дата копирайт.
Дата ROM BIOS в формате мм/дд/гг (месяц,день,год) находится по адресу FFFF5h.

Введите

D FFFF:05 (и нажмите Enter)

Знание этой информации (даты) иногда бывает полезным для определения модели и возраста компьютера.

3. Установка адреса любой ячейки памяти для просмотра ее содержимого. Можно также пролистывать память, периодически нажимая клавишу D, DEBUG выведет на экран адреса, следующие за последней командой.
Для окончания работы и выхода из отладчика в DOS введите команду Q (Quit).
Для иллюстрации ввода программ в память, рассмотрим в качестве примера простую программу на машинном языке, ее представление в памяти и результаты ее выполнения. Программа показана в шестнадцатеричном формате:

Команда	Назначение
B82301	Переслать значение 0123h в AX
052500	Прибавить значение O025h к AX
8BD8	Переслать содержимое AX в BX
03D8	Прибавить содержимое AX к BX
88CB	Переслать содержимое BX в CX
2BC8	Вычесть содержимое AX из AX (очистка AX)
90	Нет операции
CB	Возврат в DOS.

Машинные команды имеют различную длину: один, два или три байта. Машинные команды находятся в памяти непосредственно друг за другом. Выполнение программы начинается с первой команды, и далее последовательно выполняются остальные.

Можно ввести эту программу непосредственно в память машины и выполнить ее покомандно. Одновременно на каждом шаге можно просматривать содержимое регистров после выполнения каждой команды.

Введем команду запуска отладчика DEBUG и нажмем клавишу Enter. После загрузки DEBUG на экране высвечивается приглашение к вводу команд в виде дефиса.

Для непосредственного ввода программы на машинном языке введите следующую команду, включая пробелы:

E CS:100 B8 23 01 05 25 00 (нажмите Enter)

Команда E обозначает Enter (ввод). Параметр CS:100 определяет адрес памяти, куда будут вводиться команды, шест. 100 (256) байт от начала сегмента кодов (обычный стартовый адрес для машинных кодов в отладчике DEBUG). Команда E записывает каждую пару шестнадцатеричных цифр в память в виде байта, начиная с адреса CS:100 до адреса CS:105.

Следующая команда Enter

E CS:106 8B D8 03 D8 8B CB (Enter)

вводит шесть байтов в ячейки, начиная с адреса CS:106 и далее в 107, 108, 109, 10A и 10B. Последняя команда Enter:

E CS:10C 2B C8 2B C0 90 CB (Enter)

вводит шесть байтов, начиная с CS:10C в 10D,10E, 10F, 110 и 111. Проверьте правильность ввода значений. Если имеются ошибки, то следует повторить команды, которые были введены неправильно. Для выполнения машинных команд используйте команду T. Введите команду R для просмотра содержимого регистров и флагов. В данный момент отладчик покажет содержимое регистров в шестнадцатеричном формате, например,

AX = 0000, BX = 0000, ...

Содержимое регистра IP (указатель команд) выводится в виде IP = 0100, показывая, что выполняемая команда находится на смещении 100h байт от начала сегмента кодов. (Вот почему использовалась команда E CS:100 для установки начала программы.) Регистр флагов показывает следующие значения флагов:

NV UP DI PL NZ NA PO NC.

Данные значения соответственно показывают: нет переполнения, правое направление, прерывания запрещены, знак плюс, не ноль, нет внешнего переноса, контроль на четность и нет переноса. В данный момент значение флагов несущественно. Команда R показывает также по смещению 0100h первую выполняемую машинную команду, которая выглядит следующим образом:

13C6:0100 B82301 MOV AX, 0123.

CS = 13C6 указывает, что начало сегмента кода находится по смещению 13C6 или, точнее, 13C60. Значение 13C6:01OO обозначает 100 (шестн.) байт от начального адреса 13C6 в регистре CS.

B8201 машинная команда, введенная по адресу CS:100. MOV AX,0123 ассемблерный мнемонический код соответствующий введенной машинной команде. Это есть результат операции дисассемблирования, которую обеспечивает отладчик для более

простого понимания машинных команд. Рассматриваемая в данном случае команда обозначает пересылку непосредственного значения 0123 в регистр AX.

В данный момент команда MOV еще не выполнена. Для ее выполнения нажмите клавишу T (для трассировки) и клавишу Enter . В результате команда MOV будет выполнена и отладчик выдаст на экран содержимое регистров, флаги, а также следующую на очереди команду. Заметим, что регистр AX теперь содержит значение 0123. Машинная команда пересылки в регистр AX имеет код B8, за которым следуют непосредственные данные 2301. В ходе выполнения команда B8 пересылает значение 23 в младшую часть регистра AX, в AL, а значение 01 в старшую часть регистра AX, т. е. в регистр AH.

Содержимое регистра IP:0103 показывает адрес следующей выполняемой команды в сегменте кодов:

13C6:0103 052500 ADD AX,0025

Для выполнения данной команды снова введите T. Команда прибавит 25H к младшей (AL) части регистра AX и 00 к старшей (AH) части регистра AX, т.е прибавит 0025 к регистру AX. Теперь регистр AX содержит 0148, а регистр IP 0106 адрес следующей команды для выполнения. Введите снова команду T. Следующая машинная команда пересылает содержимое регистра AX в регистр BX, и после ее выполнения в регистре BX будет содержаться значение 0148. Регистр AX сохраняет прежнее значение 0148, поскольку команда MOV только копирует данные из одного места в другое. Теперь вводите команду T для пошагового выполнения каждой оставшейся в программе команды. Следующая команда прибавит содержимое регистра AX к содержимому регистра BX, в последнем получим 0290. Затем программа скопирует содержимое регистра BX в CX, вычтет AX из CX и вычтет AX из него самого. После этой последней команды флаг нуля изменит свое состояние с NZ (не нуль) на ZR (нуль), так как результатом этой команды является нуль (вычитание AX из самого себя обнуляет этот регистр).

Можно ввести T для выполнения последних команд NOP и RET, но это будет сделано позже. Для просмотра программы в машинных кодах в сегменте кодов введите D:

D CS:100

В результате отладчик выдаст на каждую строку экрана по 16 байт данных в шестнадцатиричном представлении (32 шестнадцатиричные цифры) и в символьном представлении в коде ASCII (один символ на каждую пару шестнадцатиричных цифр). Представление машинного кода в символах ASCII не имеет смысла и может быть игнорировано.

Первая строка дампа начинается с 00 и представляет содержимое ячеек от CS:100 до CS:10F. Вторая строка представляет содержимое ячеек от CS:110 до CS:11F. Несмотря на то, что ваша программа заканчивается по адресу CS:111, команда Dump автоматически выдаст на восьми строках экрана дамп с адреса CS:100 до адреса

CS:170.
При необходимости повторить выполнение этих команд сбросьте содержимое регистра IP и повторите трассировку снова. Введите R IP, введите 100, а затем необходимое число команд T. После каждой команды нажимайте клавишу Enter.

Для завершения работы с программой DEBUG введите Q (Quit выход). В результате произойдет возврат в DOS.

4. В предыдущем примере использовались непосредственные данные, описанные непосредственно в первых двух командах (MOV и ADD). Рассмотрим аналогичный пример, в котором значения 0123 и 0025 определены в двух полях сегмента данных. Следующий пример показывает, как компьютер обеспечивает доступ к данным посредством регистра DS и адресного смещения.
В этом примере определены области данных, содержащие соответственно следующие значения:

Адрес в DS	Шестн. значения	Номера байтов
0000	2301	0 и 1
0002	2500	2 и 3
0004	0000	4 и 5
0006	2A2A2A	6, 7 и 8

Так как, шестнадцатеричный символ занимает половину байта, то 23 находится в байте 0 (в первом байте) сегмента данных, 01 в байте 1 (во втором байте). Ниже показаны команды машинного языка, которые обрабатывают эти данные:

Команда	Назначение
A10000	Переслать слово (два байта), начинающееся в DS по адресу 0000, в регистр AX.
03060200	Прибавить содержимое слова (двух байт), начинающегося в DS по адресу 0002, к регистру AX.
A30400	Переслать содержимое регистра AX в слово, начинающееся в DS по адресу 0004.
CB	Вернуться в DOS.

Обратите внимание, что здесь имеются две команды MOV с различными машинными кодами: A1 и A3. Фактически машинный код зависит от регистров, на которые имеется ссылка, количества байтов (байт или слово), направления передачи данных (из регистра или в регистр) и от ссылки на непосредственные данные или на память.

Воспользуемся опять отладчиком DEBUG для ввода данной программы и анализа ее выполнения. Когда отладчик выдал свое дефисное приглашение, он готов к приему команд. Сначала введите команды E (Enter) для сегмента данных:

E DS: 00 23 01 25 00 00 00 (Нажмите Enter)

E DS: 06 2A 2A 2A (Нажмите Enter)

Первая команда записывает три слова (шесть байтов) в начало сегмента данных DS:00. Заметьте, что каждое слово вводилось в обратной последовательности, так что 0123 есть 2301, а 0025 есть 2500. Когда команда MOV будет обращаться к этим словам, нормальная последовательность будет восстановлена и 2301 станет 0123, а 2500 0025.

Вторая команда записывает три звездочки (***) для того, чтобы их можно было увидеть впоследствии по команде D (Dump) другого назначения эти звездочки не имеют.

Введем теперь команды в сегмент кодов, опять начиная с адреса CS:100:

E CS:100 A1 00 00 03 06 02 00
E CS:107 A3 04 00 CB

Эти команды теперь находятся в ячейках памяти от CS:100 до CS:10A. Команды можно выполнить так же, как это делалось ранее.

Для просмотра информации в сегменте данных и в сегменте кодов введите команды D (Dump) соответственно

1. для сегмента данных: D DS:0000 (Enter)
2. для сегмента кодов: D CS:100 (Enter)

Теперь введите R для просмотра содержимого регистров и флагов и для отображения первой команды. Регистры содержат те же значения, что и в первом примере. Команда отобразится в виде 13C6:0100 A10000 MOV AX,[0000] Так как регистр CS содержит 13C6, то CS:100 содержит первую команду A10000. Отладчик интерпретирует эту команду как MOV и определяет ссылку к первому адресу [0000] в сегменте данных. Квадратные скобки необходимы для указания ссылки к адресу памяти, а не к непосредственным данным. Если бы квадратных скобок не было, то команда MOV AX,0000 обнулила бы регистр AX непосредственным значением 0000.

Теперь введем команду T. Команда MOV AX, [0000] перешлет содержимое слова, находящегося по нулевому смещению в сегменте данных, в регистр AX. Содержимое 2301 преобразуется командой в 0123 и помещается в регистр AX. Следующую команду ADD можно выполнить введением еще раз команды T. В результате содержимое слова в DS по смещению 002 прибавится в регистр AX. Теперь регистр AX будет содержать сумму 0123 и 0025, т.е. 0148.

Следующая команда MOV [0004],AX выполняется опять по вводу T. Эта команда пересылает содержимое регистра AX в слово по смещению 0004. Для просмотра изменений содержимого сегмента данных введите D DS:00. Первые девять байт будут следующими:

Значение в сегменте данных: 23 01 25 00 48 01 2A 2A 2A

Величина смещения: 00 01 02 03 04 05 06 07 08

Значение 0148, которое было занесено из регистра AX в сегмент данных по смещению 04 и 05, имеет обратное представление 4801. Заметьте, что эти шестнадцатиричные значения представлены в правой части экрана их символами в ASCII-коде. Например, 23h генерируется в символ #, а 25h в символ %. Три байта с значениями 2Ah - высвечиваются в виде трех звездочек (***). Левая часть дампа показывает действительные машинные коды, которые находятся в памяти. Правая часть дампа только помогает проще локализовать символьные (строчные) данные.

Для просмотра содержимого сегмента кодов введите D DS:100.

Для завершения работы с программой введите Q.

5. Для доступа к машинной команде процессор определяет ее адрес из содержимого регистра CS плюс смещение в регистре IP. Например, пусть регистр CS содержит 04AFh (действительный адрес 04AF0), а регистр IP содержит 023h: CS: 04AF0 IP: 0023 Адрес команды: 04B13 Если, например, по адресу 04B13 находится команда: A11200 MOV AX,[0012], то в памяти по адресу 04B13 содержится первый байт команды. Процессор получает доступ к этому байту и по коду команды (A1) определяет длину команды три байта.

Для доступа к данным по смещению [0012] процессор определяет адрес, исходя из содержимого регистра DS (как правило) плюс смещение в операнде команды. Если DS содержит 04B1h (реальный адрес 04B10), то результирующий адрес данных определяется следующим образом: DS: 04B10 Смещение: 0012 Адрес данных: 04B22. Предположим, что по адресам 04B22 и 04B23 содержатся следующие данные: Содержимое: 24 01 Адрес: 04B22 04B23 Процессор выбирает значение 24 из ячейки по адресу 04B22 и помещает его в регистр AL, а значение 01 по адресу 04B23 в регистр AH. Регистр AX будет содержать в результате 0124. Во время выборки каждого байта команды процессор увеличивает значение регистра IP на единицу, так что к началу выполнения следующей команды в нашем примере IP будет содержать смещение 0026. Таким образом, процессор теперь готов для выполнения следующей команды, которую он получает по адресу из регистра CS [04AF0] плюс текущее смещение в регистре IP (0026), т.е. 04B16.

Процессоры 8086, 80286 и 80386 действуют более эффективно, если в программе обеспечивается доступ к словам, расположенным по четным адресам. В предыдущем примере процессор мог сделать выборку слова по адресу 4B22 для загрузки его

непосредственно в регистр за одно обращение к памяти. Но если слово начинается на нечетном адресе, процессор производит два обращения к памяти.

Пусть команда должна выполнить выборку слова, начинающегося по адресу 04B23, и загрузить его в регистр AX: Содержимое памяти: |xx| 24 | 01 | xx | Адрес: 04B23 04B24 04B25 Сначала процессор получает доступ к байтам по адресам 4B22 и 4B23 и пересылает байт из ячейки 4B23 в регистр AL. Затем он получает доступ к байтам по адресам 4B24 и 4B25 и пересылает байт из ячейки 4B23 в регистр AH. В результате регистр AX будет содержать 0124.

Команды обращения к памяти меняют слово при загрузке его в регистр так, что получается правильная последовательность байтов, и, если программа имеет частый доступ к памяти, то для повышения эффективности можно определить данные так, чтобы они начинались по четным адресам.

Например, поскольку начало сегмента должно всегда находиться по четному адресу, первое поле данных начинается также по четному адресу, и пока следующие поля определены как слова, имеющие четную длину, они все будут начинаться на четных адресах.

Ассемблер имеет директиву EVEN, которая вызывает выравнивание данных и команд на четные адреса памяти.

3. Порядок выполнения работы

В первом упражнении в этой лабораторной работе проводилась проверка объема памяти RAM, которую имеет компьютер. Базовая система ввода-вывода (BIOS) имеет в ROM подпрограмму, которая определяет размер памяти. Можно обратиться в BIOS по команде INT, в данном случае по прерыванию 12h. В результате BIOS возвращает в регистр AX размер памяти в килобайтах. Загрузите в память отладчик DEBUG и введите для команд INT 12h и RET следующие машинные коды:

E CS: 100 CD 12 CB

Нажмите R (и Enter)

для отображения содержимого регистров и первой команды. Регистр IP содержит 0100, при этом высвечивается команда INT 12h. Теперь нажмите T (и Enter) несколько раз и просмотрите выполняемые команды BIOS (отладчик показывает мнемокоды, хотя в действительности выполняются машинные коды):
STI

PUSH DS

MOV AX,0040

MOV DS,AX

MOV AX,[0013]

POP DS

IRET

В этот момент регистр AX содержит размер памяти в шестнадцатеричном формате. Теперь введите еще раз команду T для выхода из BIOS и возврата в вашу программу. На экране появится команда RET для машинного кода CB, который был введен вами.

В операционной системе DOS версии 2.0 и старше можно использовать программу DEBUG для ввода команд ассемблера так же, как и команд машинного языка. На практике можно пользоваться обоими режимами.

Команда отладчика A (Assemble) переводит DEBUG в режим приема команд ассемблера и преобразования их в машинные коды. Установим начальный адрес следующим образом:

A 100 [Enter]

Отладчик выдаст значение адреса сегмента кодов и смещения в виде xxxx:0100. Теперь можно вводить каждую команду, завершая клавишей Enter. Когда вся программа будет введена, нажмите снова клавишу Enter для выхода из режима ассемблера. Введите следующую программу:

MOV AL,25 [Enter]

MOV BL,32 [Enter]

ADD AL,BL [Enter]

RET [Enter]

По завершении на экране дисплея будет следующая информация:

xxxx:0100 MOV AL,25

xxxx:0102 MOV BL,32

xxxx:0104 ADD AL,BL

xxxx:0106 RET

В этот момент отладчик готов к приему следующей команды. При нажатии Enter операция ввода будет прекращена.

Можно видеть, что отладчик определил стартовые адреса каждой команды. Прежде чем выполнить программу, проверим сгенерированные машинные коды.

Команда отладчика U (Unassemble) показывает машинные коды для команд ассемблера. Необходимо сообщить отладчику адреса первой и последней команд, которые необходимо просмотреть, в данном случае 100 и 106. Введите:

U 100,106 [Enter]

На экране дисплея появится

xxxx:0100 B025 MOV AL,25

xxxx:0102 B332 MOV BL,32

xxxx:0104 00D8 ADD AL,BL

xxxx:0106 C3 RET

Проведем трассировку выполнения программы, начиная с команды R, для вывода содержимого регистров и первой команды программы. С помощью команд T выполним последовательно все команды программы.

Ввод на языке ассемблера обычно используется, когда машинный код неизвестен, а ввод в машинном коде для изменения программы во время выполнения. Однако в действительности программа DEBUG предназначена для отладки программ, и в последующем, основное внимание будет уделено использованию языка ассемблера.

Можно использовать DEBUG для сохранения программ на диске в следующих случаях:

а) после загрузки программы в память машины и ее модификации необходимо сохранить измененный вариант. Для этого следует:
-загрузить программу по ее имени:
DEBUG имя файла [Enter],
просмотреть программу с помощью команды D и
ввести изменения по команде E,
записать измененную программу:
W [Enter];

б) необходимо с помощью DEBUG написать небольшую по объему программу и сохранить ее на диске. Для этого следует:
вызвать отладчик DEBUG,
с помощью команд A (assemble) и E (Enter) написать программу, присвоить программе имя:
N имя файла.COM [Enter].
Тип программы должен быть COM;
указать отладчику длину программы в байтах.
В последнем примере концом программы является команда
xxxx:0106 C3 RET
Эта команда однобайтовая, и поэтому размер программы бу
дет равен 107 (конец) минус 100 (начало), т.е. 7.

в) запросить регистр CX командой R CX [Enter], отладчик
выдаст на этот запрос CX:0000 (нулевое значение),
указать длину программы 7,
записать измененную программу: W [Enter].

В обоих случаях DEBUG выдает сообщение "Writing nn bytes". Если nn равно 0, то произошла ошибка при вводе длины программы, и необходимо повторить запись снова.

Отладчик DOS DEBUG это достаточное мощное средство, полезное для отладки ассемблерных программ. Однако следует быть осторожным при ее использовании, особенно для команды E (Enter).

Ввод данных в неправильные адреса памяти или ввод некорректных данных могут привести к непредсказуемым результатам. На экране в этом случае могут появиться "странные" символы, клавиатура заблокирована или даже DOS прервет DEBUG и перезагрузит себя с диска. Какие-либо серьезные повреждения вряд ли произойдут, но возможны некоторые неожиданности, а также потеря данных, которые вводились при работе с отладчиком.

Если данные, введенные в сегмент данных или сегмент кодов, оказались некорректными, следует, вновь используя команду E, исправить их. Однако, можно не заметить ошибки и начать трассировку программы. Но и здесь возможно еще использовать команду E для изменений. Если необходимо начать выполнение с первой команды, то следует установить в регистре командного указателя (IP) значение 0100. Введите команду R (Register) и требуемый регистр в следующем виде:

R IP [Enter]

Отладчик выдаст на экран содержимое регистра IP и перейдет в ожидание ввода. Здесь следует ввести значение 0100 и нажать для проверки результата команду R (без IP). Отладчик выдаст содержимое регистров, флагов и первую выполняемую команду. Теперь можно, используя команду T, вновь выполнить трассировку программы.

Если ваша программа выполняет какие-либо подсчеты, то, возможно, потребуется очистка некоторых областей памяти и регистров. Но убедитесь в сохранении содержимого регистров CS, DS, SP и SS, которые имеют специфическое назначение.

4. Требования к отчету

Отчет по лабораторной работе должен содержать:

1.Цель работы
2.Условие задания;
3.текст программ на языке Ассемблера;
4.ответы на контрольные вопросы.

5. Задание для выполнения лабораторной работы

1. Напишите машинные команды для:

а) пересылки значения 4629h в регистр AX;
б) сложения 036Ah с содержимым регистра AX.

2. Предположим, что была введена следующая E-команда:
E CS:100 B8 45 01 05 25 00
Вместо шестьнацатиричных . значения 45 предполагалось 54. Напишите команду E для корректировки только одного неправильно введенного байта, т.е. непосредственно замените 45 на 54.

3. Введена следующая команда:
E CS:100 B8 04 30 05 00 30 CB
Ответьте на вопросы

а) Что представляют собой эти команды?
б) После выполнения этой программы в регистре AX должно быть значение 0460, но в действительности оказалось 6004. В чем ошибка и как ее исправить?
в) После исправления команд необходимо снова выполнить программу с первой команды. Какие две команды отладчика потребуются ?

4. В машинных кодах имеется программа:
B0 25 D0 E0 B3 15 F6 E3 CB
Программа выполняет следующее:

- пересылает значение 25h в регистр AL;
- сдвигает содержимое регистра AL на один бит влево (в результате в AL будет 4A);
- пересылает значение 15h в регистр BL;
- умножает содержимое регистра AL на содержимое регистра BL.

Используйте отладчик для ввода (E) этой программы по адресу CS:100. Не забывайте, что все значения представлены в шестнадцатеричном виде. После ввода программы наберите D CS:100 для просмотра сегмента кода. Затем введите команду R и необходимое числа команд T для пошагового выполнения программы до команды RET. Какое значение будет в регистре AX в результате выполнения программы?

5. Используйте отладчик для ввода (E) следующей программы в машинных кодах
Данные: 25 15 00 00 код:

A0 00 00 D0 E0 F6 26 01 00 A3 02 00 CB
Программа выполняет следующее:

- пересылает содержимое одного байта по адресу DS:00 (25) в ре гистр AL;
- сдвигает содержимое регистра AL влево на один бит (получая в результате 4A);
- умножает AL на содержимое одного байта по адресу DS:01 (15);
- пересылает результат из AX в слово, начинающееся по адресу DS:02.

После ввода программы используйте команды D для просмотра сегмента данных и сегмента кода. Затем введите команду R и необходимое число команд T для достижения конца программы (RET). В этот момент регистр AX должен содержать результат 0612. Еще раз используйте команду D DS:00 и обратите внимание на то, что по адресу DS:02 значение записано как 1206.

6. Для предыдущего задания (5) постройте команды для записи программы на диск под именем TRIAL.COM.

7. Используя команду A отладчика, введите следующую программу:

```
MOV BX,25
ADD BX,30
SHL BX,01
SUB BX,22
NOP
RET
```

Сделайте ассемблирование и трассировку выполнения этой программы до команды NOP. Каков результат выполнения этой программы и где он находится? Объясните смысл каждой команды и напишите эту программу в машинных кодах. С помощью отладчика DEBUG выполните зти машинные коды с начального адреса CS:160 и сравните полученные результаты программы в машинных кодах. С помощью отладчика DEBUG выполните зти машинные коды с начального адреса CS:160 и сравните полученные результаты.

Контрольные вопросы

1. В чем назначение отладчика и отличие debugger от turbo debugger
2. В чем суть сегментации памяти.
3. Какие регистры являются регистрами общего назначения.
4. Что такое реальная адресация и виртуалmная память.
5. Какие модели распределения памяти существуют.
6. Какую роль выполняет транслятор.

Лабораторная работа 4. Таймер.

Цель работы

Изчение работы формирователя временных интервалов – таймера. Разработка программы синтеза музыкальных звуков с использоватенм таймера.

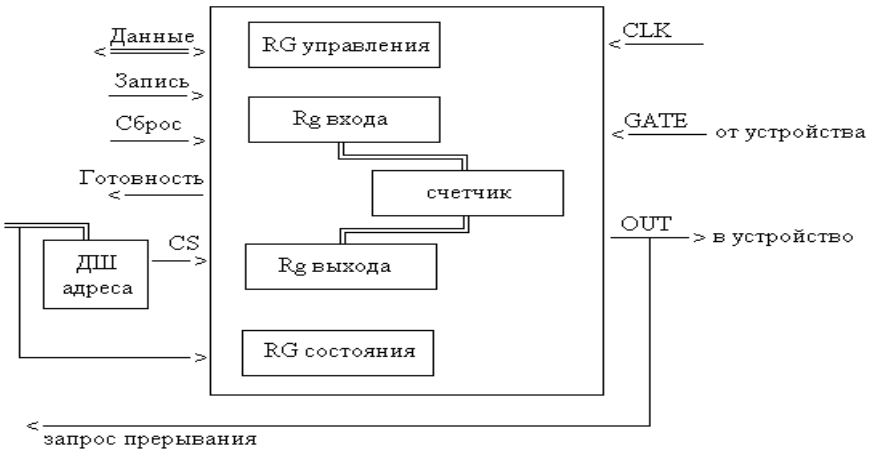
Общие сведения

Для эффективной работы систем преобразования информации необходимо применять различные устройства формирования временных интервалов. Такие устройства называются программируемыми интервальными таймерами или системными таймерами. К процедурам применения таймера можно отнести следующие:

- контроль за работой двигателя НГНД;
- прерывание функционирования операционной системы для переключения программ;
- вывод точных временных интервалов с программными периодами в устройство ввода-вывода;
- измерение временной задержки между событиями;
- подсчет числа появления событий во внешнем эксперименте;
- формирование случайных чисел;
- прерывание работы процессора после появления запрограммированного числа каких-либо событий.

Обычно системный таймер представляет собой устройство с возможностью программирования. Структурная схема широко распространенного системного таймера приведена на рис.1.

Рис.1. Структурная схема системного таймера



Регистры управления и входа - это входные порты, а регистры состояния и выходной - это выходные порты. Счетчик программно процессору не доступен и запускается от входного регистра с начального значения "0". Выход OUT подключается к линии запроса прерывания в системной шине, поэтому прерывание возникает при достижении счетчиком "0".

Устройство вводит во входной регистр значение начального счета, передает его в счетчик и выполняет счет "назад" (вычитание) импульсами с входа CLK. Текущее содержимое можно ввести в процессор немедленно, не нарушая его работы. Индикация нуля в счетчике обычно фиксируется на выходе OUT и в одном бите регистра состояния. Регистр управления определяет режим работы и выполняет другие функции. Возможные действия: вход GATE для разрешения и запрещения входа CLK; вход GATE вызывает реинициализацию входа CLK; вход GATE прекращает счет и формирует высокий уровень на OUT.

На рис.2. показан пример обработки прерывания таймера для контроля работы двигателя НГНД.

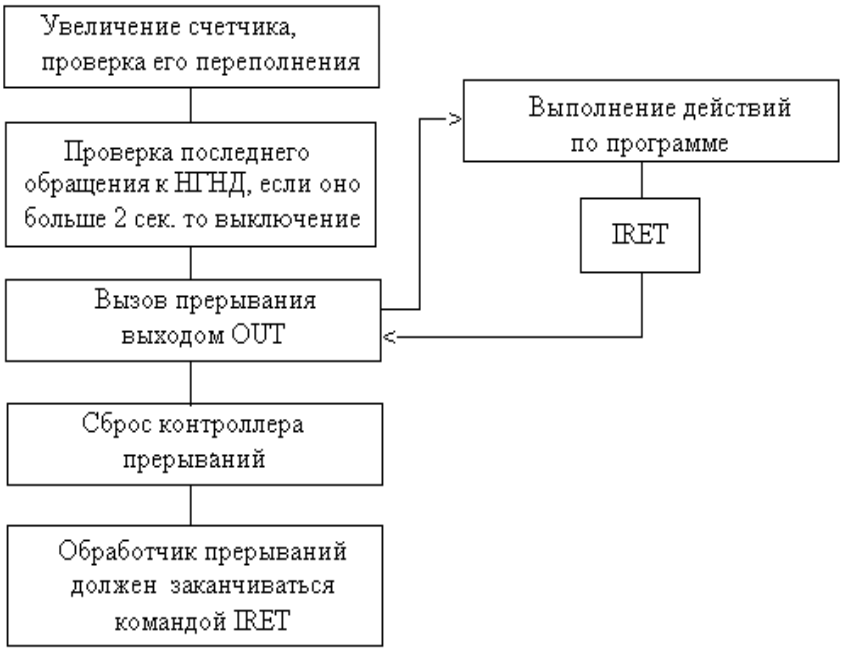


Рис.2. Обработка прерывания таймера для контроля работы двигателя НГНД

Системный таймер обычно реализуется на микросхеме Intel 8253 (IBM PC/XT) и 8254 (IBM AT и PS/2). Отечественные аналоги К1810ВИ53 и К1810ВИ54.

Программируемый системный таймер

Эти таймеры состоят из трех независимых каналов или счетчиков. Каждый канал содержит регистры: состояния канала RS (8 разрядов); управляющего скоба RSW (8 разрядов); буферный регистр OL (16 разрядов); регистр констант пересчета CR (14 разрядов). Каналы таймера подключаются к внешним устройствам при помощи трех линий: GATE - управляющий вход; CLOCK - вход тактовой частоты; UT - выход таймера. Регистр счетчика работает в режиме вычитания. Его содержимое уменьшается по заднему фронту CLOCK, при условии, что на входе GATE установлена "1".

Упрощенная схема работы представлена на рис. 3.

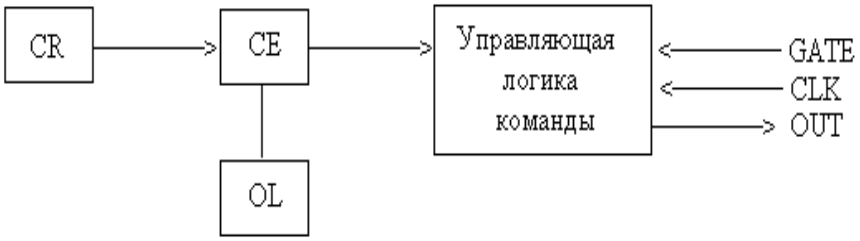


Рис. 1.3 Схема работы системного таймера

Обращение к регистру производится в соответствии с таблицей 1.

Таблица 1

CS	RO	WL	A1	A1	Передача
0	1	0	0	0	В счетчик 0 СК
0	1	0	0	1	В счетчик 1СК
0	1	0	1	0	В счетчик 1 СК
0	1	0	1	1	В регистр управления
0	0	1	0	0	Из счетчика 0 OL
0	0	1	0	1	Из счетчика 0 OL
0	0	1	1	0	Из счетчика 0 OL

Все остальные комбинации вызывают нулевой режим.

Возможны шесть режимов работы таймера. Они разделяются на три типа:

режимы 0,4 - однократное выполнение функций;

режимы 1,5 - работа с перезапуском;

режимы 2,3 - работа с автозагрузкой.

Для работы необходимо на вход подать сигнал GATE=1, тогда содержимое регистра констант CR переписывается в счетчик CE и по сигналу CLOCK содержимое регистра CE начинает уменьшаться. Процесс счета можно приостановить, если GATE=0. Для повторения выполнения функции необходима новая загрузка CR. При работе на канале с перезапуском для продолжения работы не требуется перепрограммирование. В режиме автозагрузки содержимое регистра CR автоматически переписывается в регистр CE после завершения счета. Сигнал на выходе OUT появляется только при GATE=1. В компьютере IBM PC/XT/AT/PS2 задействованы все три канала: канал 0 - используется в системных часах времени суток; канал 1 - используется для регенерации содержимого динамической памяти компьютера, OUT к каналу ПДП, выполняющего регенерацию содержимого памяти; канал 2 - подключен к громкоговорителю компьютера (при этом используется режим 3).

Программирование таймера на уровне портов

Таймеру соответствует четыре порта ввода/вывода со следующими адресами:
40h - канал 0;
41h - канал 1;
42h - канал 2;
43h - управляющий регистр.
Формат управляющего регистра имеет вид, показанный на рисунке 4.

Поле M определяет режимы микросхемы 8254.
0 - прерывание от таймера;
1 - программируемый ждущий мультивибратор;
2 - программируемый генератор импульсов;
3 - генератор меандра;
4 - программно-запускаемый одновибратор;
5 - аппаратно-запускаемый одновибратор.

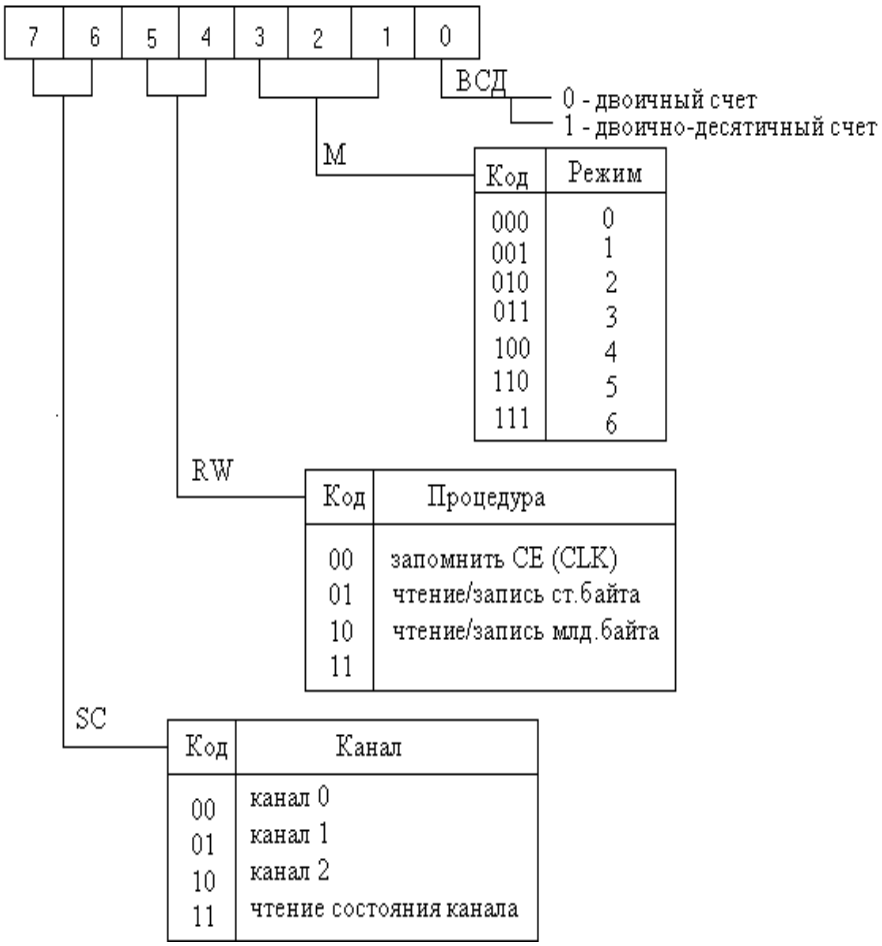


Рис. 4. Формат управляющего регистра.

Наиболее часто используется режим 3. Поле SC определяет номер канала, для которого предназначено управляющее слово. Если в этом поле задано значение 11, будет выполняться чтение состояния канала.

Формат команды RBC чтения состояния канала представлен на рис.5.

Формат слова состояния канала совпадает с форматом управляющего регистра, кроме 6 и 7 бита. Разряд FN - в основном используется в режимах 1 и 5; разряд OUT - позволяет определить состояние входной линии в момент выполнения команды RBC.
Для программирования канала таймера необходимо выполнить следующие действия: вывести в порт управляющего регистра с адресом 43h управляющее слово; требуемое значение счетчика посылается в порт канала (адрес 40 h - 42h).

Для воспроизведения звуков используется канал 2, связанный с громкоговорителем. Младший бит порта 61h подключен к входу GATE канала 2 таймера и при установке в 1 разрешает генерацию импульсов для громкоговорителя. Также первый бит порта 61h должен быть установлен в 1.

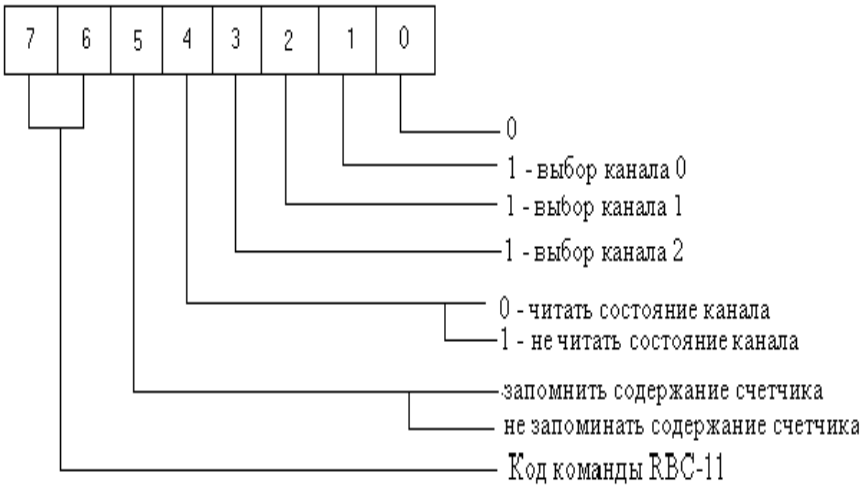


Рис.5 Формат команды RBC чтения состояния канала

Для включения звука нужно выполнить: загрузить регистр счетчика необходимыми значениями частоты; для включения звука установить в 1 два младших бита порта 61h, а для выключения их необходимо сбросить.
Остальные 6 битов порта 61h не должны быть изменены, поэтому установку младших битов нужно производить в рабочей ячейке.

Одноголосая мелодия состоит из нот, разделенных или не разделенных паузами. При воспроизведении мелодии необходимо для каждой ноты запрограммировать соответствующим образом канал 2 таймера и включить громкоговоритель (порт 61h) на определенное время равное длительности ноты. Затем выключить громкоговоритель и выдержать паузу.

Ход работы

Для выполнения работы рекомендуется придерживаться следующей последовательности:

- ♦ ознакомиться с описанием таймера;
- ♦ изучить принцип работы и процедуры формирования звука;
- ♦ составить значения частот и длительностей;
- ♦ написать программу и отладить ее;
- ♦ сформировать требуемую мелодию.

Для определения значения, которое должно быть записано в регистр счетчика канала 2 надо разделить 1193180 на требуемую частоту в герцах.

Примерная структура алгоритма приведена на рис. 6.

Для подготовки частот звуков используйте следующую таблицу:

до	до	ре	ре	ми	фа	фа	соль
261,7	277,2	293,7	311,1	329,6	349,2	370	392
соль	ля	ля	си				
415,3	440	466,2	493,9				

Варианты заданий
Воспроизвести "мелодию" согласно индивидуальному заданию.
Для этого необходимо:

- программировать канал 2 таймера в режиме 3 на нужную частоту (определяемую нотой, см. табл. нот);
- с помощью порта 61H задавать требуемую длительность звучания ноты и длительность паузы (длительность ноты в секундах задана в индивидуальном задании после наименования ноты).

1. До(1), ре(2), ми(3).
2. Си(3), ля(2), соль(1).
3. До-диез(5), соль(1), ре(3).
4. Ми(2), до(3), ля-диез(1)
5. Фа(3), соль-диез(1), фа(3).
6. Фа-диез(2), ре(4), си(2).
7. Соль(1), ля-диез(1), ля(2).
8. Ля(3), до-диез(1), ля-диез(3).
9. Соль-диез(2), си(3), ми(1).
10. Ля-диез(1,5), фа(1), до(2).
11. Си(0,5), до-диез(3), ля-диез(0,5).
12. До(2,5), фа(1), соль(2).
13. Фа(2), фа-диез(2), соль(2).
14. Ре-диез(2,5), ре(2,5), до-диез(2,5).

15. Ми(1), ре(2), до(1).
16. Ре(1,5), ля-диез(2), ре-диез(1).
17. Соль(2), ля(1), до-диез(0,5).

18. Ля(4), ре-диез(1), си(0,5).
19. Ре(3), ми(0,5), фа(0,9).
20. Си(0,5), до(1,3), ля(1,5).

Рис. 6. Структура алгоритма

СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы.
2. Ход работы.
3. Ответы на вопросы.
4. Листинг программы
5. Заключение.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Приведите пример использования таймера в компьютере.
2. Сколько каналов имеет системный таймер 8254?
3. Какой канал используется для системных часов, а какой подключен к громкоговорителю?
4. Как много режимов используется в таймере 8254 и что означает режим 2,3?

5. Каким образом осуществляется формирование длительности звуков?
6. Какое прерывание вырабатывает системный таймер, и с какой частотой?

Лабораторная работа № 5 Перцептрон

Цель работы

Целью данной лабораторной работы является изучение работы персептрона и подготовка программы, позволяющей определить сдвиг одного восьмиразрядного двоичного числа относительно другого. (Возможно выполнение усложненного задания) Разпознавание образа знака на основе модели однослойного персептрона.

Общие сведения

Персептрон - простая нейронная модель, показанная на рис. 1 состоит элемента **Σ**, и пороговой функции. Элемент **Σ** умножает каждый вход *x* на вес *w* и суммирует взвешенные входы. Если эта сумма больше заданного порогового значения, выход равен единице, в противном случае – нулю. В зависимости от выходного сигнала принимают решение: 1 – входной сигнал принадлежит классу №1, 0 – классу №2.
Такие модели, а также модели, подобные им, применяют в распознавании образов, и в задачах классификации. Персептронные сети обладают простотой обучения и программной реализации, а также служат основой для изучения более сложных нейронных сетей. Ёмкость данной сети совпадает с числом нейронов в сети.

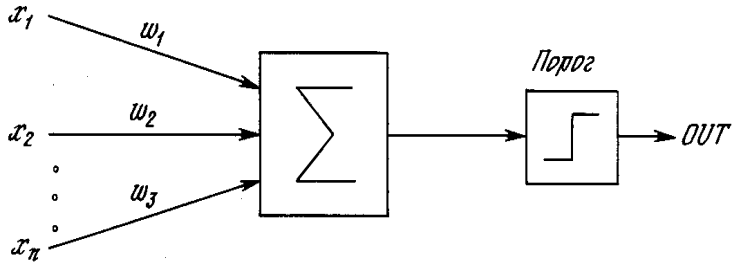


Рис.1. Математическая модель нейрона

1. Алгоритм обучения

1. Инициализация сети (задаём параметры: скорость обучения, величина порога, число обучений, веса w задаём случайным образом).
2. Подача входного образа (вектор Xi).
3. Подача желаемого выходного результата. Y.
4. Вычисление суммы Sj=∑Xi(t)*Wi(t).
5. Вычисление выходного сигнала Yвыч.=F(Sj), где F(Sj) – пороговая функция.
6. Проверка достижимости действительного выходного сигнала желаемому. Если достигли, переход на шаг2, иначе, подстраиваем весовые коэффициенты Wi(t).
7. Проверка окончания входных образов. Если закончились, то конец, иначе, переход на шаг2.

Ход работы

1. Ознакомится с теоретической частью по НС.
2. Составить алгоритм обучения НС методом обратного распространения ошибки.
3. Написать программу моделирования химической системы, реализующую этот алгоритм. (данные получить у преподавателя).
4. Ответить на контрольные вопросы.
5. Подготовить отчет и защитить лабораторную работу.

Пример реализации интерфейса для программного модуля персептрон

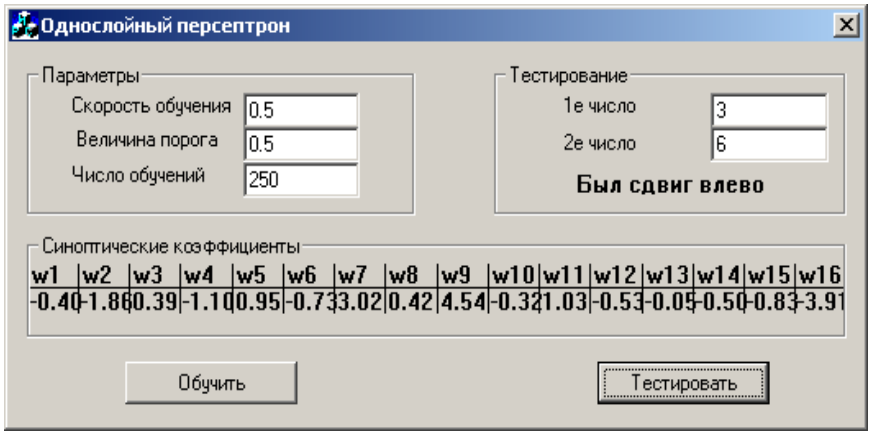


Рис. 2. Сдвиг влево

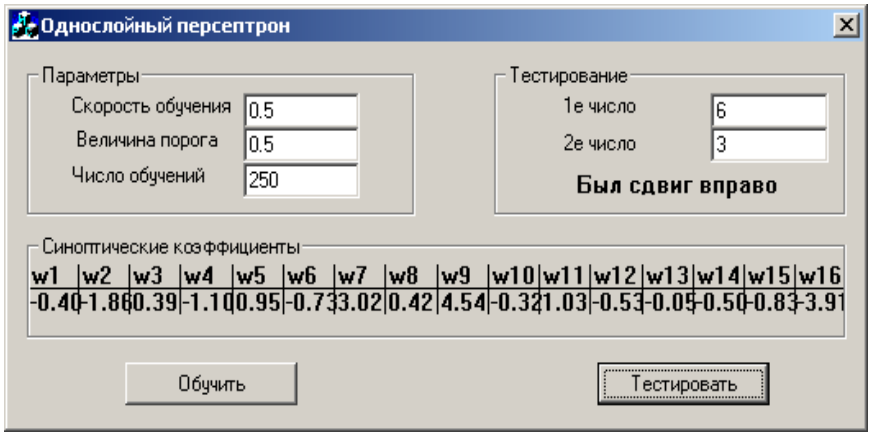


Рис. 3. Сдвиг вправо.

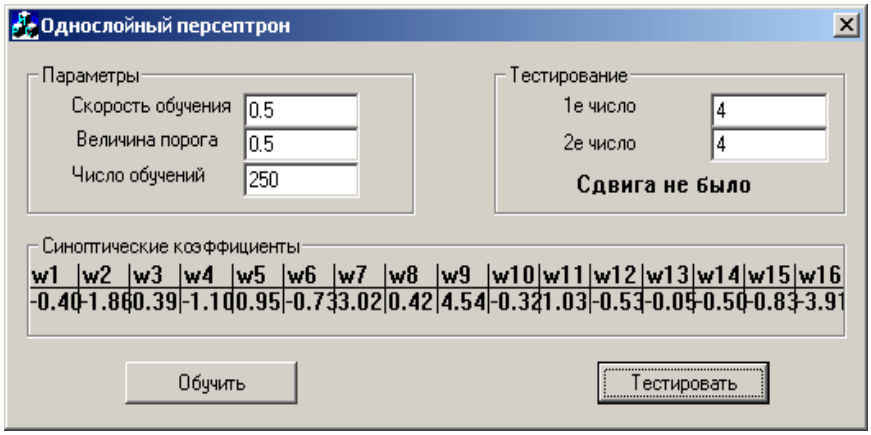


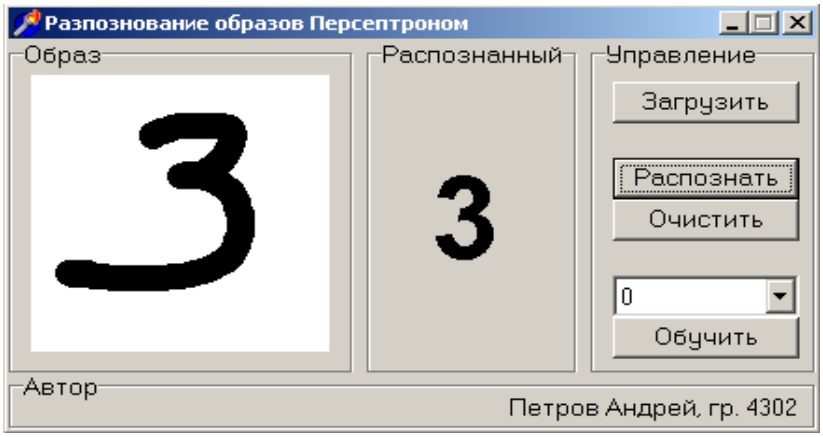
Рис. 4. Сдвига не было.

Также возможно выполнение усложненного задания с реализацией персептрона для распознавания образов цифр персептроном. Обучение можно выполнить на 3-х наборах цифр (3 шрифта).

0 3 7

Arial Comic Sans Times New Roman

Затем провести тестирование на рукописных цифрах с определением процентов правильных распознаваний. Пример реализации интерфейса приведен на рис. 5.



2. Содержание отчета

1. Цель работы
2. Ход работы
3. Описание работы перцептрона
4. Задание на лабораторную работу
5. Схема алгоритма
6. Результаты работы

7. Выводы
8. Листнг программы

Контрольные вопросы

1. Чем отличаются перцептроны с прямыми и обратными связями,
2. Чем отличается обучение с учителем от обучения без учителя.
3. Какие существуют методы обучения нейронных сетей.
4. какую функцию активации имеет однослойный перцептрон.
5. Как влияет скорость обучения на качество обуения перцептрона .
6. Почему порядок предъявления примеров в обучающей выборке может влиять на качество обучения.
7. В чем преимущества метода обратного распространения ошибки.
8. В чем его недостатки.
9. Как влияет уменьшение количества входных нейронов на функционирование сети.
10. Какими свойствами должна обладать функция активации при использовании алгоритма обратного распространения ошибки.

Лабораторная работа 6. Видеосистема компьютера

Цель работы

Изучение особенностей функционирования видеосистемы в текстовом режиме и приобретение практических навыков работы с видеомонитором в этом режиме.

Ход работы

Для организации управления экраном используется принцип асинхронного вывода с общей памятью, который заключается в том, что ЦП и видеоадаптер имеют доступ к общей памяти, используя ее в качестве общего буфера для информационного обмена между ними. Часть адресного пространства ОП (112 К, начиная с адреса A0000h) используется для адресации видеопамяти. Видеопамять входит в состав видеоадаптера. И представляет двупортовую память на плате видеоконтроллера (адаптера), в который микропроцессор записывает данные для изображения на экране. Экран периодически обновляется путем сканирования этих данных со стороны дисплея. Размер и расположение буфера в адресном пространстве зависит от типа видеоконтроллера и режима экрана. Когда в буфере хранится несколько изображений экрана, каждое отдельное изображение экрана называют дисплейной страницей.

Видеоадаптер имеет два разных режима работы: текстовый и графический. Для каждого из этих режимов имеется несколько форматов. Например для текстового режима основным является формат 80*25 – 25 строк по 80 символов в каждой.

Монохромный адаптер.

Имеет 4К байт памяти на плате, начиная с адреса B0000H (т.е. B000:0000H). Этой памяти хватает для хранения одной 80-символьной страницы текста.

Цветной графический адаптер (CGA).

CGA имеет 16К байт памяти на плате, начиная с адреса памяти B8000H. Этого достаточно для отображения одного графического экрана, или для отображения от четырех до восьми экранов текста в зависимости от числа символов в строке - 40 или 80.

Улучшенный графический адаптер (EGA).

EGA может иметь 64K, 128K или 256K памяти. Эта память, помимо ее использования в качестве видеобуфера, может также хранить последовательности битов для описания формы символов (до 1024 символов).

Стартовый адрес буфера дисплея программируемый, поэтому буфер, начиная с адреса A000H, используется для улучшенных графических режимов, а начиная с адресов B000H и B800H - для режимов, совместимых соответственно с монохромным и цветным графическими режимами. В большинстве случаев EGA занимает два сегмента с адресами от A000H до BFFFH, даже когда имеется 256K памяти. Это возможно, поскольку в некоторых режимах два или более байтов памяти дисплея считываются из одних и тех же адресов.

Доступное число страниц зависит как от режима экрана, так и от количества имеющейся памяти. Вследствие своей сложности EGA имеет ПЗУ на 16К байт, которое заменяет и расширяет процедуры BIOS работы с терминалом. Начало области ПЗУ - адрес C000:0000H.

Содержание буфера в текстовом режиме.

В текстовых режимах устанавливается следующее соответствие между памятью видеоконтроллера и изображением на экране. В начале памяти записываются данные о символе, находящемся на первой строке в левом углу, затем данные об остальных символах первой строки, затем данные о символах второй строки начиная слева и т.д.

При выводе текста различные видиосистемы работают одинаково. Для экрана отводится 4000 байт, так что на каждую из 2000 позиций экрана (25 строк x 80 символов) приходится 2 байта . Первый байт содержит код ASCII символа. Аппаратура дисплея преобразует номер кода ASCII в связанный с ним символ и посылает его изображение на экран. Второй байт (байт атрибутов) содержит информацию о том, как должен быть выведен данный символ. Для монохромного дисплея он устанавливает, будет ли данный символ подчеркнут, выделен яркостью или негативом, либо применяется комбинация этих атрибутов. В цветных системах байт атрибутов устанавливает основной и фоновый цвета символа, а также признак его мигания.

При записи данных прямо в буфер монитора значительно повышается скорость вывода на экран.

Видеоатрибуты символов.

Когда дисплей установлен в текстовом режиме в любой из видеосистем, каждой позиции символа на экране отводится два байта памяти. Первый байт содержит код ASCII символа, а второй - видеоатрибут символа. Цветной адаптер может выводить в цвете как символ, так и всю область, отведенную данному символу (фоновый цвет).

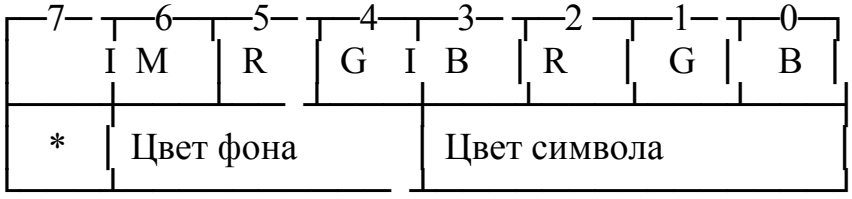
Монохромный адаптер ограничен только черным и белым цветом, но он может генерировать подчеркнутые символы, чего не может делать цветной адаптер.

EGA может делать все, что и остальные системы, а также многое другое. В частности, на улучшенном дисплее он выводит подчеркнутые цветные символы, поскольку матрица изображения символов 8*14 предоставляет такую возможность.

В таблице 1 приведен формат байта видеоатрибута

Табл. 1

Байт видеоатрибута



R - вхождение красного цвета :0=не входит;
1=входит

G - вхождение зеленого цвета:0=не входит;
1=входит
B - вхождение синего цвета :0=не входит;
1=входит
I - интенсивность: 0=символ неяркий;
1=символ яркий;
M - мерцание: 0=символ не мерцает;
1=символ мерцает.

* - возможен режим, когда этот бит будет определять не мигание символа, а интенсивность цвета фона.

Как цвет символа, так и цвет фона определяются как сумма трех основных цветов - красного, зеленого и синего. Все возможные цвета приведены в таблице 2.

Таблица 2

Генерация цвета символов и фона

I R G B	Цвет
0 0 0 0	черный
0 0 0 1	синий
0 0 1 0	зеленый
0 0 1 1	голубой (циан)
0 1 0 0	красный
0 1 0 1	розовый (магента)
0 1 1 0	коричневый
0 1 1 1	серый
1 0 0 0	темно-серый
1 0 0 1	ярко-синий

I R G B	Цвет
1 0 1 0	светло-зеленый
1 0 1 1	светло-голубой
1 1 0 0	светло-красный
1 1 0 1	светло-розовый
1 1 1 0	желтый
1 1 1 1	белый

Цвет мвола

Монохромные символы используют байт атрибутов несколько по другому. Как и с атрибутами цвета, биты 0-2 устанавливают цвет символа, а биты 4-6 - фона. Эти цвета могут быть только белым и черным.

Нормальный режим - белый на черном, когда биты 0-2 установлены в 111, а биты 4-6 установлены в 000. Негативное изображение создается обратными значениями битов. Символы выводятся с повышенной яркостью, когда бит 3 установлен в 1. Во

всех случаях установка в 1 бита 7 дает мигание символов. Возможны только 10 вариантов изображения монохромных символов. Они приведены в таблице 3.

Таблица 3

Цепочка битов	Атрибут
00000111	нормальный
00001111	интенсивный
00000001	нормальный-подчеркнутый
00001001	интенсивный подчеркнутый
01110000	негативный
10000111	нормальный мигающий
10001111	интенсивный мигающий
10000001	нормальный мигающий подчеркнутый
10001001	яркий мигающий подчеркнутый
11110000	яркий негативный

Регистры видеоконтроллера.

Во всех видеосистемах контроллер видеотерминала выполнен на базе микросхемы Motorola 6845 (EGA использует заказную микросхему, основанную на 6845).

Микросхема 6845 имеет 18 управляющих регистров, пронумерованных от 0 до 17. Первые 10 регистров фиксируют горизонтальные и вертикальные параметры дисплея. Эти регистры автоматически устанавливаются BIOS при изменении режима экрана. Не желательно экспериментировать с этими регистрами, поскольку можно испортить терминал. Регистры имеют размер 8 бит, но некоторые связаны в пары, чтобы хранить 16-битовые величины. Пары регистров 10-11 и 14-15 устанавливают форму и местоположение курсора. Пара 12-13 управляет страницами дисплея. Пара 16-17 определяет позицию светового пера.

Большинство регистров доступно только для записи; регистр адреса курсора можно и читать, и писать, а регистр светового пера предназначен только для чтения.

EGA имеет 6 добавочных регистров, из них наиболее интересен регистр 20. Он определяет, какая линия сканирования в строке символа используется для подчеркивания.

Доступ ко всем 18 регистрам осуществляется через один и тот же порт, адрес которого для монохромного адаптера - 3B5H, для цветного адаптера и PCjr - 3D5H. EGA использует один из этих двух адресов в зависимости от того, присоединен к нему цветной или монохромный монитор. Для записи в регистр надо сначала в регистр адреса, расположенный в порте 3B4H (3D4H для цветного), послать номер требуемого регистра. Тогда следующий байт, посланный в порт с адресом 3B5H, будет записан в этот регистр.

У CGA и MDA есть еще три регистра которые важны для программистов. Они имеют адреса 3B8H, 3B9H и 3BAH для монохромного и

3D8H, 3D9H и 3DAH - для цветного адаптера. Первый устанавливает режим экрана, второй связан в основном с установкой цветов экрана, а третий сообщает полезную информацию о статусе дисплея.

Цветовая палитра хранится в 16 регистрах палитры с номерами от 10H до 1FH.

EGA распределяет эти функции между микросхемой контроллера атрибутов (адрес порта 3C0H) и двумя микросхемами контроллера графики (адреса портов 3CCH-3CFH). Контроллер атрибутов содержит 16 регистров палитры EGA, пронумерованных от 00 до 0FH. Эти регистры могут содержать 6-битовые коды цветов, когда EGA связан с улучшенным цветным дисплеем, поэтому могут быть использованы любые 16 цветов из набора 64.

Режимы дисплея.

Возможные режимы приведены в таблице Таблица номер

Режим	Адаптеры
0	40*25(320*200) B&W алфавитно-цифровой цветной, PCjr, EGA
1	40*25(320*200) цветной алфавитно-цифровой цветной, PCjr, EGA
2	80*25(640*200) B&W алфавитно-цифровой цветной, PCjr, EGA
3	80*25(640*200) цветной алфавитно-цифровой цветной, PCjr, EGA
4	320*200 4-цветная графика цветной, PCjr, EGA
5	320*200 B&W графика (4 тени на PCjr) цветной, PCjr, EGA
6	640*200 B&W графика цветной, PCjr, EGA
7	80*25(720*350) B&W алфавитно-цифровой монохромный, EGA
8	160*200 16-цветная графика PCjr
9	320*200 16-цветная графика PCjr
A	640*2004 цветная графика PCjr
B	зарезервирован для EGA
C	зарезервирован для EGA
D	320*200 16-цветная графика EGA
E	640*200 16-цветная графика EGA
F	640*350 4-цветная графика на монохромном EGA
10	640*350 4- или 16-цветная графика EGA

Функция 0 прерывания 10H устанавливает режим дисплея. В AL должен находиться номер режима согласно табл. 4.

Залание. Установить текстовый (80*25) режим работы дисплея.

В соответствии с индивидуальным заданием вывести в заданное место экрана слова с требуемыми видеоатрибутами.

Варианты заданий на лабораторную работу

1.Первые два слова красным цветом на белом фоне; третье слово синим высокой интенсивности; четвертое слово зеленое мигающее на

красном.

2.Первые три слова синим цветом на красном фоне; второе слово высокой интенсивности.

3.Второе слово магента на черном фоне; пятое слово мигает; седьмое слово белым низкой интенсивности.

4.Первые три слова синим на белом фоне; четвертое слово красным высокой интенсивности; шестое слово зеленым мигающим.

5.Седьмое слово белым черном фоне; пятое слово красным цветом высокой интенсивности; первые три слова синим цветом мигают.

6.Два слова цвета циан на красном фоне; третье слово мигает; первое слово коричневым высокой интенсивности.

7.Первые два слова синим высокой интенсивности; третье слово мигает; четвертое слово зеленое на красном фоне.

8.Первое слово черным на белом фоне; четвертое слово красным на зеленом фоне мигает; шестое слово коричневым высокой интенсивности.

9.Второе слово розовое высокой интенсивности; третье слово циан на зеленом фоне; пятое слово мигает.

10.Первые два слова синим низкой интенсивности; третье слово мигает; четвертое слово красным цветом на белом фоне.

11.Четные слова голубым по розовому; нечетные - красным по

черному; последнее пятое слово мигает.

12.Первое слово черным на белом фоне; четвертое слово зеленое на красном фоне; остальные - голубым на черном фоне.

Указанные слова вывести на экран в строку и в столбец, номера которых совпадают с номером студента в журнале.

7. Содержание отчета

7.1.Тема лабораторной работы.
7.2.Цель работы.

7.3.Индивидуальное задание.
7.4.Текст программы.
7.5.Результаты работы программы.

Контрольные вопросы

1. Какие регистры отвечают за формирование цвета
2. Какие функции поддерживают видео режим
3. Что такое страницы видеопамяти
4. Сколько и какие регистры отвечают за растр экрана
5. Какая часть адресного пространства используется для видеопамяти.

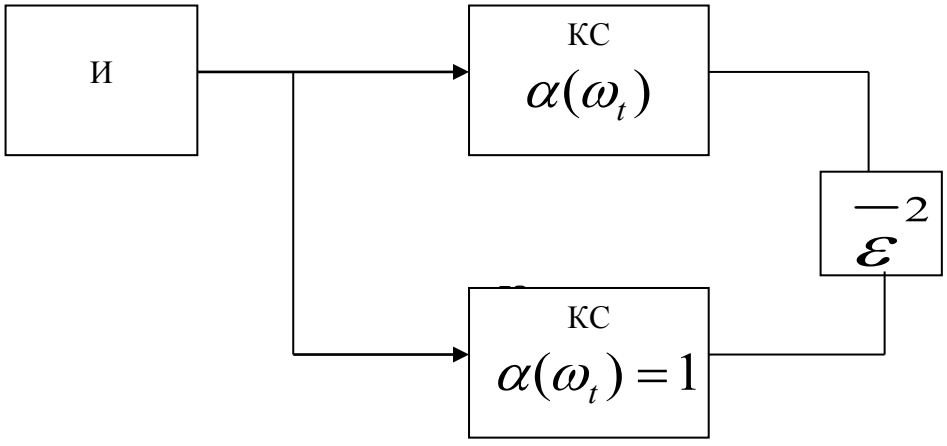
Лабораторная работа 7. Исследование канала связи в сетях ЭВМ в виде RLC-цепи

Цель работы.

Цель работы состоит в изучении метода оценки качества канала связи на основе потерь энергии при передаче сигналов определенной формы по этому каналу. В качестве канала принять RLC-цепь.

Ход работы.

Для изучения качества передачи канала связи наиболее целесообразно использовать критерий в виде потерь энергии сигнала. В нашем случае был выбран прямоугольный сигнал. Для анализа будем использовать следующую модель:



где И - источник, формирующий сигнал U(t),

$\alpha(\omega_t)$ – передаточная характеристика канала,

ε^2 – потери энергии сигнала связи.

Для оценки потерь энергии сигнала необходимо найти разность энергий сигналов, прошедших реальный канал с передаточной характеристикой $\alpha(\omega_t)$ и идеальный канал с $\alpha(\omega_t) = 1$. Тогда энергия потерь ε^2 будет иметь вид:

$$\overline{\varepsilon}^2 = \frac{1}{2\pi}\int_{-\infty}^{\infty} |W(\omega_t)|^2 * d\omega_t - \frac{1}{2\pi}\int_{-\infty}^{\infty} |W(\omega_t)|^2 * |\alpha(\omega_t)|^2 \, d\omega_t =$$

$$= \frac{1}{2\pi}\int_{-\infty}^{\infty} |W(\omega_t)|^2 * (1 - |\alpha(\omega_t)|^2) d\omega_t,$$

где $W(\omega_t)$ – спектральная функция сигнала,

$\alpha(\omega_t)$ – передаточная характеристика канала,

$\overline{\varepsilon}^2$ - потери энергии, усредненные по всем частотам

Прямоугольный сигнал описывается функцией:

$$U(t) := \begin{cases} U0 & \text{if } 0 < t < Ti \\ 0 & \text{otherwise} \end{cases},$$

и имеет вид, представленный на рис. 1. U_0 и T были взяты в интервале от 0 до 1, так как при таких значениях потери энергии будут незначительными. T=0,8 U_0 =0.9

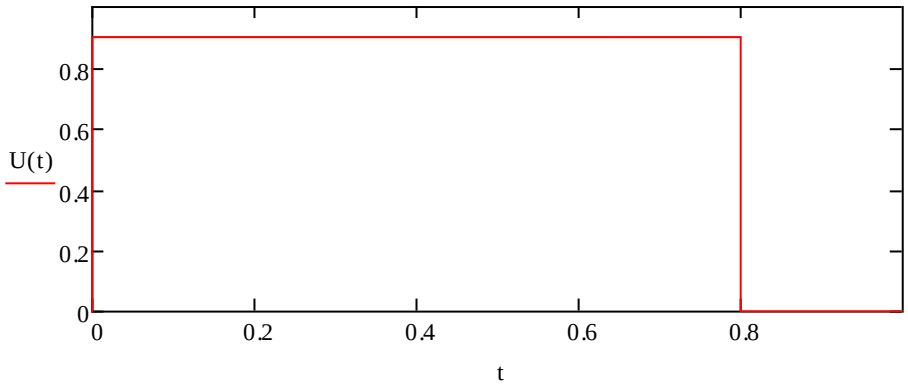


Рис. 1 – Функция прямоугольного сигнала.

Спектральная функция сигнала и зависимость W(ω_t) приведены рис.2:

$$W(\omega_t) = U_0 T_i * \frac{\sin\frac{\omega_t T}{2}}{\frac{\omega_w T}{2}}$$

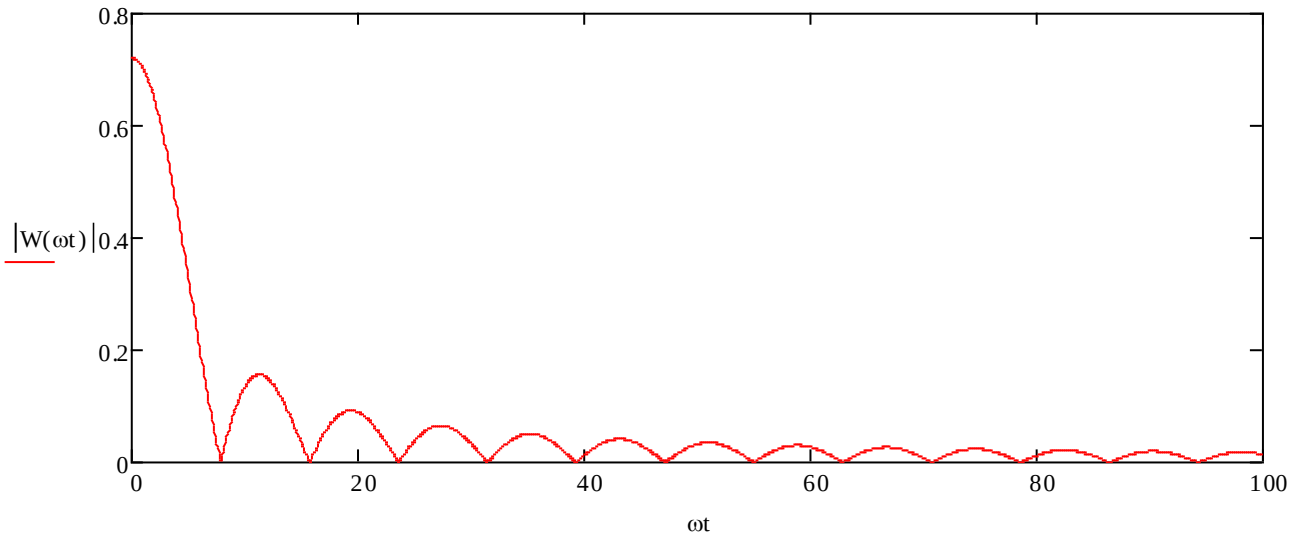


Рис. 2 – Спектральная функция сигнала

Для описания функции пропускания канала была использована интегральная RLC-цепь. В этом случае передаточная характеристика $\alpha(\omega_t)$ определяется соотношением

$$\alpha(\omega_t) = \frac{Q}{\sqrt{1+(2\cdot\frac{Q\cdot(\omega_t-\omega_0)}{\omega_0})^2}}, \text{где } Q = \frac{\rho}{R}, \rho = \sqrt{\frac{L}{C}}, \omega_0 = \frac{1}{\sqrt{L\cdot C}}$$

В нашем случае R=30 Ом, C=0.4 мкФ, L=10^{-3} (значения подобрать самостоятельно)

На рис. 3 представлена передаточная характеристика канала.

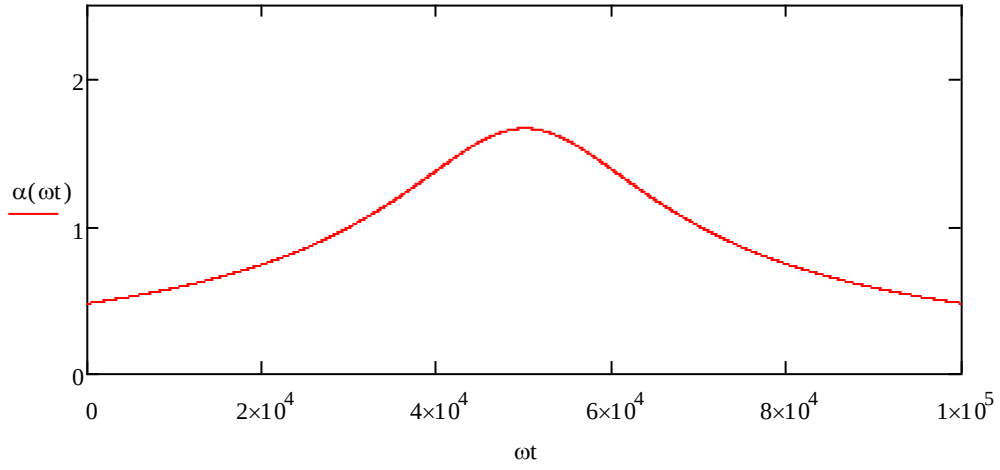


Рис.3 – Передаточная характеристика канала.

Для наглядного выявления частот сигнала, которые не пройдут при использовании канала связи, зависимости пропускных способностей реального и идеального каналов представлены на одном графике (рис. 4).

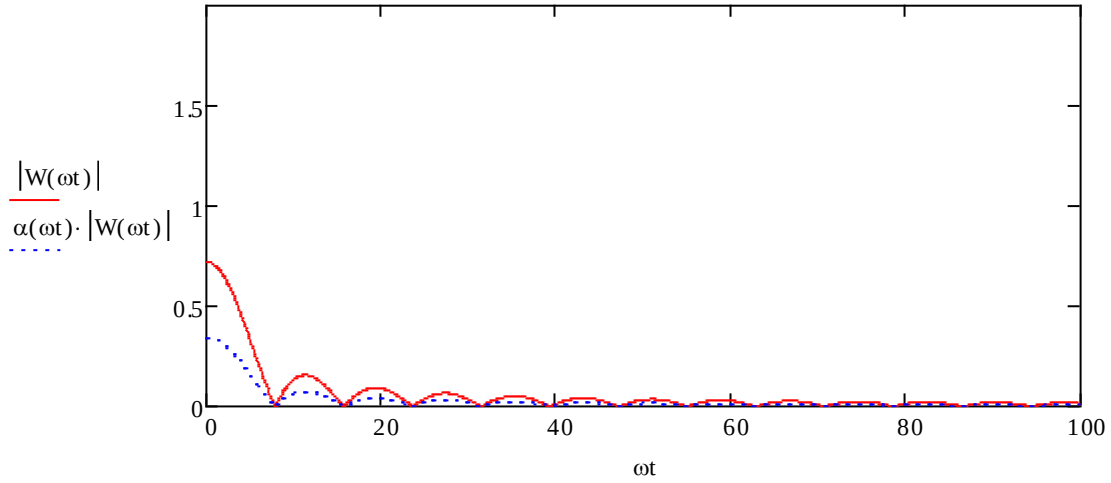


Рис.4 – пропускные способности реального и идеального каналов.

Энергии для идеального и реального каналов определяются по следующим формулам:

$$E1 := \frac{1}{2\pi} \cdot \int_{-\infty}^{\infty} (|W(\omega t)|)^2 \, d\omega t$$

$$E2 := \frac{1}{2\pi} \cdot \int_{-\infty}^{\infty} (|W(\omega t)|)^2 \cdot (|\alpha(\omega t)|)^2 \, d\omega t$$

$$E1 = 0.648$$

$E2 = 0.149$

Количество энергии, которое теряется при несовпадении полосы частот сигнала и канала определяется формулой:

$$\varepsilon(\omega t) := (|W(\omega t)|)^2 \cdot \left[1 - (|\alpha(\omega t)|)^2\right]$$

График зависимости количества энергии, которая теряется при несовпадении полосы частот и канала приведен на рис.5.

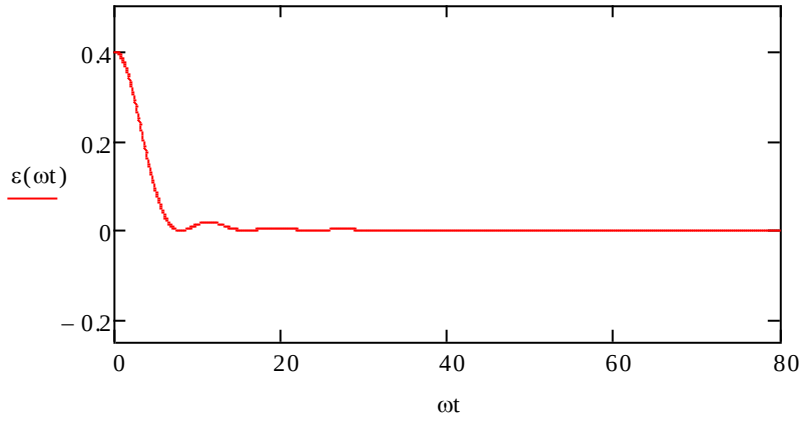


Рисунок 5 - График зависимости $\varepsilon(\omega t) := (|W(\omega t, t)|)^2 \cdot \left[1 - (|\alpha(\omega t)|)^2\right]$

Результаты работы.

В результате проделанной работы был изучен метод оценки качества канала связи на основе потерь энергии при передаче сигналов определенной формы по этому каналу.

Контрольные вопросы

1. Для какой цели предназначен канал связи в сетях ЭВМ
2. Что такое спектральная функция сигнала
3. Каким образом представляется спектральная функция сигнала
4. Каким образом представляется передаточная функция канала
5. Почему в цифровых сетях эвм используют цифровое представление сигналов.

Содержание отчета

1.Цель работы

2.Краткое содержание хода работы

3. Формула и график спектральной функции сигнала

4. Формула и график передаточной функции канала

5. Графики пропускной способности реального и идеального каналов

6. График зависимости потерь энергии сигнала

7. Выводы по работе

8. Ответы на контрольные вопросы.

8.РАБОТА С ОТЛАДЧИКОМ TURBO DEBUGGER (TD)

Tasm предназначен для ассемблирования синтаксически правильных программ, но как эта программа работает не понимает. Поэтому часто программа работает не так, как должна была бы работать. *TD*-программа, разработанна для поиска и исправления логических ошибок. Подобно всем отладчикам *TD* может работать в режиме супервизора, беря на себя управление программой в режиме пошагового исполнения, кода программы. Можно при этом изменять значения операндов в памяти, а также значения регистров и флагов. *TD* используется и в качестве учителя при изучении форматов машинных команд процессора в различных режимах адресации операндов.

Чтобы показать, как использовать *TD* при изучении языка ассемблера, исследуем программу *Hello* под управлением отладчика. Произведём заново ассемблирование и компоновку программы с опциями, которые добавляют отладочную информацию в *obj*- и *exe*-файлы:

tasm/zi hello.asm

tlink/v hello.obj

td hello.exe

После выполнения последней команды на экране увидите окно *Module* с исходным текстом программы *Hello.asm*. Это окно номер 1. Просмотреть программу можно, используя клавиши управления курсором, передвигая его вверх и вниз по тексту.

Для получения другого представления программы войдите в меню *Viev* [обзор], выберите команду *CPU* и нажмите <*F5*> для распахивания окна на весь экран. В *CPU*-окне, состоящем из пяти областей, содержится в сокращённом виде исходный текст вашей программы, действительные машинные коды, находящиеся в памяти, значения регистров и флагов, стек и дамп байтов памяти. Для передвижения курсора из одной области в другую нажимайте клавишу <*Tab*>.

Область окна *CPU,* в которой находится курсор, считается активной. Нажатие клавиши <*Alt+F10*> вызывает появление локального меню для активной секции окна *CPU*. Перейдите в главную область окна *CPU* и нажмите <*Alt+F10*>. Выберите команду *Mixed* (смесь), которая имеет три установки: *No, Yes* и *Both*. Режим *Both (*оба) устанавливается по умолчанию и является наилучшим способом просмотра, показывая в левой колонке байты машинного кода, а в правой – строки исходного текста программы.

Окно *CPU* используется для наблюдения за текущим состоянием процессора при пошаговом выполнении инструкций программы. Маленькая стрелка-треугольник слева от первой команды *mov ax,@data* показывает, что она является следующей исполняемой командой. Для выполнения этой команды нажмите <*F8*>, стрелка перейдёт на следующую команду с изменённым значением регистра *ax* в окне регистров. Снова нажмите <*F8*>, для выполнения инструкции *mov ds,ax*. Обратите внимание, что значение в регистре *ds* стало таким же, как и в регистре *ax*, но произошло изменение сегментации дампа памяти (левая нижняя область) с регистра *ds* на регистр *es*.

Сделайте активной область с дампом памяти, нажмите <*Alt+F10*> и в появившемся локальном меню выберите команду *Goto* (переход). Появится заставка, в которой наберите *ds*:*0000* и нажмите <*Enter*>. Теперь дамп памяти будет соответствовать массиву инициализированных данных вашей программы при их побайтовом представлении (*ASCII*-код символов переменной *Promt, Good Morning и Good Afternoon*). Для просмотра всего дампа используйте передвижение курсора с помощью стандартных клавиш. Объяснение этому факту следует из особенностей загрузки операционной системой MS-DOS в память программ с расширением *exe* (рис. 1.2).

Чтобы лучше разобраться с представлением переменной *Promt* в дампе памяти, войдите в меню *Data* (данные), выберите команду *Inspect* (проверка). В появившейся заставке наберите *Promt* и нажмите <*Enter*>. Раздвиньте с помощью мыши появившееся окно *Inspecting Promt* по вертикали на всю высоту экрана в правой его части. В левой колонке данного окна указывается номер элемента переменной массива *Promt[i]*, а в правой – значение *ASCII*-кода этого элемента. Найдите эти коды в дампе памяти, а именно:

ds:0000 exCode=00h

ds:0001 Promt[0]='Э'=9Dh

ds:0002 Promt[1]='т'=E2h

ds:0003 Promt[2]='о'=AEh

и т. д.

Продолжим покомандное исполнение программы с помощью клавиши <*F8*>, предварительно закрыв окно *Inspecting Promt*, щёлкнув мышью по кнопке закрытия. После выполнения каждой команды наблюдаем за изменением значений регистров.

Описанный выше процесс выполнения программы прерывается после исполнения команды прерывания DOS (*Int 21*,функция *ah* = 1) по вводу символа с клавиатуры на

экран. *TD* переводит вас с окна *CPU* в окно пользователя *User Screen*, в котором появляется сообщение-запрос переменной *Promt.* Ответив на этот запрос (прописной или строчной буквой: *Y* или *y*), вы автоматически снова перейдёте в окно *CPU*, для дальнейшего покомандного выполнения программы. Переход, при необходимости, к окну пользователя от *TD* и обратно можно осуществить нажатием клавиш <*Alt+F5*>. Если вы забыли это сочетание клавиш, то обратитесь к меню управления окнами *Window*, найдите строку *User Screen* с уже упомянутыми клавишами. Кстати, ниже этой строки будет приведён список всех открытых вами окон *TD*, а именно:

Module Hello (1)

CPU (2)

Inspector (3)

Можно вызвать любое открытое вами окно с помощью нажатия клавиш <*Alt+ номер окна*> или путём их последовательного перебора с помощью клавиши <*F6-Next*>. Если вы хотите закрепить данное расположение окон при следующей работе с *TD*, то войдите в меню *Options* (параметры), выберите кнопку с командой *Layot* (схема окон). Данная схема будет зафиксирована утилитой *Tdconfig.dt*. Раз уж мы находимся в меню *Options,* укажем здесь ещё на одну полезную команду *Display options* (вывести параметры). Всплывающая заставка *Display swapping* (переключение экрана) позволяет выбрать один из трёх способов управления переключением между экраном *TD* и экраном пользователя. По умолчанию устанавливается параметр *Smart* (эффективный), который позволяет переключиться на экран пользователя *User Screen* автоматически по требованию программы (это мы наблюдали только что при выполнении требования программы *Hello* операции ввода одиночного символа).

Вернёмся снова в окно *CPU* для завершения работы с программой *Hello*. Используя клавишу <*F8*>, доводим маркер исполнения текущей команды до команды с меткой *Exit* (вызов функции DOS с номером 4Ch для выхода из программы) и, нажав <*Alt+F5*>, увидим на экране пользователя ответ программы на наш диалог с ней. Если вы хотите повторить выполнение программы, то она может быть перезагружена с произвольной команды с помощью нажатия клавиш <*Ctrl+F2*> или командой меню *Run = Program Reset* (сброс программы). При этом программа снова загружается с диска и *TD* восстанавливает свои исходные опции. Если вы находитесь в окне *CPU* или *Module*, то на дисплее не будет показан возврат к началу вашей программы – для этого надо нажать <*F8*>.

режимы исполнения программы в *TD.*

- Режим автоматического исполнения, клавиша *F9 (Run).*

- Выполнение по шагам (клавиши *F8* или *F7-Trace into*). Отличие в назначении этих клавиш проявляется в том, что клавиша *F7* используется для пошагового исполнения тела цикла, процедуры или подпрограммы обработки прерывания. Клавиша же *F8* исполняет эти процедуры как одну обычную команду и передаёт управление следующей команде программы.

· Выполнение до текущего положения курсора. Для активизации этого режима необходимо установить курсор на нужную строку программы (строка будет подсвечиваться другим цветом) и нажать клавишу *F4*.

· Выполнение с установленными точками прерывания (*Breakpoints*). Перед исполнением программы необходимо установить эти точки, для чего следует перейти в нужную строку программы и нажать клавишу *F2* (*toggle*). Выбранная строка с контрольной точкой подсвечивается красным цветом. Чтобы убрать контрольную точку, надо повторить эту операцию снова. После установки точек прерывания программа запускается на исполнение клавишей *F9*. После первого нажатия клавиши *F9* программа остановится на первой точке прерывания, после второго – на второй точке и.т.д. Это очень удобный режим отладки программы, когда необходимо контролировать правильность её исполнения в некоторых характерных точках.

Завершать работу с отладчиком следует командой *File = Quit* или с помощью клавиш <*Alt+X*>. *TD* имеет множество других возможностей, изучить которые вам предлагается самостоятельно при исследовании демонстрационных программ, приведённых в приложении П.1.2 к данной работе

9. Методические указания по работе с системой моделирования электронных схем Electronics Workbench (EWB)

Программный комплекс EWB разработан фирмой Interactive Image Technologies (Канада) для схемотехнического моделирования цифровых и аналоговых радиоэлектронных устройств.

Предварительное исследование электронной схемы с применением компьютерного моделирования позволяет найти оптимальные параметры для работы исследуемого устройства,
не прибегая к его практической реализации. Исследование на программной модели позволяет ознакомиться с возможностями
проверки правильности построения схем. При разработке сложных
схем физическое моделирование бывает просто невозможно из-за чрезвычайной сложности устройства.

Особенность программы EWB в наличии в ней контрольно-измерительных приборов, по внешнему виду, органам управления и характеристикам максимально приближенных к их промышленным аналогам.

Программа EWB 4.1 рассчитана для работы в среде Windows 3.xx или 95.98 и занимает около 5 Мбайт дисковой памяти, EWB 5.0 - в среде Windows 95/98 и NT 3.51, требуемый объём дисковой памяти - около 16 Мбайт. Для размещения временных файлов требуется дополнительно 10.....20 Мбайт свободного пространства.

В данном руководстве рассматривается версия EWB5PRO и EWB v.5.12.

Структура окна и система меню.

Окно содержит строку команд меню, строку основных типовых
электронных устройств , поле для составления исследуемой схемы
и полосы управления прокруткой.

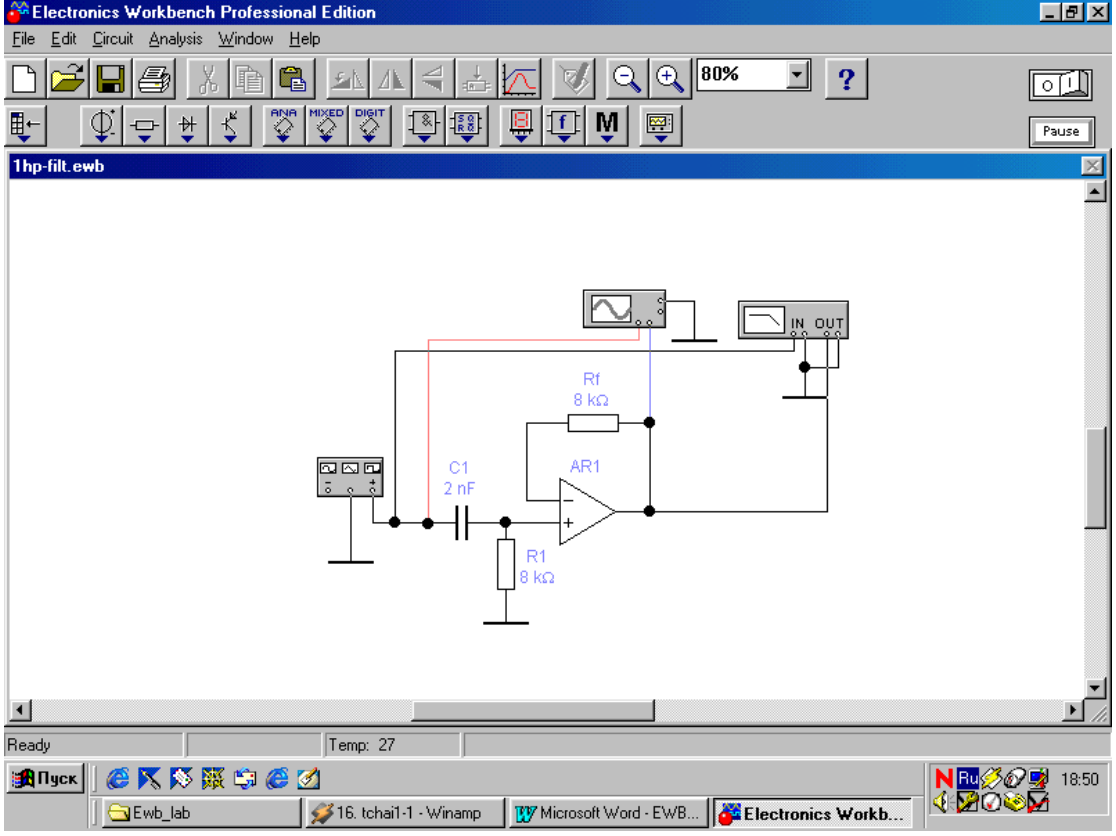
Основные команды меню:

Меню File:

первые четыре команды меню типовые и пояснений не требуют.

-**Revent to Saved** -стирание всех изменений, внесенных в текущем сеансе редактирования, и восстановление схемы в первоначальном виде.

- **Install** - установка дополнительных программ с жёстких дисков.
- **Import** - импорт текстовых файлов описания схемы .
- **Export** - составление текстового описания схемы и задания на моделирование в формате SPICE.



Окно программы EWB 5.x.

Меню Edit:.

- **CUT -** стирание (вырезание) выделенной части схемы с сохранением в буфере обмена. Выделение одного компонента производится щелчком мыши на изображении компонента. Для выделения части схемы или нескольких компонентов курсор мыши в левый угол воображаемого прямоугольника, охватывающего выделяемую часть, нажать левую кнопку мыши и, не отпуская её, протянуть курсор по диагонали этого прямоугольника, контуры которого появляются уже в начале движения мыши, и затем отпустить кнопку. Выделенные компоненты окрашиваются в красный цвет.
- **COPY -** копирование выделенной части схемы в буфер обмена.
- **PAST-** вставка содержимого буфера обмена на рабочее поле программы. Фрагмент затем ещё будучи отмеченным перетаскивается с помощью мыши в нужное место.
- **DELETE -** стирание выделенной части схемы.
- **SELECT ALL -** выделение всей схемы.
- **COPYBITS -** команда превращает курсор мыши в крестик, которым по правилу прямоугольника можно выделить нужную часть экрана, после отпускания левой кнопки мыши выделенная часть копируется в буфер обмена, после чего его содержимое может быть импортировано в любое приложение Windows. Копирование всего экрана производится нажатием клавиш Print Screen; копирование активной в данный момент части экрана, например, диалогового окна - комбинацией Alt+Print Screen.
- **Show Clipboard-** показать содержимое буфера обмена.
- **Copy as Bitmap -** копирует выделенный участок в буфер обмена.

Меню Circuit - используется при подготовке схем, а также для задания параметров моделирования.

- **Activat** - запуск моделирования.
- **Stop** - остановка моделирования. Эти две команды дублируются нажатием кнопки выключателя, расположенного в правом верхнем углу экрана.
- **Pause -** прерывание моделирования.
- **Label -** ввод позиционного обозначения выделенного компонента с помощью диалогового окна.
- **Value** - изменение номинального значения параметра компонента с помощью диалогового окна.
- **Model** - выбор модели компонента, команда выполняется также двойным щелчком по компоненту. Работа с меню, как и во всех других подобных случаях, заканчивается нажатием кнопок **Accept** или **Cancel -** с сохранением или без сохранения введённых изменений.
- **Zoom** - раскрытие (развёртывание) выделенной подсхемы или контрольно-измерительного прибора, команда выполняется также двойным щелчком мыши по иконке компонента или прибора.
- **Rotate**- вращение выделенного компонента.
- **Fault -** имитация неисправности выделенного компонента путём введения :
  -leakage- сопротивления утечки,
  -short - короткого замыкания,
  -open - обрыва,
  - none - отсутствие неисправности (включено по умолчанию).
- **Subcircuit** - преобразование предварительно выделенной части схемы в подсхему.
- **Wire Color** - изменение цвета предварительно выделенного проводника. Расцветка проводников важна в случае применения логического анализатора, - в этом случае цвет проводника определяет цвет временной диаграммы.
- **Preferences-** выбор элементов оформления схемы в соответствии с меню.

Для создания схем, рассматриваемых в рамках лабораторных работ по дисциплине "организация ЭВМ и систем" достаточно воспользоваться имеющимися типовыми компонентами.

Для открытия нужной библиотеки компонентов нужно подвести курсор мыши к соответствующей иконке и нажать один раз её левую кнопку. В выпадающем множестве выбирается необходимый значок, и передвигается при удержании левой клавишей мыши на рабочее поле программы. Для установки параметров необходимо двойным нажатием левой кнопкой мыши раскрыть меню настройки параметров компонента. Выбор подтверждается нажатием кнопкой **Accept** и клавишей **Enter.**

После размещения компонентов производится соединение их выводов проводниками. При этом необходимо учитывать, что к выводу компонента можно подключить только один проводник.

Для выполнения подключения курсор мыши подводится к выводу компонента и после появления прямоугольной площадки синего цвета, нажимается левая кнопка и появляющийся при этом проводник протягивается к выводу другого компонента до

появления на нём такой же прямоугольной площадки, после чего кнопка мыши отпускается и соединение готово. При необходимости
подключения к этим выводам других проводников в библиотеке Passive выбирается точка (символ соединения) и переносится на ранее установленный проводник. После удачной постановки точки
к проводнику подсоединяется ещё два проводника.
Точка соединения может быть использована не только для подключения проводников, но и для введения надписей.
Если необходимо переместить отдельный сегмент проводника, к нему подводится курсор, нажимается левая кнопка и после появления в вертикальной или горизонтальной плоскости двойного курсора производятся нужные перемещения.
Подключение к схеме контрольно-измерительных приборов производится аналогично. Причём для таких приборов, как осциллограф или логический анализатор, соединения целесообразно проводить цветными проводниками, поскольку их цвет определяет соответствующую осциллограмму.

Основные компоненты EWB.
Компонент **Выход из EWB.**

Вспомогательные компоненты **-группа SOURCES**:

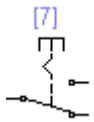
- заземление (метка) . точка нулевого потенциала в схеме.

- источник фиксированного напряжения +5 вольт

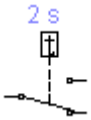
- генератор однополярных прямоугольных импульсов (амплитуда, частота, коэффициент заполнения).

Основные пассивные элементы - **группа BASIC**:

- точка соединения проводников, используется также для введения на схему надписей длиной не более 14 символов (других способов введения текста в EWB не существует).



- переключатель, управляемый нажатием задаваемой клавишей клавиатуры (в квадратных скобках), по умолчанию- клавиша пробела.
- переключатель, автоматически срабатывающий через заданное время на включение

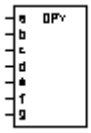


и выключение (время в секундах).

Индикаторные приборы - **группа INDICATORS**.

- светоиндикатор (свет свечения может быть настроен красным, зелёным и синим)

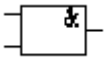
- семисегментный индикатор с дешифратором .



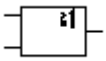
- семисегментный индикатор .

- лампа накаливания.

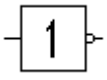
Логические элементы - **группа LOGIC GATES**



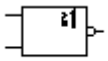
- логический элемент "И"



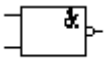
- логический элемент "ИЛИ"



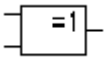
- логический элемент "НЕ"



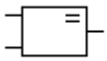
- логический элемент "ИЛИ-НЕ"



- логический элемент "И-НЕ"

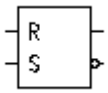


- логический элемент исключающее "ИЛИ"

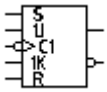


- логический элемент импликация

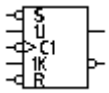
4.6. Комбинированные цифровые компоненты.



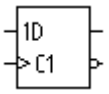
- асинхронный RS-триггер



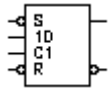
- универсальный JK-триггер с прямым тактовым входом и входами предустановки



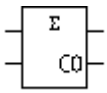
- универсальный JK-триггер с инверсным тактовым входом и инверсными входами предустановки



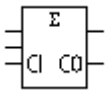
- D-триггер без предустановки



- D- со входами предустановки



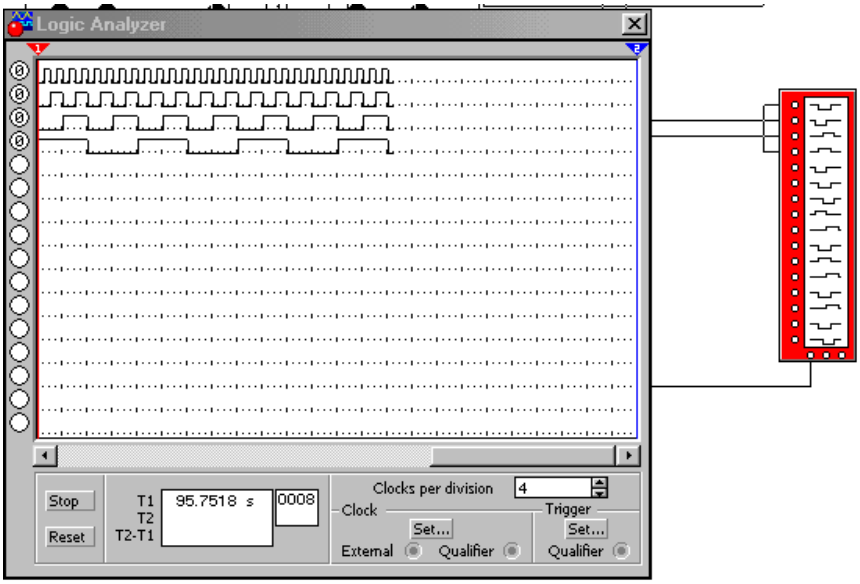
- полусумматор

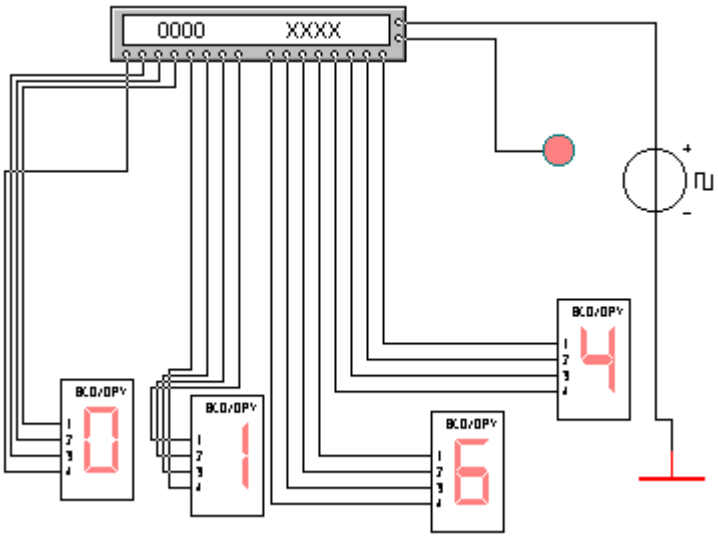


- полный сумматор

Приборы, группа **INSTRUMENTS.**

- логический анализатор

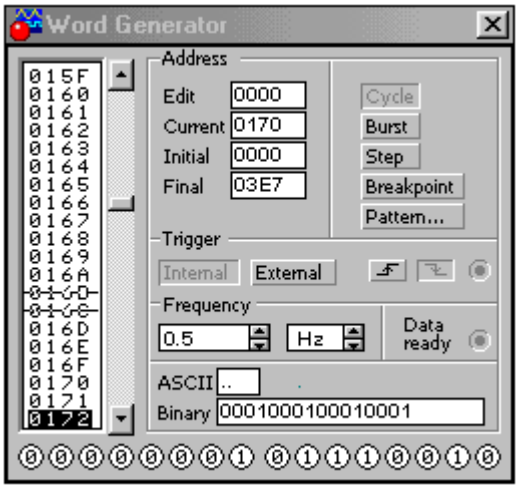




- генератор слова - **Word Generator**

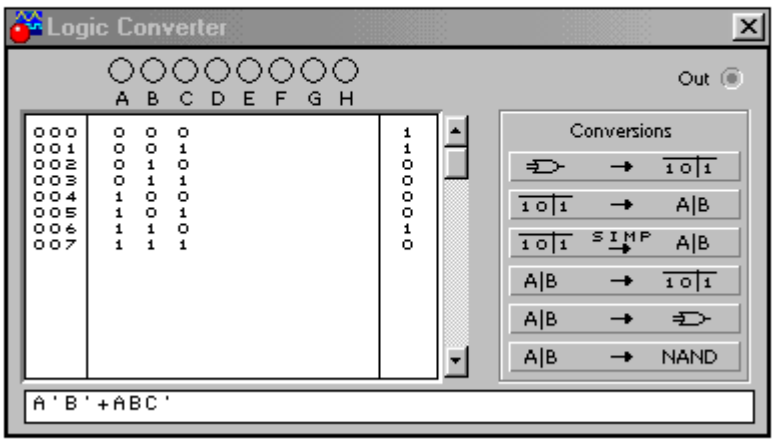
На первом рисунке показан генератор слова с подключенными семисегментными индикаторами и внешним генератором синхроимпульсов.

На втором рисунке генератор слова показан в развёрнутом виде.
Генератор (или кодовый генератор) предназначен для генерации
16-ти 16-ти разрядных двоичных слов, которые набираются пользователем на экране, расположенным в левой части лицевой панели. Для набора двоичных комбинаций



необходимо щёлкнуть мышью на соответствующем разряде и затем ввести с клавиатуры
число в десятичном коде.

Сформированные слова выдаются на шестнадцать расположенных в нижней части прибора выходных клемм-индикаторов:
- с индикацией в двоичном коде в строке окна binary;
- в пошаговом (step), циклическим (cycle) или с выбранного слова до конца (при нажатии кнопки BURST) при заданной частоте посылок (установка- заданием частоты в окнах FREQUENCY);
- при внутреннем или внешнем запуске (при нажатии кнопки EXTERNAL, справа верхняя клемма служит для подключения
сигнала синхронизации);
- призапуске по переднему или заднему фронту сигнала синхронизации служит кнопка
- на правую нижнюю клемму выдается выходной синхронизирующий импульс.



Логический преобразователь- **Logic Converter.**
На лицевой панели преобразователя показаны клеммы-индикаторы входов A,B,.....H и одного выхода OUT, экран для отображения таблицы истинности исследуемой схемы, экран-строка для отображения её булева выражения (в нижней части).

Логический анализ n-входового устройства с одним выходом
может осуществлять следущюие действия, используя кнопки управления:

1.

- таблицу истинности исследуемого устройства;

2.

- булево выражение, реализуемое устройством;

3.

- минимизированное булево выражение;

4.

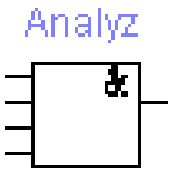
- схему устройства на логических элементах без ограничения их типа;

5.

- схему устройства только на логических элементах И-НЕ.

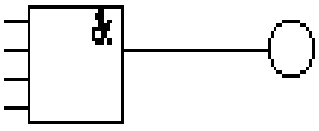
Пример составления исследуемой схемы.

Собрать схему логического элемента "И".
В группе Logic Gates, выбирается логический элемент "И".



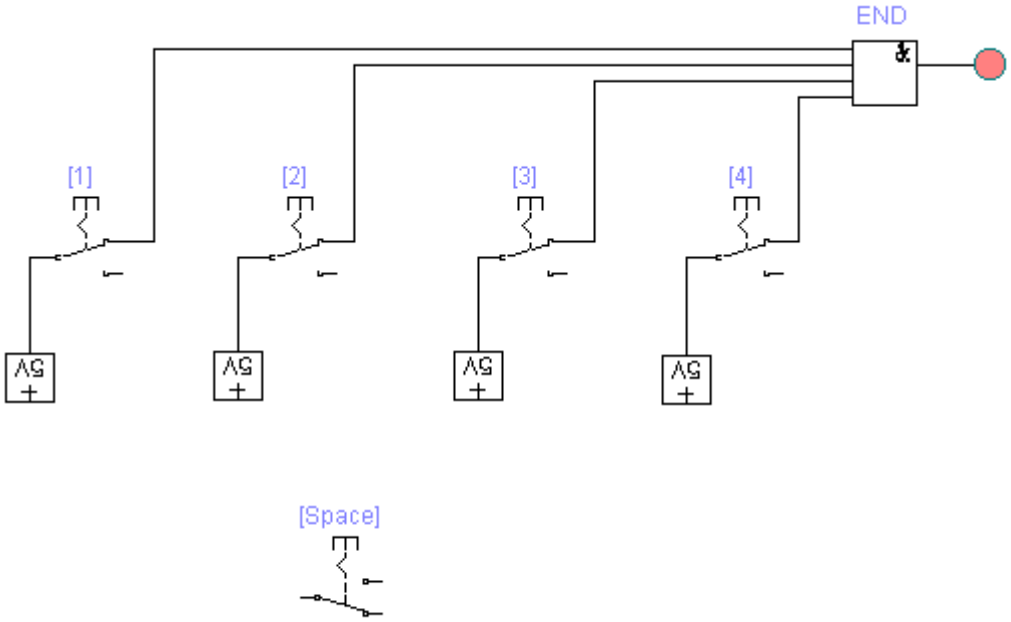
Двумя щелчками мыши на изображении логического элемента переходим к настройкам параметров логического
элемента "И". Выбираем количество входов, например 4.
Можно присвоить название логическому элементу.

К выходу логического элемента присоединяем из группы **INDICATORS** красный светодиод.

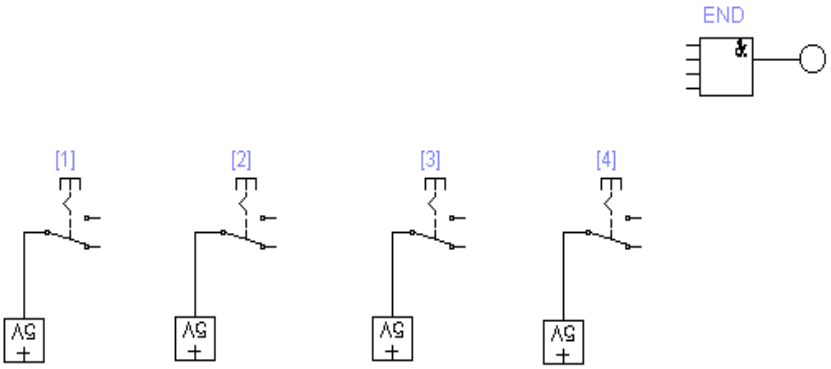


Для получения логического сигнала (0 или 1) удобно воспользоваться источником напряжения

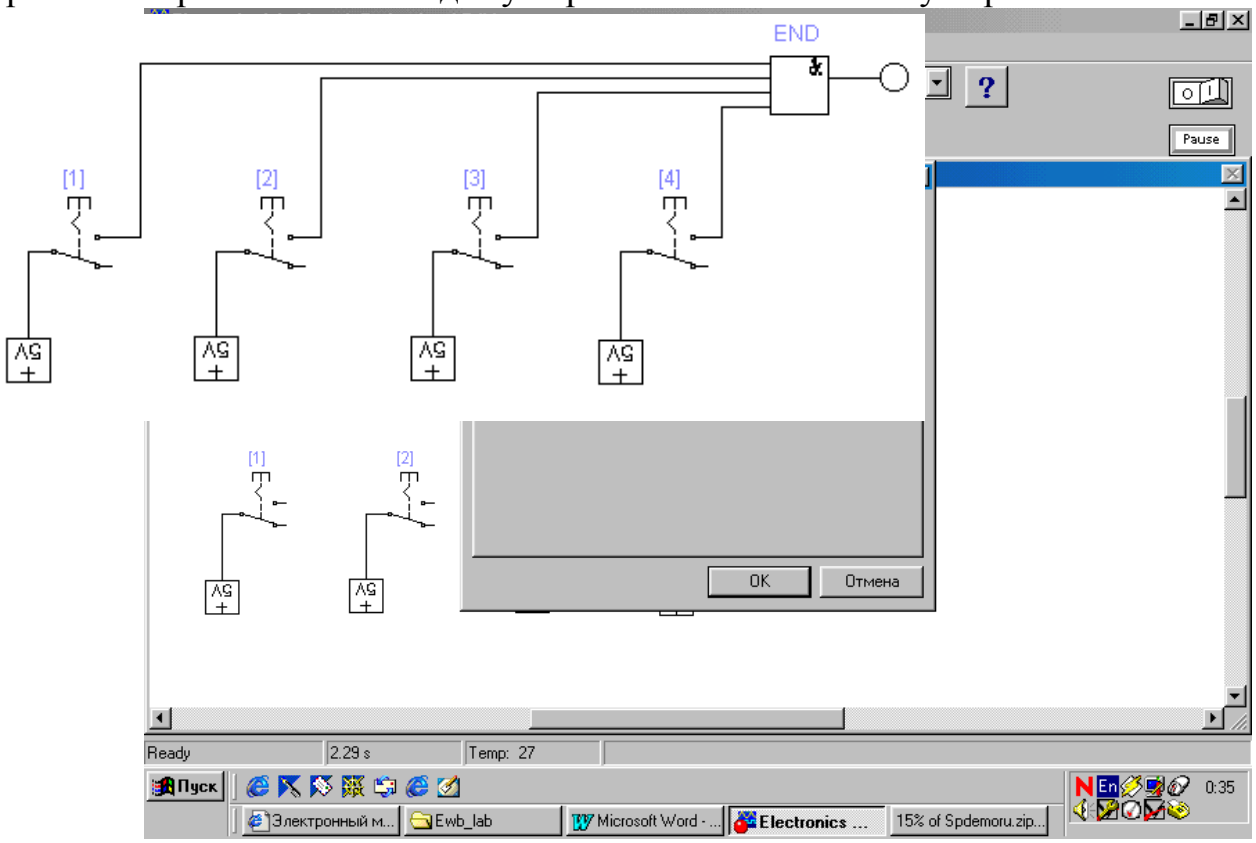
и переключателем



Затем набираем 4 источника и 4 переключателя



При этом присваиваем каждому переключателю клавишу переключения



Затем соединяем входы логической схемы "И" с каждым из переключателей.

Проверка состоит в подаче различных кодовых комбинаций
на вход логической схемы.

На выходе логической схемы "И" появляется логическая 1 (горит светодиод) только при подаче логических 1 (потенциал 5 вольт) на все четыре входа логической схемы "И".

10. Методические указания для выполнения заданий на АССЕМБЛЕРЕ

Материал, необходимый для выполнения контрольных работ

Структурная схема микропроцессора, представлена на рис. 5.1. , включает: устройство управления (УУ), арифметико-логическое устройство (АЛУ), блок преобразования адресов и регистры.

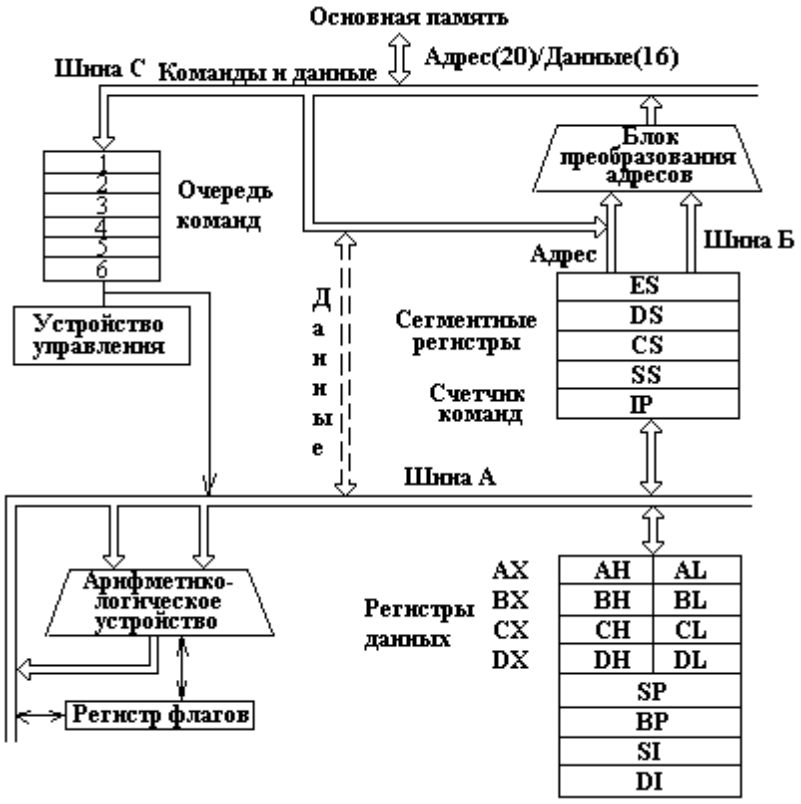


Рис.5.1.Структурная схема микропроцессора

Устройство управления дешифрирует коды команд и формирует необходимые управляющие сигналы. Арифметико-логическое устройство осуществляет необходимые арифметические и логические преобразования данных. В блоке преобразования адресов формируются физические адреса данных, расположенных в основной памяти. Наконец, регистры используются для хранения управляющей информации: адресов и данных.

Всего в состав микропроцессора i8086 входит четырнадцать 16-битовых регистров:

а) четыре регистра общего назначения (регистры данных):

AX - регистр-аккумулятор,

BX - базовый регистр,

CX - счетчик,

DX - регистр-расширитель аккумулятора;

б) три адресных регистра:

SI - регистр индекса источника,
DI - регистр индекса результата,
BP - регистр-указатель базы;

в) три управляющих регистра:

SP - регистр-указатель стека,
IP - регистр-счетчик команд,
регистр флагов;

г) четыре сегментных регистра:

CS - регистр сегмента кодов,
DS - регистр сегмента данных"
ES - регистр дополнительного сегмента данных,
SS - регистр сегмента стека.

Минимальной адресуемой единицей основной памяти ПЭВМ является байт, состоящий из 8 бит. Доступ к байтам основной памяти осуществляется по номерам (номер байта является ого физическим адресом в устройстве памяти).

Для адресации основной памяти в микропроцессоре i8086 предусматриваются 20-битовые адреса, что позволяет работать с основной памятью до 1 Мбайта.

Физический адрес формируется из 16-битового смещения и содержимого 16-битового сегментного регистра, сдвинутого влево на 4 бита (см. рис. 2).

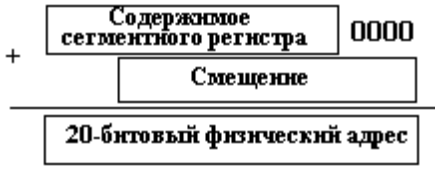


Рис. 5.2. Формирование физического адреса

Для размещения программ и данных в основной памяти выделяются специальные области - сегменты. Адреса этих областей хранятся в специальных сегментных регистрах.

Каждый из четырех сегментных регистров используется для хранения адреса определенного сегмента (см. рис. 3):

- ♦ сегмента кодов, т. е. области программ;
- ♦ сегмента данных, т. е. области размещения данных;
- ♦ дополнительного сегмента данных, используемого некоторыми командами;
- ♦ сегмента стека, т.е. области размещения стека.

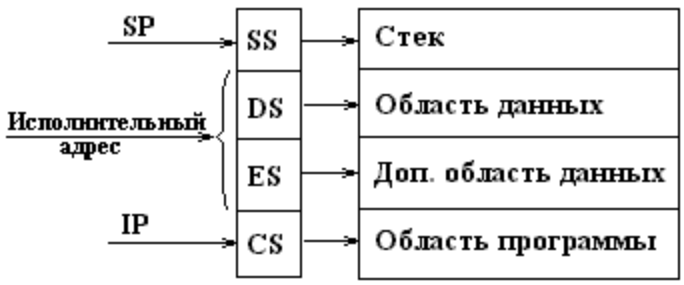


Рис. 5. 3. Сегментные регистры

Стек представляет собой специальным образом организованную область памяти, допускающую последовательную запись элементов данных длиной 2 байта (слово) и чтение их в порядке, обратном порядку записи. Для хранения адреса последнего слова, занесенного в стек, служит регистр-указатель стека **SP** (см. рис. 4, где а - текущее состояние стека, б - запись X, в - чтение X).

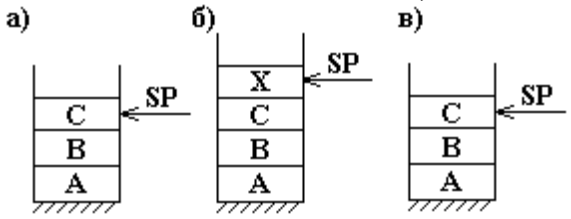


Рис. 4. Организация стека

Стек используется для временного хранения данных и адресов, например при вызове подпрограмм, когда в стек заносится адрес возврата и значения параметров, передаваемых в подпрограмму.

Формат команд микропроцессора позволяет указывать в команде только один операнд, размещенный в основной памяти, т. е. одной командой нельзя, например, сложить содержимое двух ячеек памяти,

Принципиально допускается 8 способов задания смещения (исполнительного адреса) операндов, размещенных в основной памяти:

SI + <индексное смещение>
DI + <индексное смещение>
BP + <индексное смещение>
BX + <индексное смещение>
BP + **SI** + < индексное смещение>
BP + **DI** +< индексное смещение>
BX + **SI** + <индексное смещение>
BX + **DI** + <индексное смещение>

Во всех случаях исполнительный адрес операнда определяется как сумма содержимого указанных регистров и индексного смещения, представляющего собой некоторое число (одно- или двухбайтовое).

Содержимое регистров **CS** и **IP**, в которых хранится базовый адрес сегмента кодов и смещение очередной команды относительно начала сегмента, определяет физический адрес команды, которая должна быть выполнена на следующем шаге.

По указанному адресу из основной памяти считывается команда и пересылается в микропроцессор. Команда длиной от 1 до 8 байт помещается в очередь команд, откуда поступает в устройство управления, где дешифрируется.

Если при выполнении команды требуются данные, расположенные в основной памяти, то специальные полем кода команды определяется способ адресации и вычисляется исполнительный, и затем и физический адрес данных.

Данные, считанные из основной памяти по указанному адресу, пересылаются в регистр данных или в арифметико-логическое устройство и обрабатываются в

соответствии с кодом команды. Результат помещается либо в регистры, либо (в соответствии с командой) в какую-либо область основной памяти.

Если выполненная команда не являлась командой передачи управления, то содержимое регистра **IP** увеличивается на длину выполненной команды, в противном случае в регистр **IP** заносится исполнительный адрес команды, которая должна выполняться следующей.Затем процесс повторяется.

На рис. 5.5. представлен флажковый регистр микропроцессора i8086, в котором в виде однобитовых признаков по принципу ДА - НЕТ (ВКЛЮЧЕНО - ВЫКЛЮЧЕНО) фиксируется информация о результатах выполнения некоторых команд, например арифметических.

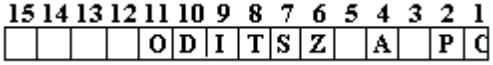


Рис. 5.5. Регистр флагов

O - признак переполнения;
D - признак направления;
I - признак прерывания;
T - признак трассировки;
S - признак знака: 1 - число < 0, 0 - число > 0
Z - признак нуля: 1 - число = 0
A - признак переноса из тетрады;
P - признак четности;
C - признак переноса.

В последующем эта информация может использоваться, например, командами условной передачи управления.

Формат операторов ассемблера

Операторы языка ассемблера ПЭВМ имеют следующий формат:

[<метка> :] <код операции > [<список операндов >1 [<комментарии>].

Запись программы выполняется по свободному формату, т. е. специально не оговариваются правила заполнения каких бы то ни было позиций строки. Точка с запятой в начале строки означает, что данная строка является строкой комментария.

Программа может записываться как заглавными, так и строчными буквами. Метку произвольной длины следует записывать с начала строки и отдалять от кода операции двоеточием, за которым может следовать произвольное количество пробелов (вплоть до конца строки).

Код операции должен отделяться от списка операндов хотя бы одним пробелом. Операнды отделяются один от другого запятой.

Определение полей памяти для размещения данных

Для определения данных в основной памяти и резервирования полей памяти под данные, размещаемые в основной памяти в процессе выполнения программы, используются следующие операторы:

DB - определить однобайтовое поле, **DW** - определить слово (двухбайтовое поле), **DD** - определить двойное слово (четырехбайтовое поле).

Формат команды:

DB

[<имя поля>] **DW** [< количество > **DUP** (]{ <список чисел >}[ )]

DD ?

где <количество >- количество полей памяти указанной длины, которое определяется данной командой (указывается, если определяется не одно поле памяти); ? - используется при резервировании памяти.

Приведем примеры.

1. Записать в байт памяти десятичное число 23 и присвоить этому байту имя a:
   a db 23.
2. Зарезервировать 1 байт памяти: **db ?**
3. Записать в слово памяти шестнадцатеричное число 1234: **dw 1234H.**
4. Определить 31 байт памяти, повторяя последовательность 1, 2, 3, 4, 5, 1, 2, 3, 4,... :

db 31 dup (1,2,3,4,5)

Примечание. При записи слов в память младший байт записывается в поле с младшим адресом. Например, в примере 3, если запись выполнялась по адресу 100, то по адресу 100 будет записано 34H, а по адресу 101 - 12H.

Операнды команд ассемблера

Операнды команд ассемблера могут определяться непосредственно в команде, находиться в регистрах или в основной памяти,

Данные, непосредственно записанные в команде, называются литералами. Так, в команде

mov ah, 3 3 - литерал.

Если операнды команд ассемблера находятся в регистрах, то в соответствующих командах указываются имена регистров (если используемые регистры особо не оговариваются для данной команды. Например, в приведенном выше примере **ah** - имя регистра аккумулятора.

Адресация операндов, расположенных в основной памяти, может быть прямой и косвенной.

При использовании прямой адресации в команде указывается символическое имя поля памяти, содержащего необходимые данные, например:

inc OPND

Здесь **OPND** - символическое имя поля памяти, определенного оператором ассемблера

OPND dw ?

При трансляции программы ассемблер заменит символическое имя на исполнительный адрес указанного поля памяти (смещение относительно начала сегмента) и занесет этот адрес на место индексного смещения. Адресация а этом случае выполняется по схеме: **BP** + <индексное смещение>, но содержимое регистра **BP** при вычислении исполнительного адреса не используется (частный случай).

В отличие от прямого косвенный адрес определяет не местоположение данных в основной памяти, а местоположение компонентов адреса этих данных. В этом случае в команде указываются один или два регистра в соответствии с допустимыми схемами адресации и индексное смещение, которое может задаваться числом или символическим именем. Косвенный адрес заключается в квадратные скобки весь или частично, например:

[OPND +SI]
OPND [SI]
OPND + [SI]
[OPND] +[SI]

Приведенные выше формы записи косвенного адреса интерпретируются одинаково.

При трансляции программы ассемблер определяет используемую схему адресации и соответствующим образом формирует машинную команду, при этом символическое имя заменяется смещением относительно начала сегмента так же, как в случае прямой адресации.

При использовании косвенной адресации по схеме BP + <индексное смещение> индексное смещение не может быть опущено, так как частный случай адресации по данной схеме с нулевой длиной индексного смещения используется для организации прямой адресации. Следовательно, при отсутствии индексного смещения в команде следует указывать нулевое индексное смещение, т.е. [**BP** + **0**] .

Приведем два примера: [**a** + **bx**] и [**bp**]+[**si] +6.**

В первом случае исполнительный адрес операнда определяется суммой содержимого регистра **bx** и индексного смещения, заданного символическим именем "**a**", а во втором - суммой содержимого регистров **bp**, **si** и индексного смещения, равного 6.

Длина операнда может определяться:

а) кодом команды - в том случае, если используемая команда обрабатывает данные определенной длины, что специально оговаривается;

б) объемом регистров, используемых для хранения операндов (1 или 2 байта);

в) специальными указателями **byte ptr** (1 байт) и **word ptr** (2 байта), которые используются в тех случаях, когда длину операнда нельзя установить другим способом. Например,

mov byte ptr x, 255

т. е. операнд пересылается в поле с именем "**x**" и имеет длину I байт.

Команды пересылки / преобразования данных

Команда пересылки данных

MOV <адрес приемника> ,< адрес источника>

используется для пересылки данных длиной I или 2 байта из регистра в регистр, из регистра в основную память, из основной памяти в регистр, а также для записи в регистр или основную память данных, непосредственно записанных в команде. Все возможные пересылки представлены на рис. 6.

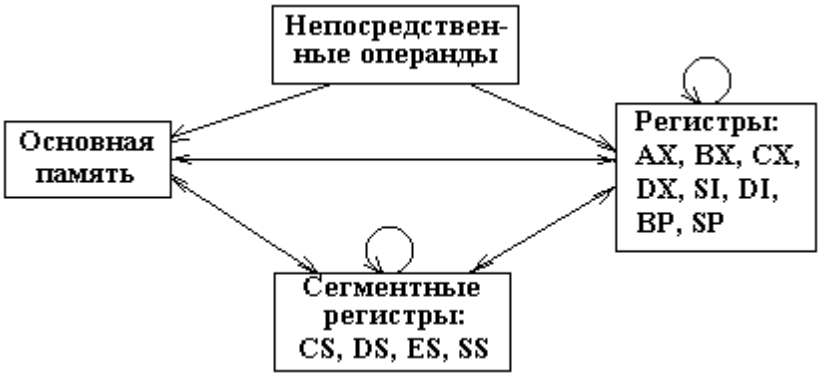


Рис. 5. 6. Команды пересылки

Примеры:

а) **mov ax, bx**- пересылка содержимого регистра **bx** в регистр **ax**;

б) **mov cx, exword** - пересылка 2 байт, расположенных в поле **exword**, из основной памяти в регистр **cx**;

в) **mov si, 1000** - запись числа 1000 в регистр **si**;

г) **mov word ptr [di+515], 4** - запись числа 4 длиной 2 байта в основную память по адресу **[di+515]**.

Для загрузки "прямого" адреса в сегментный регистр используются две команды пересылки:

mov ax, code
mov ds, ax

Команда обмена данных.

XCHG <операнд 1> , <операнд 2>

организует обмен содержимого двух регистров (кроме сегментных) или регистра и поля основной памяти. Например:

xchg bx, cx - обмен содержимого регистров **bx** и **cx.**

Команда загрузки исполнительного адреса

LEA < операнд l > , < операнд 2 >

вычисляет исполнительный адрес второго операнда и помещает его в поле, на которое указывает первый операнд. Приведем примеры:

а) **lea bx, exword** - в регистр **bx** загружается исполнительный адрес **exword**;

б) **lea bx, [di+10]** - в регистр **bx** загружается адрес l0-го байта относительно точки, на которую указывает адрес в регистре **di**.

Команды загрузки указателя

LDS < регистр > ,<операнд 2>
LES < регистр > ,< операнд 2>

Команда **LDS** загружает в регистры **DS** :< регистр> указатель (< адрес сегмента > : < исполнительный адрес >), расположенный по адресу, указанному во втором операнде.

Команда **LES** загружает указатель по адресу, расположенному во втором операнде, в регистры **ES**:< регистр> .

Например:

lds si, exword

т.е. слово (2 байта) по адресу **exword** загружается в **si**, а по адресу **exword**+ **2 -** в **ds**.

Команда записи в стек

PUSH < операнд>

организует запись в стек слова, адрес которого указан в операнде. Например;

push dx - запомнить содержимое регистра **dx** в стеке.

Команда восстановления из стека

POP < операнд>

организует чтение из стека последнего слова и помещает его по адресу, указанному во втором операнде. Например:

pop dx - восстановить содержимое регистра **dx** из стека.

Команды сложения

ADD <операнд 1> , <операнд 2>

ADC <операнд 1> , <операнд 2>

устанавливают флаги четности, знака результата, наличия переноса, наличия переполнения.

Ilo команде **ADD** выполняется сложение двух операндов. Результат записывается по адресу первого операнда. По команде **ADC** также выполнятся сложение двух операндов, но к ним добавляется еще значение, записанное в бите переноса, установленном предыдущей командой сложения.

На рис. 5.7. показаны возможные способы размещения слагаемых, где а - операнды - слова, б - операнды - байты.

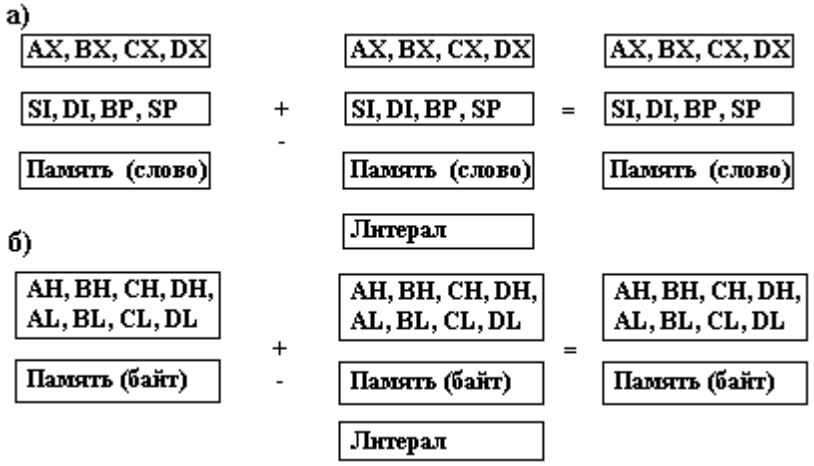


Рис. 5.7. Способы размещения слаемых

Приведем пример сложения двух 32-разрядных чисел:

```
mov   ax,value1
add   value2,ax
mov   ax,value1+2
adc   value2+2,ax
```

Исходные числа находится в основной памяти по адресам **value1** и **value2**, а результат записывается по адресу **value1.**

Команды вычитания

SUB <уменьшаемое-результат> , <вычитаемое>
SBB <уменьшаемое-результат>, <вычитаемое>

устанавливают флаги четности, знака результата, наличия заема, наличия переполнения.

При выполнении операции по команде **SUB** заем не учитывается, а по команде **SBB** - учитывается. Ограничения на местоположение операндов такие же, как и у команды сложения.

Команда изменения знака

NEG <операнд>

знак операнда изменяется на противоположный.

10. Команда добавления единицы.

INC <операнд>

значение операнда увеличивается на единицу.

Команда вычитания единицы

DEC <операнд>

значение операнда уменьшается на единицу.

Команда сравнения

CMP <операнд 1> , < операнд 2>

выполняется операция вычитания без записи результата и устанавливаются признаки во флажковом регистре.

Команды умножения

MUL <операнд>
IMUL <операнд>

устанавливают флаги наличия переноса или переполнения.

По команде **MUL** числа перемножаются без учета, и по команде - **IMUL** с учетом знака (в дополнительном коде).

На рис. 5.8. (где а - операнды - слова, б - операнды - байты) приведены возможные способы размещения сомножителей и результата (один из сомножителей всегда расположен в регистре-аккумуляторе.

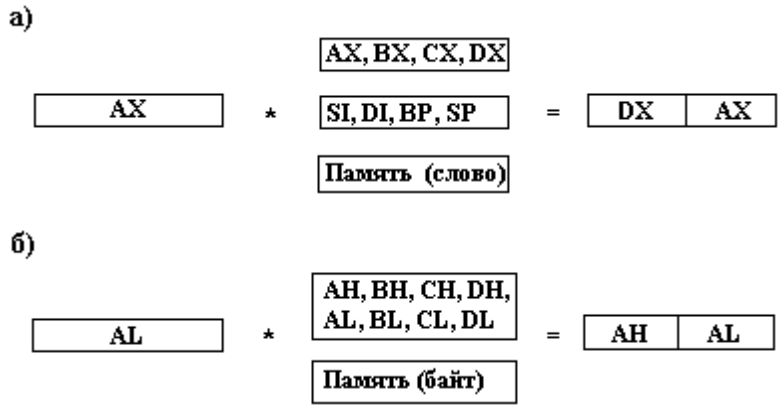


Рис. 5.8. Способы размещения сомножителей

Пример:

imul word ptr c

Здесь содержимое основной памяти по адресу "**c"** длиной слово умножается на содержимое регистра **ax**. Младшая часть результата операции записывается в регистр **ax**, а старшая часть - и регистр **dx.**

Команда деления

DIV <операнд-делитель>

IDIV <операнд-делитель>

По команде **DIV** операция деления выполняется без учета, а по команде **IDIV** - с учетом знака (в дополнительном коде).

Команды передачи управления

Команда безусловного перехода

JMP <адрес перехода>

имеет три модификации в зависимости от длины ее адресной части:

short - при переходе по адресу, который находится на расстоянии -128...127 байт относительно адреса данной команды (длина адресной части 1 байт);

near ptr - при переходе по адресу, который находится на расстоянии 32 Кбайта (-32768...32767 байт) относительно адреса данной команды (длина адресной части 2байта);

far ptr - при переходе по адресу, который находится на расстоянии превышающем 32 Кбайта (длина адресной части 4 байта).

При указании перехода к командам, предшествующим команде перехода, ассемблер сам определяет расстояние до метки перехода и строит адрес нужной длины. При указании перехода к последующим частям программы необходимо ставить указатели **short**, **near ptr** и **far ptr**.

В качестве адреса команды перехода используются метки трех видов:

а) < имя > : **nop** (**nop** - команда "нет операции");

б)< имя> **label near** (для внутрисегментных переходов);

в) <имя> **label far** (для внесегментных переходов).

Примеры:

а) **jmp short b** - переход по адресу **b**;

б) **jmp [bx]** - переход по адресу в регистре **bx** (адрес определяется косвенно);

в) **a : nop** - описание метки перехода "**a**";

г) **b label near** - описание метки перехода "**b**".

Команды условного перехода

<мнемоническая команда> <адрес перехода>

Мнемоника команд условного перехода:

JZ - переход по "ноль";

JE - переход по "равно";

JNZ - переход по "не нуль";

JNE - переход по "не равно";

JL - переход по "меньше";

JNG, JLE - переход по "меньше или равно ";

JG - переход по "больше";

JNL, JGE - переход по "больше или равно ";

JA - переход по "выше" (беззнаковое больше);

JNA, JBE - переход по "не выше"(беззнаковое не больше);

JB - переход по "ниже" (беззнаковое меньше);

JNB, JAE - переход по"не ниже" (беззнаковое не меньше).

Все команды имеют однобайтовое поле адреса, следовательно, смешение не должно превышать -128...127 байт. Если смещение выходит за указанные пределы, то используется специальный прием:

вместо	программируется
jz zero	**jnz continue**
	jmp zero
	continue: ...

Команды организации циклической обработки

В качестве счетчика цикла во всех командах циклической обработки используется содержимое регистра **cx**.

1) Команда организации цикла.

LOOP < адрес перехода >

при каждом выполнении уменьшает содержимое регистра **cx** на единицу и передает управление по указанному адресу, если **cx** не равно 0:

```
        mov   cx, loop_count        ; загрузка счетчика
begin_loop:
    ; ... тело цикла ...
        loop  begin_loop
```

2) Команда перехода по обнуленному счетчику.

JCXZ <адрес перехода>

передает управление по указанному адресу, если содержимое регистра **cx** равно 0. Например:

```
        mov    cx,  loop_count          ;  загрузка счетчика
        jcxz    end_of_loop             ;  проверка счетчика
begin_loop:
    ;  ...  тело цикла ...
            loop    begin_loop
end_of_loop:    ...
```

3) Команды организации цикла с условием.

LOOPE <адрес перехода>

LOOPNE <адрес перехода>

уменьшают содержимое на единицу и передают управление по указанному адресу при условии, что содержимое **cx** отлично от нуля, но **LOOPE** дополнительно требует наличия признака "равно", а **LOOPNE** - "не равно", формируемых командами сравнения. Например:

```
        mov    cx,  loop_count          ;  загрузка счетчика
        jcxz    end_of_loop             ;  проверка счетчика
begin_loop:
    ;  ...  тело цикла ...
            cmp     al, 100                 ; проверка содержимого al
            loopne  begin_loop    ; возврат в цикл, если cx≠0 и al≠100
end_of_loop:    ...
```

Команды вызова подпрограмм

1) Команда вызова процедуры.

CALL <адрес процедуры>

осуществляет передачу управления по указанному адресу, предварительно записав в стек адрес возврата.

При указании адреса процедуры так же как и при указании адреса перехода в командах безусловного перехода, возникает необходимость определить удаленности процедуры от места вызова:

а) если процедура удалена не более чем на -128...127 байт, то специальных указаний не требуется;

б) если процедура удалена в пределах 32 кбант, то перед адресом по процедуры необходимо указать **near ptr**,

в) если процедура подпрограмма удалена более, чем на 32 кбайта, то перед адресом процедуры необходимо записать **far ptr.**

Пример:

call near ptr p - вызов подпрограммы "**p**".

Текст процедуры должен быть оформлен в виде:

< имя процедуры> **proc** < указатель удаленности>

... тело процедуры ...

<имя процедуры> **end**

Здесь указатель удаленности также служит дли определения длины адресов, используемых при обращении к процедуре: **near** - при использовании двухбайтовых адресов, **far** - при использовании четырехбайтовых адресов.

2) Команда возврата управления.

RET [<число>]

извлекает из стека адрес возврата и передает управление по указанному адресу.

Если в команде указано значение счетчика, то после восстановления адреса возврата указанное число добавляется к содержимому регистра-указателя стека. Последний вариант команды позволяет удалить из стека параметры, передаваемые в процедуру через стек.

флажка нуля равно нулю.

Команды манипулирования битами

Логические команды

NOT <операнд> - логическое НЕ;

AND <операнд 1>, <операнд 2> - лигическое И;

OR <операнд 1>, <операнд 2> - логическое ИЛИ;

XOR <операнд 1>, <операнд 2> - исключающее ИЛИ;

TEST <операнд 1>, <операнд 2> - И без записи результата.

Операнды байты или слова.

Пример. Выделить из числа в AL первый бит:

```
and   al,  10000000B
```

Команды сдвига

<код операции> <операнд>, <счетчик>

Счетчик записывается в регистр CL. Если счетчик равен 1, то его можно записать в команду.

Коды команд сдвига:

SAL - сдвиг влево арифметический;

SHL - сдвиг влево логический;

SAR - сдвиг вправо арифметический;

SHR - сдвиг вправо логический;

ROL - сдвиг влево циклический;

ROR - сдвиг вправо циклический;

RCL - сдвиг циклический влево с флагом переноса;

RCR - сдвиг циклический вправо с флагом переноса.

Пример. Умножить число в AX на 10:

```
mov   bx, ax
shl   ax,  1
shl   ax,  1
add   ax, bx
shl   ax,  1
```

Команды ввода - вывода

Обмен данными с внешней средой осуществляемся с помощью следующих команд:

IN <регистр>, <порт> (ввод из порта в регистр),

IN <регистр >, **DX** (ввод из порта, номер которого указан в регистре **DX** в регистр);

OUT <порт>, <регистр> (вывод содержимого регистра в порт),

OUT DX, <регистр> (вывод содержимого регистра в порт, номер которого указан в регистре **DX**).

В качестве регистра можно указать **AL** или **AX** (соответственно будет обрабатываться байт или два байта). Порт отождествляется с некоторым внешним устройством (0...255).

Однако при организации ввода - вывода помимо самой операции необходимо осуществить ряд дополнительных действий, например, проверить готовность устройства. В связи с этим для типовых устройств разработаны стандартные программы организации ввода - вывода, которые вызываются по команде прерывания **int 21h**.

В таблице 1 приведен перечень основные функции, реализуемые подпрограммами ввода - вывода, и их коды. Код функции должен передаваться в подпрограмму в регистре **AH.**

Таблица 1

Код функции	Функция
01	Ввод с клавиатуры одного символа в регистр AL (с проверкой на Ctrl-Break, с ожиданием, с эхо)
02	Вывод одного символа на экран дисплея из регистра DL (с проверкой на Ctrl-Break)
06	Непосредственный ввод - вывод: ввод в регистр AL (без ожидания, без эхо, без проверки на Ctrl-Break, регистр DL должен содержать 0FFH), вывод из регистра DL (без проверки на Ctrl-Break).
07	Ввод в регистр AL (без проверки на Ctrl-Break, с ожиданием, без эхо)
08	Ввод в регистр AL (с проверкой на Ctrl-Break, с ожиданием, без эхо)
09	Вывод строки на экран (DS:DX - адрес строки, которая должна завершаться символом "$")
10(0Ah)	Ввод строки в буфер (DS:DX - адрес буфера, первый байт которого должен содержать размер буфера, после ввода - второй байт содержит количество введенных символов)
11(0Bh	Чтение состояния клавиатуры (если буфер пуст, то AL=0, иначе

)	AL=0FFh)

Примеры:

а)
```
mov   ah, 1   ; номер функции
int   21h     ;  ввод символа: символ в AL
```

б)
```
mov   ah, 2   ; номер функции
mov   dl, 'A'
int   21h     ;  вывод символа из DL
```

в)
```
lea   dx, STRING  ; адрес буфера ввода
mov   ah, 0Ah   ; номер функции
int   21h   ; ввод строки:  во втором байте буфера - количество
...         ; введенных символов, далее в буфере символы
STRING  db   50, 50 dup (?)
```

г)
```
lea   dx, MSG  ; адрес выводимой строки
mov   ah, 9   ; номер функции
int   21h     ; вывод строки
...
MSG   db   'Пример вывода', 13, 10, '$'
```

Структура программы на ассемблере

Структура программы на языке ассемблера выглядит следующим образом (**.exe**):

```
                    TITLE  <имя программы>
<имя сегмента стека>   SEGMENT  STACK
                        DB   3000  DUB   (?)
<имя сегмента стека>   ENDS
<имя сегмента данных > SEGMENT
                  <данные>
<имя сегмента данных > ENDS
<имя сегмента кодов>   SEGMENT
                    ASSUME CS: <имя сегмента кодов>,  DS:<имя сегмента данных>
                    EXTRN  <имя внешней процедуры >:<тип>
                    PUBLIC <имя внутренней процедуры>
<имя основной процедуры> PROC FAR
                       PUSH  DS
                       MOV   AX, 0
                       PUSH  AX
                       MOV   AX, <имя сегмента данных>
                       MOV   DS, AX
                   <тело процедуры>
                       RET
```

<имя основной процедуры> ENDP
<имя внутренней процедуры> PROC NEAR
<тело внутренней процедуры>
<имя внутренней процедуры> ENDP
<имя сегмента кодов> ENDS
END **<**имя основной процедуры **>**

Первая строка программы - заголовок, состоящий из служебного слова **TITLE** и имени программы.

Текст программы состоит из отдельных сегментов, каждый из которых начинается оператором **SEGMENT** и завершается оператором **ENDS**:

<имя сегмента > **SEGMENT**
... тело сегмента ...
<имя сегмента > **ENDS**

Сегмент стека содержит специальный описатель **STACK.**

Сегмент кодов, в котором располагается текст программы, начинается псевдокомандой **ASSUME**, которая сообщает ассемблеру, какой сегментный регистр должен использоваться для адресации каждого сегмента.

За псевдокомандой **ASSUME** может следовать описание используемых программой внешних подпрограмм:

EXTRN <имя внешней процедуры >:<тип **near** или **far**>
PUBLIC <имя внутренней процедуры**>**

Сегмент кодов всегда адресуется сегментным регистром **CS.** Значение этого регистра операционная система устанавливает автоматически. Значения сегментного регистр **DS** загружается программистом:

MOV AX, <имя сегмента данных**>**
MOV DS, AX

При необходимости также загружается регистр **ES**:

MOV ES, AX

Команды

PUSH DS
MOV AX, 0
PUSH AX

организуют возможность возврата управления в MS DOS командой **RET.**

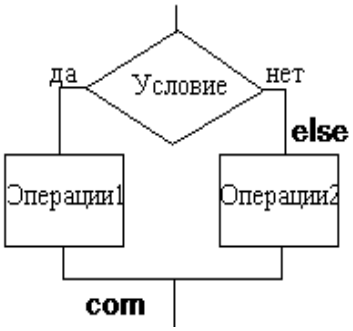
В этом случае в стек в качестве адреса возврата помещается адрес префиксной области программы, первые два байта которой содержат команду **INT 20H** возврата управления операционной системе.

Основные приемы программирования на ассемблере

Ассемблер, являясь языком низкого уровня не содержит операторов ветвления, циклов, не поддерживает автоматического формирования адресов для структур данных, не обеспечивает автоматического выполнения преобразований при вводе-выводе данных. Все перечисленные операции программируются "вручную" с использованием имеющихся команд ассемблера.

Программирование ветвлений

Ветвления программируются с использованием команд условной и безусловной передачи управления.




```
cmp    ...
j<условие>     ELSE
<операции 1>
jmp    COM
       ELSE:    <операции 2>
       COM:     <продолжение>
```

Пример.

Написать процедуру вычисления **X=max(A,B)**:

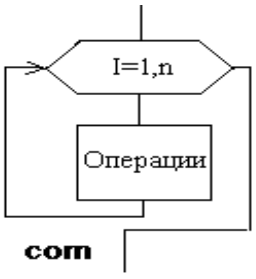
```
max     proc    near
        mov     ax, A
        cmp     ax, B      ; сравнение A и B
        jl      LESS       ; переход по меньше
        mov     X, ax
        jmp     CONTINUE   ;  переход на конец ветвления
LESS:   mov     ax,  B
        mov     X, ax
CONTINUE: ret
max     endp
```

Программирование циклических процессов

Программирование циклических процессов осуществляется с использованием либо команд переходов, либо - в случае счетных циклов - с использованием команд организации циклов.

а) программирование итерационных циклов (цикл-пока):


```
CYCL:   cmp  ...
        jne    COM
      <операции>
        jmp    CYCL
COM:    ....
        :
```




```
        mov    cx, N
CYCL:   <операции>
        loop   CYCL
```

com: **.....**

Пример.

Процедура суммирования чисел от 1 до 10, используя счетный цикл.

```
sum       proc     near
          mov      ax, 0     ;  обнуление суммы
          mov      bx, 1     ;  первое слагаемое
          mov      cx, 10    ;  загрузка счетчика
CYCL:     add      ax, bx    ;  суммирование
          inc      bx        ;  следующее число
          loop     CYCL      ;  возврат в цикл
continue: ret                ; выход,  сумма - в ax
sum       endp
```

11. Литература

1. Орлов С. А., Цилькер Б. Я. Организация ЭВМ и систем: Учебник для вузов. 2-е изд. — СПб.: Питер, 2011. — 688 с.
2. Максимов Н.В. Архитектура ЭВМ и вычислительных систем: Учебник / Н.В. Максимов, Т.Л. Партыка, И.И. Попов. - 5-е изд., перераб. и доп. - М.: Форум:НИЦ ИНФРА-М, 2013. - 512 с. - http://znanium.com/bookread2.php?book=405818
3. Колдаев В.Д. Архитектура ЭВМ: Учебное пособие / В.Д. Колдаев, С.А. Лупин. - М.: ИД ФОРУМ: НИЦ ИНФРА-М, 2014. - 384 с. - http://znanium.com/bookread2.php?book=424016
4. Калабеков Б. А. Микропроцессоры и их применение в системах передачи и обработки сигналов. – М. : Радио и связь, 1988.
5. Майоров С. А. Новиков Г. И. Структура цифровых вычислительных машин. – СПб. : Машиностроение, 1970.
6. Каган Б. М. Электронные вычислительные машины и системы. – М. : Энергоатомиздат, 1985.
7. Стрыгин В. В., Щарев Л. С. Основы вычислительной микропроцессорной техники и программирования. – М. : Высшая школа, 1989.
8. Замятин Н. В. Нечеткая логика и нейронные сети: учебное пособие. - Томск: Эль Контент, 2014. - 146 с.
9. Баранов С. И. Синтез микропрограммных автоматов. – Л. : Энергия, 1979.
10. Метлицкий Е. А., Каверзнев В. В. Системы параллельной памяти. Теория, проектирование, применение. – Л., 1989.