Главное управление образования Гродненского облисполкома

Управление профессионального образования и профориентации
ГУО «Гродненский областной институт развития образования»

Учреждение образования
«Гродненский государственный колледж техники, технологий и дизайна»

АРИФМЕТИКО-ЛОГИЧЕСКИЕ ОСНОВЫ ВЫЧИСЛТЕЛЬНОЙ ТЕХНИКИ

Лабораторный практикум

Специальность: 2-40 01 01 Программное обеспечение информационных технологий

Гродно
2022

СОДЕРЖАНИЕ

АННОТАЦИЯ

В современном мире вычислительная техника занимает ведущую роль в различных сферах человеческой деятельности: производстве, транспорте, логистике, образовании и медицине. Умение пользоваться вычислительной техникой становится базовым, а вместе с тем возрастает важность знаний об устройстве вычислительных систем и их компонентах.

В рамках изучения учебного предмета «Арифметико-логические основы вычислительной техники» формируется понимание основополагающих вопросов построения вычислительных систем, предусматривается изучение принципов организации вычислительного процесса, конструктивных особенностей, технических и эксплуатационных характеристик современных вычислительных средств.

Предмет посвящён изучению арифметических основ вычислительной техники, алгебры логики, схемотехники ПК.

Цель изучения – освоение учащимися следующих областей знаний: арифметических основ вычислительной техники на основе двоичной арифметики; логических основ вычислительной техники на базе изучения алгебры логики; схемотехнических основ и архитектурной организации ПК.

В ходе выполнения лабораторных работ по учебному предмету «Арифметико-логические основы вычислительной техники» рассматриваются следующие вопросы:

арифметические и логические основы вычислительной техники;

принципы построения и функционирования устройств вычислительной техники;

функциональный состав и основы организации микропроцессора;

система команд управления работой микропроцессора;

организация и методы адресации памяти;

взаимодействие аппаратного и программного обеспечения вычислительной техники;

перевод чисел из одной системы счисления в другую и арифметические действия в различных системах счисления;

представление чисел в машинном формате;

оптимизация логических выражений и синтез логических схем;

анализ принципов работы элементов, узлов и устройств вычислительной техники;

приемы составления команды и программного кода на машинно-ориентированном языке программирования.

Лабораторные работы служат связующим звеном между теорией и практикой. Они помогают углубить и закрепить теоретические знания, изучить на практике схемы работы логических устройств.

Лабораторный практикум по предмету «Арифметико-логические основы вычислительной техники» разработан с учетом специфики подготовки учащихся в рамках среднего специального образования по специальности: 2-40 01 01 Программное обеспечение информационных технологий, квалификация специалиста: техник-программист.

При проведении лабораторных работ учащиеся соблюдают установленные правила безопасности. В начале занятия преподаватель сообщает тему и цель работы. Учащиеся в ходе занятия повторяют и закрепляют ранее полученные теоретические знания, выполняют различные задания, формирующие умения практического характера. Преподаватель контролирует ход работы, отвечает на вопросы учащихся и при необходимости корректирует возникающие ошибки. В конце работы учащихся отвечают на контрольные вопросы. В данных вопросах выделены основные аспекты рассматриваемой темы, которые должны быть усвоены учащимся.

В заключительной части выполнения лабораторных работ составляется отчет. Окончательное оформление отчета становится домашним заданием учащихся.

ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Арифметико – логическое устройство (***АЛУ***) – блок процессора, предназначенный для выполнения логических и математических операций над двоичными числами.

Демультиплексор (DMX или DMS) – комбинационное устройство, преобразующее последовательный сигнал в параллельный и выполняющее функцию, обратную мультиплексору.

Дешифратор (DC) – комбинационное устройство, позволяющее распознавать числа, представленные позиционным n-разрядным кодом.

Единая система конструкторской документации (системаЕСКД) – комплекс межгосударственных стандартов, устанавливающих взаимосвязанные правила, требования и нормы по разработке, оформлению и обращению конструкторской документации, разрабатываемой и применяемой на всех стадиях жизненного цикла изделия.

Команда – это код, определяющий операцию вычислительной машины и данные, участвующие в операции.

Командный цикл – интервал, на протяжении которого осуществляется одно обращение микропроцессора к памяти или к внешнему устройству.

Компаратор – логическое устройство с двумя входами, на которые подаются два разных двоичных слова равной в битах длины, и тремя двоичными выходами, на которые выдается признак сравнения входных слов (первое слово меньше второго, больше второго или слова равны).

Логический элемент – это простейшая структурная единица, выполняющая определённые логические операции над двоичными переменными.

Метод адресации – способы указания на определённую ячейку (ячейки) памяти ЭВМ процессору с целью записи, чтения данных или передачи управления.

Микропроцессор (МП) – процессор (устройство, отвечающее за выполнение арифметических, логических операций и операций управления, записанных в

машинном коде), реализованный в виде одной микросхемы или комплекта из нескольких специализированных микросхем.

Мультиплексор (MUX или MS) – комбинационное устройство, обеспечивающее передачу в желаемом порядке цифровой информации, поступающей по нескольким входам на один выход.

Оперативные запоминающие устройства (ОЗУ или RAM) – память с произвольной выборкой, которая служит для хранения выполняемой программы и оперативных данных.

Подпрограмма именованная или иным образом идентифицированная часть компьютерной программы, содержащая описание определённого набора действий.

Полусумматор – логическое комбинационное устройство, имеющее два входа и два выхода. Позволяет вычислять сумму $A+B$, где A и B –это разряды (биты) обычно двоичного числа, при этом результатом будут два бита S и C, где S – это бит суммы по модулю 2, а C – битпереноса.

Постоянные запоминающие устройства (ПЗУ или ROM) –устройства, предназначенные для хранения постоянных данных и служебных программ.

Регистр – последовательное устройство, предназначенное для приема, хранения и преобразования двоичной информации.

Система команд – это строго заданный конечный набор команд, которые может выполнить исполнитель.

Система счисления – символический метод записи чисел, представление чисел с помощью письменных знаков.

Стек – абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. lastin – firstout, «последним пришёл – первым вышел»).

Структура команды – этологическое понятие, которое определяется составом, назначением и расположением полей в команде.

Сумматор – логическое комбинационное устройство, предназначенное для выполнения операции арифметического сложения чисел, представленных в виде двоичных кодов.

Счетчик – последовательное цифровое устройство, предназначенное для счета входных импульсов, фиксации их числа в двоичном коде и выполнения микрооперации счета по изменению значения числа в счетчике на единицу.

Триггер – устройством, которое находится в одном из двух устойчивых состояний и скачкообразно переходит из одного состояния в другое под воздействием внешнего управляющего сигнала.

Условно графическое обозначение (УГО) – аббревиатура условного графического обозначения на принципиальной электрической схеме.

Формат команд – это структура команды с разметкой номеров разрядов (битов), определяющих границы отдельных полей команды, или с указанием числа битов в определенных полях.

Шифратор (CD) – комбинационное устройство, выполняющее функции, обратные дешифратору.

ТРЕБОВАНИЯ ПО ОХРАНЕ ТРУДА ПРИ ПРОВЕДЕНИИ ЛАБОРАТОРНЫХ РАБОТ

При работе в учебном кабинете обучающиеся должны неукоснительно соблюдать правила внутреннего распорядка для обучающихся учреждений образования.

При разработке правил безопасного поведения при организации образовательного процесса по учебному предмету «Арифметико-логические основы вычислительной техники» определяются правила безопасного поведения при работе с видеотерминалом, персональной электронно-вычислительной машиной.

К работе в компьютерном классе допускаются обучающиеся, ознакомленные с правилами безопасного поведения в компьютерном классе и не имеющие противопоказаний по состоянию здоровья.

При работе в компьютерном классе обучающиеся должны соблюдать правила поведения, расписание учебных занятий, установленные режимы труда и отдыха.

При работе в компьютерном классе возможно воздействие на обучающихся следующих опасных и вредных производственных факторов: неблагоприятное воздействие на организм человека неонизирующих электромагнитных излучений видеотерминалов; неблагоприятное воздействие на зрение визуальных эргономических параметров видеотерминалов, выходящих за пределы оптимального диапазона; неправильный подбор размеров ученической мебели; недостаточная освещенность в компьютерном классе; поражение электрическим током при неисправном электрооборудовании кабинета.

При работе в компьютерном классе педагогическому работнику и обучающимся необходимо соблюдать правила пожарной безопасности, знать места расположения первичных средств пожаротушения. Компьютерный класс должен быть оснащен углекислотным огнетушителем.

При неисправности оборудования обучающимся необходимо прекратить работу и сообщить об этом педагогическому работнику.

В процессе работы с видеотерминалами обучающиеся должны соблюдать порядок проведения работ, правила личной гигиены, содержать в чистоте рабочий стол.

Обучающимся не разрешается самостоятельно включать компьютеры или запускать компьютерные программы, трогать разъемы соединительных кабелей.

Работа в компьютерном классе разрешается только в присутствии педагогического работника. Перед началом работы педагогический работник должен:

- включить полностью освещение в кабинете, убедиться в исправной работе светильников. Уровень искусственной освещенности от системы общего освещения должен составлять не менее 400 люкс;
- убедиться в исправности электрооборудования кабинета: светильники должны быть надежно подвешены к потолку и иметь светорассеивающую арматуру; коммутационные коробки должны быть закрыты крышками; корпуса и крышки выключат елей и розеток не должны иметь трещин и сколов, а также оголенных контактов;
- убедиться в правильной расстановке мебели в компьютерном классе: расстояние между рабочими столами с видеомониторами (в направлении тыла поверхности одного видеомонитора и экрана другого видеомонитора) должно быть не менее2,0м, а расстояние между боковыми поверхностями видеомониторов – не менее 1,2 м;
- тщательно проветрить компьютерный класс и убедиться, что вне зависимости от периода года температура воздуха в пределах +19 – +21 °С, относительная влажность воздуха – 30–60 процентов, скорость движения воздуха – не более 0,1 м/с;
- убедиться в наличии защитного заземления оборудования, а также защитных экранов видеотерминалов; включить персональный компьютер

(далее – ПК) и проверить стабильность и четкость изображения на экранах. Изображение на экранах видеотерминалов должно быть стабильным, ясным и предельно четким, не иметь мерцаний символов и фона, на экранах не должно быть бликов и отражений светильников, окон и окружающих предметов.

Обучающийся должен:

- осмотреть и привести в порядок рабочий стол, освободить его от посторонних предметов;
- проверить правильность установки стола, стула, положения оборудования, угла наклона экрана, положения клавиатуры;
- при нарушении целостности корпуса компьютера, монитора, клавиатуры, мыши, любой неисправности оборудования сообщить педагогическому работнику.

Во время работы недопустимы занятия за одним видеотерминалом двух и более человек.

Меловая доска, как правило, в компьютерном классе для написания информации не используется.

Все используемые в компьютерном классе демонстрационные электрические приборы должны быть исправными и иметь заземление или зануление.

Обучающимся не допускается:

- включать ПК без разрешения педагогического работника;
- дотрагиваться до экрана монитора и вращать монитор;
- работать с ПК при снятом корпусе;
- во избежание внутреннего перегрева и выхода ПК из строя закрывать во время работы вентиляционные отверстия посторонними предметами или чехлами;
- вскрывать корпус монитора, системного блока;
- разбирать монитор, системный блок, клавиатуру, мышь;

- переключать силовые питающие кабели проводов связи с периферийными устройствами на задней крышке корпуса системного блока и монитора,
- отключать штепсельные разъемы, переключать разъемы интерфейсных кабелей периферийных устройств при включенном питании;
- производить отключение питания во время выполнения активной задачи;
- оказывать механические усилия (наступать ногами, дергать силовые питающие кабели и тянуть за провода связи с периферийными устройствами);
- ударять по клавишам клавиатуры;
- прикасаться к задней панели системного блока;
- допускать попадание влаги на поверхность системного блока, монитора, клавиатуры;
- загромождать верхние панели устройств бумагами и посторонними предметами;
- передвигать столы с оборудованием, переставлять оборудование на столах.

При поражении электрическим током обучающегося или при получении им травмы педагогический работник должен немедленно полностью отключить от питающей сети ПК (в случае поражения электрическим током), оказать первую помощь пострадавшему, вызвать медицинского работника учреждения образования или скорую медицинскую помощь по телефону 103, сообщить о несчастном случае руководителю учреждения общего среднего образования.

При возникновении пожара педагогический работник должен немедленно эвакуировать обучающихся из здания, сообщить о пожаре по телефону101, администрации учреждения образования и приступить к тушению очага возгорания с помощью первичных средств пожаротушения.

Обучающийся должен:

- в случае появления неисправности в работе ПК выключить его и сообщить об этом педагогическому работнику;
- при ухудшении самочувствия, появлении головной боли, головокружения прекратить работу и сообщить об этом педагогическому работнику;
- при возникновении нестандартной ситуации сохранять спокойствие и неукоснительно выполнять указания педагогического работника.

По окончании работы с разрешения педагогического работника обучающиеся должны закрыть активные задачи, выключить ПК и привести в порядок рабочий стол.

Педагогический работник должен тщательно проветрить учебный кабинет, закрыть окна, фрамуги, выключить свет.

РАЗДЕЛ V. Комбинационные цифровые устройства

Тема 5.1. Дешифраторы и шифраторы

ЛАБОРАТОРНАЯ РАБОТА №1

Исследование работы дешифратора экспериментальным путем

Цель: Изучить принципы работы шифратора и дешифратора. Научиться строить таблицу истинности и схему устройства.

Порядок выполнения

1. Изучить теоретическую часть
2. Выполнить практическое задание
3. Ответить на контрольные вопросы
4. Оформить отчет по проделанной работе

Теоретическая часть

Шифраторы и дешифраторы являются примерами простейших преобразователей кодов.

Дешифратор (decoder) – это комбинационное устройство, позволяющее распознавать числа, представленные позиционным n-разрядным кодом.

Если на входе дешифратора "- разрядный двоичный код, то на его выходе код "1 из N". В кодовой комбинации этого кода только одна позиция занята единицей, а все остальные – нулевые. Например, код "1 из N", содержащий 4 кодовые комбинации, будет представлен следующим образом:

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Такой код называют **унитарным,** поэтому дешифратор является преобразователем позиционного двоичного кода в унитарный. Так как возможное количество чисел, закодированных n-разрядным двоичным кодом,

равно количеству наборов из и аргументов (N = 2”), то дешифратор, имеющий n входов, должен иметь 2n выходов. Такой дешифратор называют полным. Если часть входных наборов не используется, то дешифратор называют неполным, и у него число выходов меньше 2n. Таким образом, в зависимости от входного двоичного кода на выходе дешифратора возбуждается только одна из выходных цепей, по номеру которой можно распознать входное число.

Дешифраторы применяют для расшифровки адресов ячеек запоминающих устройств, высвечивания букв и цифр на мониторах, индикаторах и других устройствах.

Проиллюстрируем синтез дешифраторов на примере полного дешифратора трёхразрядных чисел. Таблица истинности дешифратора (таблица 1) представляет ряд единиц, расположенных по диагонали таблицы, а в остальных клетках которой стоят нули.

Таблица 1

№	Входы			Выходы							
	X3	X2	X1	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	1	0	0	0	0	0
3	0	1	1	0	0	0	1	0	0	0	0
4	1	0	0	0	0	0	0	1	0	0	0
5	1	0	1	0	0	0	0	0	1	0	0
6	1	1	0	0	0	0	0	0	0	1	0
7	1	1	1	0	0	0	0	0	0	0	1

Выходы дешифратора имеют нумерацию, совпадающую с десятичным представлением двоичного числа от 0 до 2^n - 1 . Если, например, двоичное число на входе имеет код 101 (табл. 1), то единичный сигнал будет только на

пятом выходе дешифратора, т.е. Z5=1, а на остальных выходах будут нули. Работа этого дешифратора описывается восемью логическими функциями. Составленные по единицам они имеют вид:

$$Z0=\overline{X3}\,\overline{X2}\,\overline{X1};\quad Z2=\overline{X3}X2\overline{X1};\quad Z4=X3\overline{X2}\,\overline{X1};\quad Z6=X3X2\overline{X1};$$

$$Z1=\overline{X3}\,\overline{X2}X1;\quad Z3=\overline{X3}X2X1;\quad Z5=X3\overline{X2}X1;\quad Z7=X3X2X1.$$

Каждая из функций представляет конъюнкцию трёх переменных, а значит, может быть реализована на трехвходовых схемах И. Число элементов одноступенчатого дешифратора определяется числом выходов.

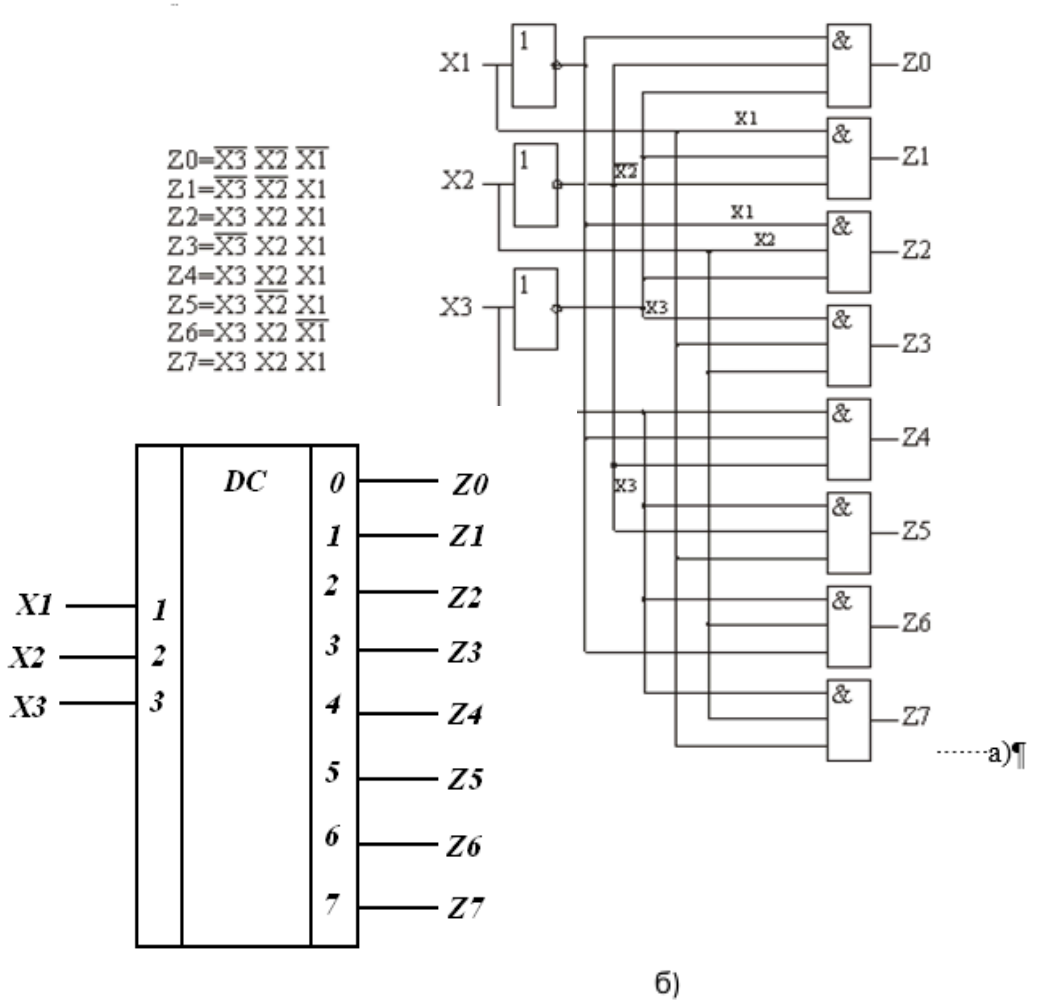


Рисунок 1. а – принципиальная схема линейного дешифратора, б – условное графическое обозначение дешифратора

Шифратор – это логическое устройство, выполняющее преобразование позиционного кода в *n* разрядный двоичный код.

Таким образом, шифратор – это комбинационное устройство, реализующее обратную дешифратору функцию.

Полный двоичный шифратор имеет 2^n входов и *n* выходов.

Шифратор **n-m** – это преобразователь унитарного кода «1 из n» в двоичный (параллельный) код, у которых число выходов m однозначно связано с числом входов n как 2^m. При $n = 2^m$ используется полный набор выходных двоичных комбинаций Y_i. Такой шифратор называется *полным*. Например, шифратор 8-3 полный, т.к. он реализует полный набор возможных комбинаций переменных $X_i(n=8)$ в полный выходной набор $Y_i(m=3)$ как $2^3=8$.

Таблица истинности полного шифратора 8-3

Число	Входы								Выходы		
	X0	X1	X2	X3	X4	X5	X6	X7	Y2	Y1	Y0
0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0	1	0
3	0	0	0	1	0	0	0	0	0	1	1
4	0	0	0	0	1	0	0	0	1	0	0
5	0	0	0	0	0	1	0	0	1	0	1
6	0	0	0	0	0	0	1	0	1	1	0
7	0	0	0	0	0	0	0	1	1	1	1

$$\left.\begin{array}{l} Y_0 = X_1 + X_3 + X_5 + X_7 = \overline{\overline{X}_1 \overline{X}_3 \overline{X}_5 \overline{X}_7} \\ \end{array}\right.;$$

$$Y_1 = X_2 + X_3 + X_6 + X_7 = \overline{\overline{X}_2 \overline{X}_3 \overline{X}_6 \overline{X}_7};$$

$$Y_2 = X_4 + X_5 + X_6 + X_7 = \overline{\overline{X}_4 \overline{X}_5 \overline{X}_6 \overline{X}_7}.$$

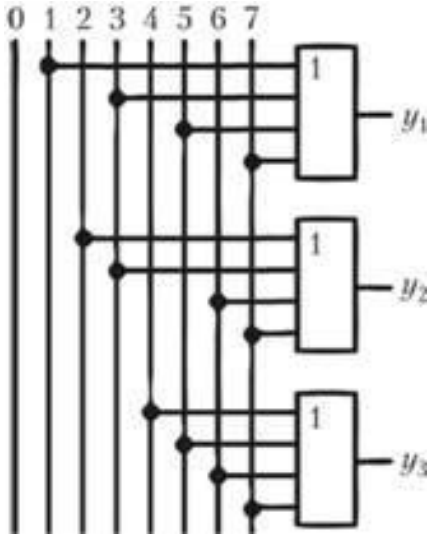


Рисунок 2 – Схема полного линейного шифратора 8-3 на элементах ИЛИ;

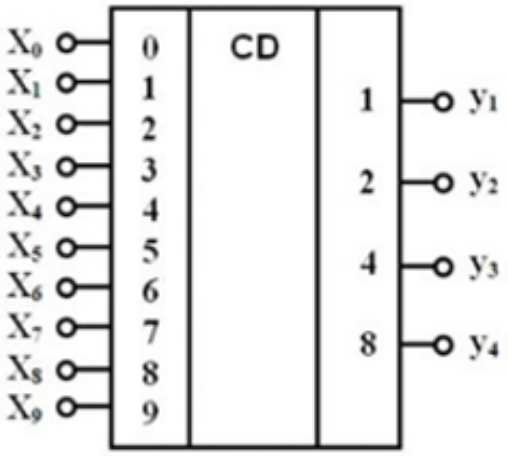


Рисунок 3 – УГО шифратора 10-4

Одно из основных применений шифратора – ввод данных с клавиатуры, при котором нажатие любой клавиши с десятичной цифрой должно приводить к передаче в устройство двоичного кода данной цифры. В этом случае нужен неполный шифратор с десятью входами и четырьмя выходами (10-4).

Таблица функционирования шифратора 10-4 имеет вид:

Входы X: десятичное число	Выходы			
	Двоичный код			
	Y3	Y2	Y1	Y0
X0: 0	0	0	0	0
X1: 1	0	0	0	1
X2: 2	0	0	1	0
X3: 3	0	0	1	1
X4: 4	0	1	0	0
X5: 5	0	1	0	1
X6: 6	0	1	1	0
X7: 7	0	1	1	1
X8: 8	1	0	0	0
X9: 9	1	0	0	1

Из таблицы следует, что

$Y0 = X1 \vee X3 \vee X5 \vee X7 \vee X9;$

$Y1 = X2 \vee X3 \vee X6 \vee X7;$

$Y2 = X4 \vee X5 \vee X6X7;$

$Y3 = X8 \vee X9$

Для реализации шифратора на элементах И-НЕ необходимо выражения преобразовать по теореме де Моргана:

$$Y0 = \overline{\overline{X1 \vee X3 \vee X5 \vee X7 \vee X9}} = \overline{\overline{X1}\,\overline{X3}\,\overline{X5}\,\overline{X7}\,\overline{X9}};$$

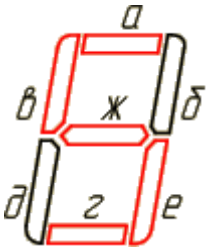
$$Y1 = \overline{\overline{X2 \vee X3 \vee X6 \vee X7}} = \overline{\overline{X2}\,\overline{X3}\,\overline{X6}\,\overline{X7}};$$

$$Y2 = \overline{\overline{X4 \vee X5 \vee X6 \vee X7}} = \overline{\overline{X4}\,\overline{X5}\,\overline{X6}\,\overline{X7}};$$

$$Y3 = \overline{\overline{X8 \vee X9}} = \overline{\overline{X8}\,\overline{X9}};$$

Преобразователи кодов – устройства, предназначенные для преобразования одного кода в другой, при этом часто они выполняют нестандартные преобразования кодов. Преобразователи кодов обозначают через X/Y.

Наиболее широко преобразователи кодов известны применительно к цифровым индикаторам. Например, преобразователь 4-х разрядного позиционного двоичного кода в десятичные цифры. Имеется семи сегментный индикатор и с его помощью требуется высветить десять цифр.



0 1 2 3 4 5 6 7 8 9

Семисегментный индикатор

Имеется семь элементов (а-ж), расположенных так, как показано на рисунке. Каждый элемент может светиться либо не светиться, в зависимости от значения соответствующей логической переменной, управляющей его свечением. Вызывая свечение элементов в определенных комбинациях, можно получить изображение десятичных цифр 0-9.

Очевидно, что двоичный код должен иметь не менее 4-х разрядов (2^4 = 16, что больше 10). Составим таблицу истинности работы такого преобразователя:

Цифра	Двоичный код x4 x3 x2 x1				а (y1)	б (y2)	в (y3)	г (y4)	д (y5)	е (y6)	ж (y7)
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0

2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

При построении таблицы были приняты следующие условия: если элемент индикатора светится, то это означает, что он находится в состоянии лог. 1, если погашен, то он находится в состоянии лог. 0.

Управление элементом осуществляется таким образом, что высокий уровень лог. 1 на некотором входе индикатора вызывает гашение соответствующего элемента. Например, для высвечивания цифры 0 необходимо погасить 7-й элемент (y7=0), оставив остальные элементы в состоянии свечения.

По таблице истинности составим систему собственных функций для всех выходов, т.е. СДНФ, минимизируем её и получим логические выражения для выходных величин y1-y7:

$$w1 = \overline{x1} \bullet \overline{x2} \cdot x3$$

$$w2 = x1 \cdot \overline{x2} \bullet \overline{x3} \bullet \overline{x4}$$

$$w3 = \overline{x1} \cdot x2 \cdot x3$$

$$w4 = x1 \cdot \overline{x2} \cdot x3$$

$$w5 = \overline{x1} \cdot x2 \cdot \overline{x3}$$

$$w6 = x1 \cdot x2 \cdot x3$$

$$w7 = x1 \cdot x2 \cdot x3$$

$w8 = x1 \cdot x2$

$w9 = \overline{x1} \bullet \overline{x2} \bullet \overline{x3}$

$y1 = w1 \cup w2 = \overline{\overline{w1} \bullet \overline{w2}}$

$y2 = w3 \cup w4$

$y3 = w5$

$y4 = w1 \cup w2 \cup w6$

$y5 = w1 \cup w7$

$y6 = w2 \cup w5 \cup w8$

$y7 = w6 \cup w9$

Кроме обычных шифраторов существуют также **приоритетные шифраторы**. Такие шифраторы выполняют более сложную операцию. При работе ЭВМ и других устройств часто решается задача определения приоритетного претендента на обслуживание. Несколько конкурентов выставляют свои запросы на обслуживание, которые не могут быть удовлетворены одновременно. Нужно выбрать, кому предоставляется право первоочередного обслуживания.

Простейший вариант задачи – присвоение каждому источнику запросов фиксированного приоритета. Например, группа из восьми запросов R7, ..., R0 (R – от англ. request – запрос) формируется гак, что высший приоритет имеет источник номер семь, а далее приоритет уменьшается от номера к номеру. Самый младший приоритет у пулевого источника – он будет обслуживаться только при отсутствии всех других запросов. Если имеются одновременно несколько запросов, обслуживается запрос с наибольшим номером.

Практические задания:

Задание 1

Вариант 1:

Построить таблицу истинности и схему двухразрядного линейного дешифратора 2-4 (с двумя входами X1 и X2, 4 выходами) на элементах И.

Вариант 2:

Построить таблицу истинности и схему линейного шифратора 4-2 (с 4 входами X1-X4 и двумя выходами У0 и У1) на элементах И.

Задание 2

Используя таблицу истинности шифратора 10-4 и логические выражения, преобразованные по теореме де Моргана (см. выше теоретические сведения о шифраторе), построить схему шифратора на элементах И-НЕ.

Контрольные вопросы:

1. Что такое дешифратор?
2. Что такое шифратор?
3. Какие логические функции выполняет дешифратор?
4. Какие логические функции выполняет шифратор?
5. Что называется полным и неполным дешифратором?
6. Опишите принцип действия приоритетного шифратора.

РАЗДЕЛ V. Комбинационные цифровые устройства

Тема 5.2. Мультиплексоры и демультиплексоры

ЛАБОРАТОРНАЯ РАБОТА №2

Исследование работы мультиплексора экспериментальным путем

Цель: Изучить принципы работы мультиплексора и демультиплексора. Научиться строить таблицы истинности и схему устройств.

Порядок выполнения

1. Изучить теоретическую часть
2. Выполнить практическое задание
3. Ответить на контрольные вопросы
4. Оформить отчет по проделанной работе

Теоретическая часть

Мультиплексор – комбинационная схема с несколькими входами и одним выходом. На выход мультиплексора передается информация с одного из пронумерованных входов. Номер входа, с которого данные поступают на выход, определяется комбинацией на управляющих (адресных) входах мультиплексора.

Мультиплексоры различаются по числу (N) входов данных

$$N = 2^n ,$$

где n – количество адресных входов и записывают "мультиплексор N-1".

Алгоритм синтеза устройства, реализующего логическую функцию на основе мультиплексора:

1) представить функцию в виде СДНФ;
2) для данной СДНФ заполнить карту Карно (Вейча);

3) на карте Карно (Вейча) выделить области по количеству информационных входов мультиплексора. Количество строк m и столбцов n в таких областях должно удовлетворять условию: $m,n=2^k$, где k=0,1,2,…Переменные, сохраняющие свое значение в пределах выделенных областей, являются адресными, а остальные – информационными;

4) подать адресные переменные любым способом на адресные входы выбранного (или заданного) мультиплексора, определив таким образом однозначное соответствие адресных областей определенному информационному входу;

5) для каждой области найти МДНФ/МКНФ относительно информационных переменных, для управления информационными входами;

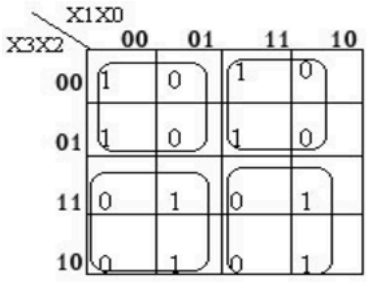
6) с помощью тождественных преобразований МДНФ/МКНФ привести к виду, удобному для совместной реализации;

7) реализовать схемы по каждому информационному входу мультиплексора в выбранном элементном базисе.

Пример построения мультиплексора, реализующего некоторую функцию:

1) $$Y = \bar{X}_3\bar{X}_2\bar{X}_1\bar{X}_0 + \bar{X}_3\bar{X}_2X_1X_0 + \bar{X}_3X_2\bar{X}_1\bar{X}_0 + \bar{X}_3X_2X_1X_0 + X_3X_2\bar{X}_1X_0 + X_3X_2X_1\bar{X}_0 + X_3\bar{X}_2\bar{X}_1X_0 + X_3\bar{X}_2X_1\bar{X}_0$$

2) Для данной функции построим карту Карно:



3) Пусть задан мультиплексор с 4 информационными входами (и 2 входа – адресные). На карте Карно выделим 4 адресные области. Переменные,

которые сохраняют свое значение в пределах выделенных областей, являются адресными: X_1, X_3.

4) Их можно двумя способами подать на адресные входы: $A_1=X_1$, $A_2=X_3$ либо $A_1=X_3$, $A_2=X_1$ (способ подачи не имеет значения).

5) Адресным областям соответствуют информационные входы D_0, D_1, D_2, D_3.

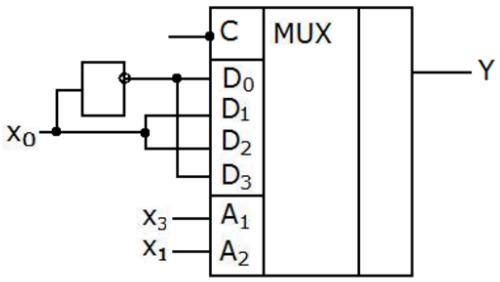
	0 0	0 1	1 1	1 0
0 0	D_0		D_1	
1 1	D_2		D_3	

Для каждой области найти МДНФ относительно информационных переменных X_0, X_2 (по принципам работы с картами Карно).

Получаем:

$D_0=\overline{x}_0$, $D_1=X_0$, $D_2=X_0$, $D_3=\overline{x}_0$

Реализуем полученные функции:



Практические задания:

Задание 1

Построить схему мультиплексора 8-1.

Задание 2

Построить схему демультиплексора (1-2), имеющего 1 вход и 2 выхода, используя элементы И, НЕ

Задание 3

Построить схему устройства, реализующего логическую функцию (по вариантам) на основе мультиплексора

x_3	x_2	x_1	x_0	*вариант 1*	*вариант 2*
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	1	0
0	1	0	0	1	1
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	0	1	1	1	0
1	1	0	0	1	0
1	1	0	1	1	1

1	1	1	0	0	1
1	1	1	1	1	1

Контрольные вопросы:

1. Что такое мультиплексор и для чего мультиплексоры используются?
2. Объясните назначение адресных входов.
3. Объясните назначение информационных входов.
4. Для чего в мультиплексорах используется вход С?
5. Для чего применяют каскадирование мультиплексоров?

РАЗДЕЛ V. Комбинационные цифровые устройства

Тема 5.3. Компараторы. Сумматоры

ЛАБОРАТОРНАЯ РАБОТА №3

Исследование работы сумматоров экспериментальным путем

Цель: Изучить принципы работы полусумматора и сумматора. Научиться строить таблицу истинности и схему устройства.

Порядок выполнения

1. Изучить теоретическую часть
2. Выполнить практическое задание
3. Ответить на контрольные вопросы
4. Оформить отчет по проделанной работе

Теоретическая часть

Арифметико-логическое устройство процессора (АЛУ) обязательно содержит в своем составе такие элементы как **сумматоры**. Эти схемы позволяют складывать двоичные числа.

Полусумматор

Полусумматор (рис. 1) имеет два входа a и b для двух слагаемых и два выхода: S – сумма, P – перенос. Обозначением полусумматора служат буквы

HS (half sum – полусумма). Работу его отражает таблица истинности 1 (табл. 1), а соответствующие уравнения имеют вид:

$$\left.\begin{aligned} S &= \bar{a}b + a\bar{b} = a \oplus b \\ P &= ab \end{aligned}\right\} \qquad \textbf{(1)}$$

Таблица 1

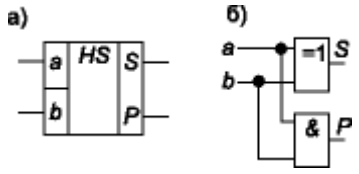


Рис. 1

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

Из уравнений (1) следует, что для реализации полусумматора требуется один элемент “исключающее ИЛИ” и один двухвходовый вентиль И (рис. 1).

Очевидно, что по отношении: к столбцу S реализуется логическая функция «исключающее ИЛИ», т. е. S = A + B. Устройство, реализующее таблицу 1, называют полусумматором, и оно имеет логическую структуру, изображенную на рисунке 2.

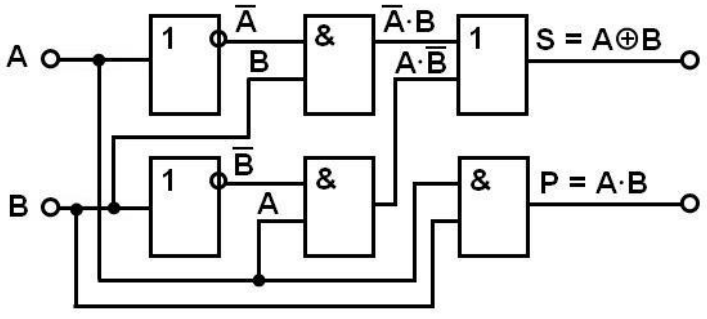


Рисунок 2 – Логическая структура полусумматора

Полный одноразрядный двоичный сумматор

При суммировании двух многоразрядных чисел для каждого разряда (кроме младшего) необходимо использовать устройство, имеющее дополнительный вход переноса. Такое устройство (рисунок 3) называют

полным сумматором и его можно представить как объединение двух полусумматоров ($P_{вх}$ – дополнительный вход переноса). Сумматор обозначают через SM.

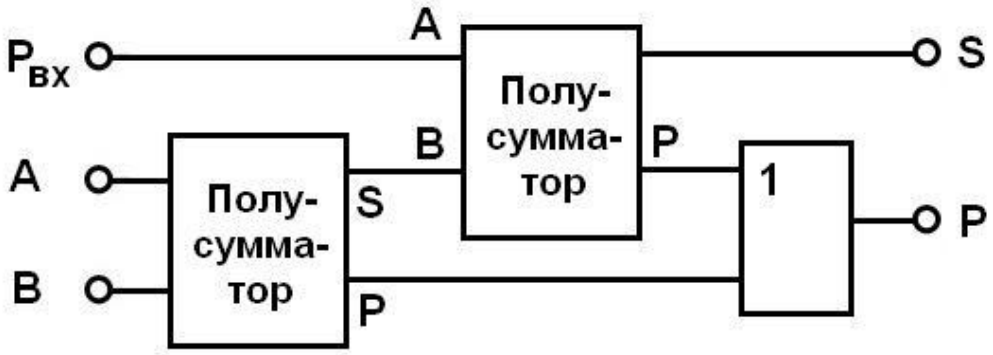


Рисунок 3 – Логическая структура сумматора

Он (рис. 2) имеет три входа: a, b – для двух слагаемых и p – для переноса из предыдущего (более младшего) разряда и два выхода: S – сумма, P – перенос в следующий (более старший) разряд. Обозначением полного двоичного сумматора служат буквы SM. Работу его отражает таблица истинности 2 (табл. 2).

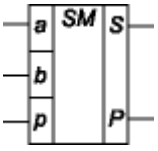


Рис. 2

Таблица 2

№ наб.	a	b	p	P	S
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	1	0
7	1	1	1	1	1

В таблицах 2 и 3 выходные сигналы P и S не случайно расположены именно в такой последовательности. Это подчеркивает, что PS рассматривается как двухразрядное двоичное число, например, $1 + 1 = 2_{10} = 10_2$, то есть P = 1, а S = 0 или $1 + 1 + 1 = 3_{10} = 11_2$, то есть P = 1, а S = 1.

Уравнения, описывающие работу полного двоичного сумматора, представленные в совершенной дизъюнктивной нормальной форме (СДНФ), имеют вид:

$$\left.\begin{aligned} S &= \bar{a}\bar{b}p + \bar{a}b\bar{p} + a\bar{b}\bar{p} + abp \\ P &= \bar{a}bp + a\bar{b}p + ab\bar{p} + abp \end{aligned}\right\} \qquad \textbf{(6)}$$

Уравнение для переноса может быть минимизировано:

P = ab + ap + bp.

Поскольку полусумматор имеет только два входа, он может использоваться для суммирования лишь в младшем разряде.

Цифровые компараторы

Цифровые компараторы выполняют сравнение двух чисел, заданных в двоичном коде. Они могут определять равенство двух двоичных чисел A и B с одинаковым количеством разрядов либо вид неравенства A> B или A < B. Цифровые компараторы имеют три выхода.

Схема одноразрядного компаратора представляет собой структуру логического элемента «исключающее ИЛИ-НЕ» (рисунок 4).

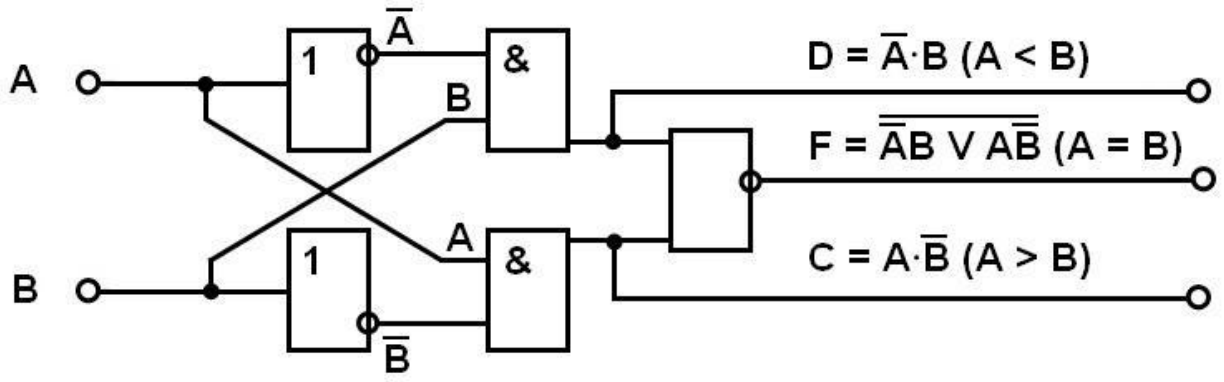


Рисунок 4 – Логическая схема компаратора

Из анализа схемы следует, что если A= B, то F = 1, в противном случае, т. е. при A ≠ B, F = 0. Если A > B, т. е. A = 1, B = 0, то C = 1, а если A < B, т. е. A = 0, B = 1, то D = 1.

Если попарно равны между собой все разряды двух n-разрядных двоичных чисел, то равны и эти два числа A и B. Применяя цифровой компаратор для каждого разряда, например, четырехзначных чисел, и определяя значения F1, F2, F3, F4 логических переменных на выходах компараторов, факт равенства A = B установим в случае, когда F = F1 · F2 · F3 · F4 = 1. Если же F = 0, то A ≠ B.

Неравенство A > B обеспечивается (для четырехразрядного числа) в четырех случаях: или $A_4 > B_4$, или $A_4 = B_4$ и $A_3 > B_3$, или $A_4 = B_4$, $A_3 = B_3$ и $A_2 > B_2$, или $A_4 = B_4$, $A_3 = B_3$, $A_2 = B_2$ и $A_1 > B_1$ (где A_4 и B_4 – старшие разряды чисел A и B). Очевидно, что если поменять местами A_1 и B_1, то будет выполняться неравенство A < B.

Цифровые компараторы выпускают, как правило, в виде самостоятельных микросхем. Так, микросхема К564ИП2 (рис. 3.52) является четырехразрядным компаратором, в котором каждый из одноразрядных компараторов аналогичен рассмотренной ранее схеме. Данная микросхема имеет расширяющие входы A < B, A = B, A > B, что позволяет наращивать разрядность обоих чисел. Для этого компараторы соединяют каскадно или параллельно (пирамидально).

Практические задания:

Задание 1

Постройте таблицу истинности и его логическую схему для полусумматора. Приведите УГО.

Задание 2

Постройте таблицу истинности и его логическую схему для сумматора. Приведите УГО.

Контрольные вопросы

1. Что представляет собой полусумматор?
2. Дайте определение и опишите назначение сумматора.
3. Приведите классификацию сумматоров по различным признакам.
4. Дайте определение и опишите назначение компаратора.

РАЗДЕЛ VI. Последовательностные цифровые устройства

Тема 6.1. Триггеры

ЛАБОРАТОРНАЯ РАБОТА №4

Исследование работы триггеров экспериментальным путем

Цель: Изучить принципы работы триггеров. Научиться строить таблицу истинности и схему устройства.

Порядок выполнения

1. Изучить теоретическую часть
2. Выполнить практическое задание
3. Ответить на контрольные вопросы
4. Оформить отчет по проделанной работе

Теоретическая часть

Триггер (триггерная система) – класс электронных устройств, обладающих способностью длительно находиться в одном из двух устойчивых состояний («1» или «0») и чередовать их под воздействием внешних сигналов.

Триггер – это простейшее устройство, предназначенное для записи и хранения одноразрядных двоичных чисел.

Триггер имеет два выхода (прямой Q и инверсный $\bar{Q}$) и один или несколько входов.

Входы триггера разделяются на информационные и управляющие.

Информационные входы обозначаются следующим образом:

S – вход для установки в состояние «1»;

R – вход для установки в состояние «0»;

J – вход для установки в состояние «1» в универсальном триггере;

K – вход для установки в состоянии «0» в универсальном триггере;

T – счётный (общий) вход;

D – вход для установки в состояние «1» или состояние «0».

Управляющие входы обозначаются:

V – дополнительный управляющий вход для разрешения приёма информации;

C – вход синхронизации.

Триггеры имеют 2 выхода:

Q – прямой,

$\bar{Q}$ – инверсный.

Триггеры можно классифицировать по способу приёма информации, принципу построения, функциональным возможностям.

По способу приёма информации триггеры подразделяются на **асинхронные** и **синхронные**.

Асинхронные триггеры воспринимают информационные сигналы и реагируют на них в момент появления на входах триггера.

Синхронные триггеры реагируют на информационные сигналы при наличии разрешающего сигнала на специальном управляющем входе C, называемом входом синхронизации. Синхронные триггеры подразделяются на триггеры со статическим и динамическим управлением по входу C. Триггеры со статическим управлением воспринимают информационные сигналы при

подаче на С-вход уровня 1 (прямой С-вход). Триггеры с динамическим управлением воспринимают информационные сигналы при изменении сигнала на С-входе от 0 к 1(прямой динамический С–вход) или от 1 к 0 (инверсный динамический С–вход).

По принципу построения триггеры со статическим управлением делятся на **одноступенчатые** и **двух ступенчатые.**

Одноступенчатые триггеры характеризуются наличием одной ступени запоминания информации, двухступенчатые триггеры имеют две ступени запоминания информации. Вначале информация записывается в первую ступень, а затем переписывается во вторую и появляется на выходе.

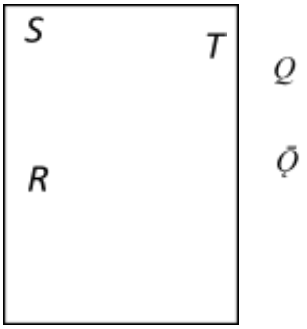
Двухступенчатый триггер обозначают через ТТ.

По функциональным возможностям триггеры разделяются на следующие классы:

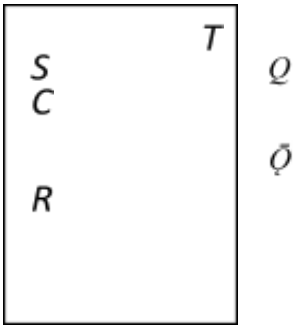
- с раздельной установкой состояния 0 и 1 (RS-триггеры);
- универсальные (JK-триггеры);
- с приемом информации по одному входу D (D-триггеры);
- со счётным входом Т (Т-триггеры).

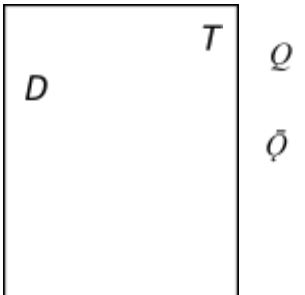
Независимо от вида, если Q= 1 и $\bar{Q}$ = 0, то триггер находится в единичном состоянии. При Q= 0, $\bar{Q}$ = 1 состояние триггеры называется нулевым. Ниже приведены графические обозначения (УГО) триггеров, принятые в системе ЕСКД:

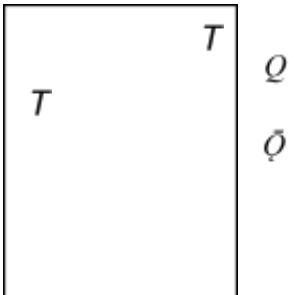
а)



б)

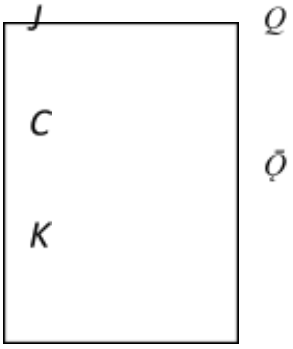






C

в) г)



д)

Рисунок 1 – Триггеры:

1) асинхронный RS-синхронный триггер
2) синхронизируемый RS-триггер
3) D-триггер
4) T-триггер
5) JK-триггер

Функционирование триггеров описывается таблицами переходов (истинности):

RS-триггер

Входы		Выходы	
R	*S*	*Q*	$\bar{Q}$
0	0	исх.	исх.
0	1	1	0
1	0	0	1
1	1	неопр.	неопр.

D-триггер

Входы		Выходы
D	*C*	*Q*
1	1	1
0	1	0

T-триггер

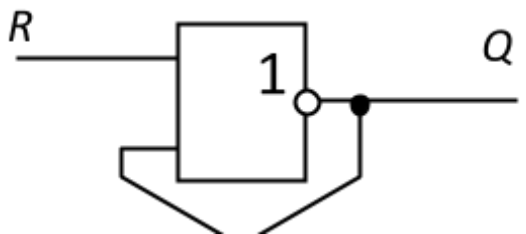
Входы	Выходы
T	*Q*
0	исх.
1	$\bar{Q}_{n-1}$

JK-триггеры

Входы		Выходы
J	*K*	*Q*
0	0	исх.
0	1	0
1	0	1
1	1	$\bar{Q}_{n-1}$

«исх.» – означает режим хранения (исходное состояние или без изменения). «неопр.» – означает, что состояние выходов неопределённое, т. е. комбинация сигналов R= S= 1 является запрещённой.

Схема асинхронного RS-триггера на логических элементах 2ИЛИ-НЕ приведена на рисунке. Управление на схеме осуществляется положительными сигналом (единицей).



Практические задания:

Вариант 1

Постройте таблицу истинности асинхронного RS-триггера. Нарисуйте схему асинхронного RS-триггера на логических элементах и его УГО. Поясните принцип работы устройства.

Вариант 2

Постройте таблицу истинности D-триггера. Нарисуйте схему D-триггера на логических элементах и его УГО. Поясните принцип работы устройства.

Вариант 3

Постройте таблицу истинности JK-триггера. Нарисуйте схему JK-триггера на логических элементах и его УГО. Поясните принцип работы устройства.

Вариант 4

Постройте таблицу истинности T-триггера. Нарисуйте схему T-триггера на логических элементах и его УГО. Поясните принцип работы устройства.

Контрольные вопросы:

1. Что такое триггер?
2. Опишите основное назначение триггеров.

3. Опишите основные виды триггеров, приведите их УГО (условно-графическое изображение).
4. По каким признакам классифицируются триггеры?
5. Опишите принцип действия синхронного и асинхронного триггера
6. Назовите область применения триггеров.
7. Почему JK-триггер считается универсальными?

РАЗДЕЛ VI. Последовательностные цифровые устройства

Тема 6.2. Регистры

ЛАБОРАТОРНАЯ РАБОТА №5

Исследование работы регистра экспериментальным путем

Цель: Изучить принципы работы регистров. Научиться строить таблицу истинности и схему устройства.

Порядок выполнения

1. Изучить теоретическую часть
2. Выполнить практическое задание
3. Ответить на контрольные вопросы
4. Оформить отчет по проделанной работе

Теоретическая часть

Регистры предназначены для хранения и преобразования многоразрядных двоичных чисел. Для запоминания отдельных разрядов числа

могут применяться триггеры различных типов. Одиночный триггер можно считать одноразрядным регистром.

Занесение информации в регистр называется операцией записи. Операция выдачи информации из регистра – считывание.

Перед записью информации в регистр, его необходимо обнулить.

Классификация регистров:

1. **по способу ввода/вывода информации:**
   - параллельные (регистры хранения) – информация вводится и выводится одновременно по всем разрядам;
   - последовательные (регистры сдвига) – информация бит за битом «проталкивается» через регистр и выводится также последовательно;
   - комбинированные (последовательно-параллельные) – параллельный ввод и последовательный вывод (и наоборот).

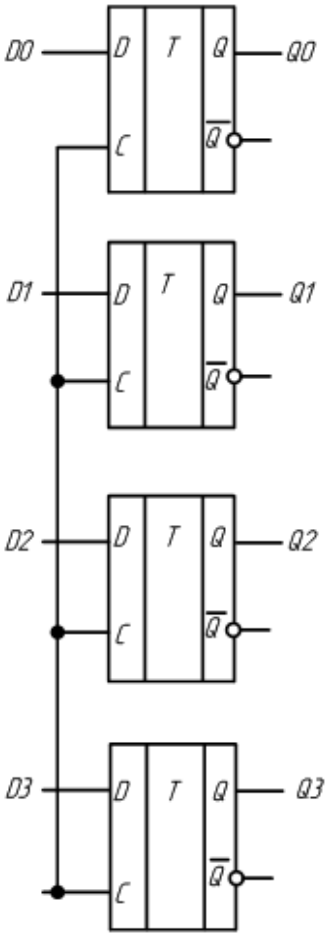
2. **по способу представления информации:**
   - однофазные – информация представляется в прямом или обратном (инверсном) виде;
   - парафазные – информация представляется и в прямом, и в обратном виде.

Параллельный регистр

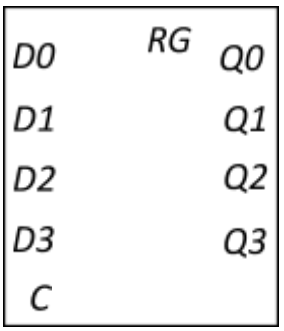
Параллельные регистры осуществляют прием и выдачу информации в параллельном коде, а это значит, что для передачи каждого разряда используется отдельная линия. Все разряды двоичного кода слова передаются в одном временном такте.

Для записи информации в регистр на его входных выводах (D0-D3) нужно установить значения (логические уровни), после чего на вход синхронизации (C) подать разрешающий импульс – логическую единицу. После этого на выходах Q0-Q3 появится записанное слово. Регистры

запоминают входные сигналы только в момент времени, определяемый сигналом синхронизации.



а)



б)

Рисунок 1 – а) схема параллельного регистра на D-триггерах, б)УГО

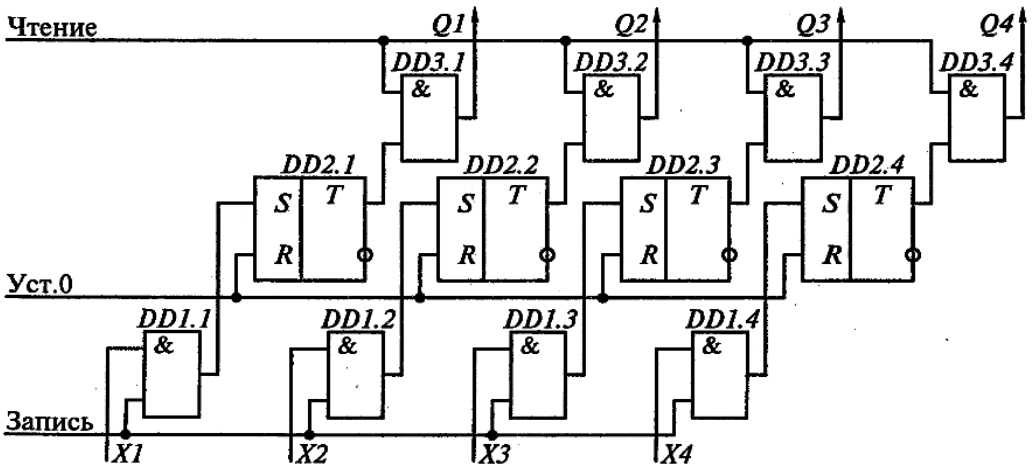


Рисунок 2 – Схема параллельного регистра на RS-триггерах

Регистр может работать в следующих режимах:

- предварительная установка (подготовка триггеров регистра к записи информации);
- запись информации в триггеры регистра;
- хранение ранее записанной информации;
- чтение информации из регистра.

Логические элементы И (DD1.1-1.4) используются для ввода в регистр четырехразрядной информации со входов X1-X4. Информация хранится в асинхронных RS-триггерах (элементы памяти DD2.1-2.4). Для чтения информации из регистра служат прямые выходы триггеров (выходные элементы DD3.1-3.4).

При решении практических задач часто требуется разрядность параллельных регистров большая восьми. В таком случае можно увеличивать их разрядность параллельным соединением готовых микросхем.

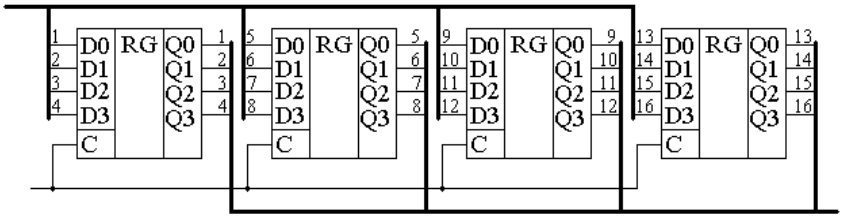


Рисунок 3 – Схема параллельного соединения четырёх регистров

Последовательные регистры

Последовательный регистр (регистр сдвига) обычно служит для преобразования последовательного кода в параллельный и наоборот. Применение последовательного кода связано с необходимостью передачи большого количества двоичной информации по ограниченному количеству соединительных линий. При параллельной передаче разрядов требуется большое количество соединительных проводников. Если двоичные разряды последовательно бит за битом передавать по одному проводнику, то можно значительно сократить размеры соединительных линий на плате (и размеры корпусов микросхем).

Внутри сдвигового регистра триггеры соединены последовательно, то есть выход первого соединён с входом второго и т.д.: межразрядные связи соединяют выходы триггеров младших разрядов с входами триггеров старших разрядов, что позволяет осуществить последовательное продвижение информации из разряда в разряд под воздействием управляющих синхроимпульсов С.

Последовательный регистр имеет один вход для последовательного ввода информации. Входы синхронизации в последовательных (сдвиговых) регистрах, как и в параллельных регистрах, объединяются. Это обеспечивает одновременность смены состояния всех триггеров, входящих в состав последовательного (сдвигового) регистра.

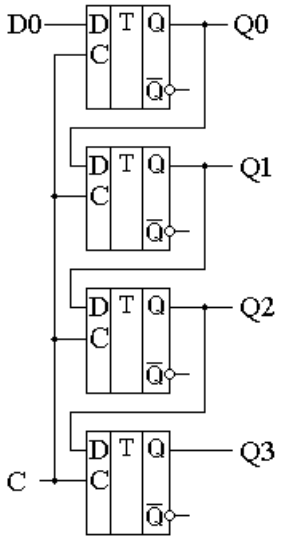


Рисунок 4 – Схема последовательного регистра на D-триггерах

Рассмотрим работу этого регистра. Можно предположить, что в начале все триггеры регистра находятся в состоянии логического нуля, т.е. Q0=0, Q1=0, Q2=0, Q3=0. Если на входе D-триггера T1 имеет место логический 0, то поступление синхроимпульсов на входы «C» триггеров не меняет их состояния.

Синхроимпульсы поступают на соответствующие входы всех триггеров регистра одновременно и записывают в них то, что имеет место на их информационных входах. На информационных входах триггеров T2, T3, T4 – уровни логического «0», т.к. информационные входы последующих триггеров соединены с выходами предыдущих триггеров, находящихся в состоянии логического «0», а на вход «D» первого триггера, по условию примера, подается «0» из внешнего источника информации. При подаче на вход «D» первого триггера «1», с приходом первого синхроимпульса, в этот триггер запишется «1», а в остальные триггеры – «0», т.к. к моменту поступления фронта синхроимпульса на выходе триггера T1 ещё присутствовал логический «0». Таким образом, в триггер T1 записывается та информация (тот бит), которая была на его входе «D в момент поступления фронта синхроимпульса и т.д.

При поступлении второго синхроимпульса логическая «1» с выхода первого триггера, запишется во второй триггер, и в результате происходит сдвиг первоначально записанной «1» с триггера Т1 в триггер Т2, из триггера Т2 в триггер Т3 и т.д. Таким образом, производится последовательный сдвиг поступающей на вход регистра информации (в последовательном коде) на один разряд вправо в каждом такте синхроимпульсов.

После поступления четырёх синхроимпульсов регистр оказывается полностью заполненным разрядами числа, вводимого через последовательный ввод «D». В течение следующих четырёх синхроимпульсов производится последовательный поразрядный вывод из регистра записанного числа, после чего регистр оказывается полностью очищенным (регистр окажется полностью очищенным только при условии подачи на его вход уровня «0» в режиме вывода записанного числа).

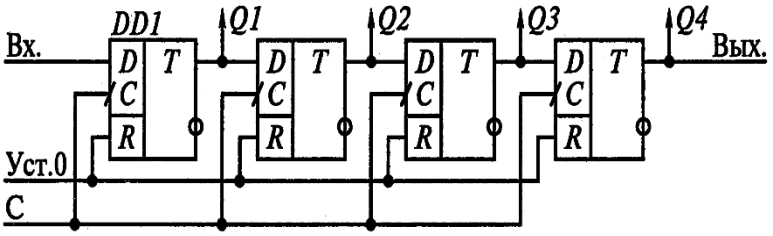


Рисунок 5 – Регистр параллельного действия

Регистр последовательного действия уступает регистру параллельного действия в быстродействии и используется только в случаях, когда количество входных и выходных линий связи ограничено.

Практические задания:

Задание 1.

Построить на D-триггерах схемы последовательного и параллельного регистров в синхронной реализации

Задание 2.

Нарисовать УГО последовательного и параллельного регистра. Определить как связанны отдельные триггеры в данных регистрах.

Контрольные вопросы:

1. Назовите основное назначение регистров.
2. По каким признакам классифицируются регистры?
3. Чем определяется разрядность регистров?
4. Объясните принцип работы последовательного регистра.
5. Объясните принцип работы параллельного регистра.
6. Опишите назначение последовательно-параллельного регистра.

РАЗДЕЛ VI. Последовательностные цифровые устройства

Тема 6.3. Счетчики

ЛАБОРАТОРНАЯ РАБОТА№6

Исследование работы счетчиков экспериментальным путем

Цель: Изучить принципы работы счетчиков. Научиться строить таблицу истинности и схему устройства.

Порядок выполнения

1. Изучить теоретическую часть
2. Выполнить практическое задание
3. Ответить на контрольные вопросы
4. Оформить отчет по проделанной работе

Теоретическая часть

Счетчики – устройства, предназначенные для счета числа импульсов, поступающих на их вход. Каждый счетный импульс изменяет состояние счетчика на единицу. Если при счете коды состояния расположены в возрастающем порядке, то счетчик называется суммирующем, если в убывающем порядке – вычитающим. Счетчики, у которых направление счета может изменяться, называются реверсивными.

Число разрешенных состояний счетчика называется **модулем счета** M (или коэффициентом пересчета счетчика $К_{сч}$). Характеризует число (количество) устойчивых состояний, в которых может находиться n-разрядный счетчик, т.е. предельное число входных сигналов, которое может быть подсчитано счетчиком. При поступлении на счетчик числа импульсов больше $К_{сч}$ счетчик возвращается в исходное состояние.

Счетчики, у которых модуль счета равен целой степени числа 2 ($M = 2^n$), называются **двоичными**.

Основу счетчиков составляют триггерные схемы. Счетчики могут быть собраны на D-триггерах, JK-триггерах. Триггеры соединяются последовательно.

По способу организации межразрядных связей счетчики делятся на:

- Асинхронные (переключение триггеров в разрядах осуществляется последовательно один за другим);
- Синхронные (переключение триггеров в разрядах осуществляется одновременно по сигналу синхронизации).

Представление счётчика цепочкой счётных триггеров справедливо как для суммирующего, так и для вычитающего вариантов. При прямом счёте (суммирование) следующий разряд переключается при переходе предыдущего в направлении из 1 в 0, а при обратном счёте – при переключении из 0 в 1.

Простейший вычитающий асинхронный счётчик на Т-триггерах

Рассмотрим схему счётчика на Т-триггерах, опрокидывающихся по переднему фронту входных импульсов.

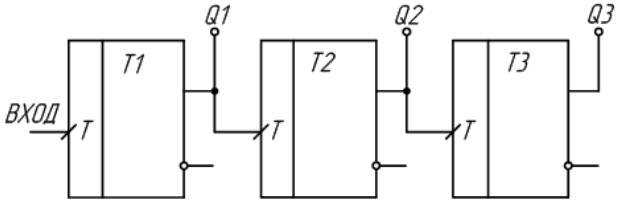
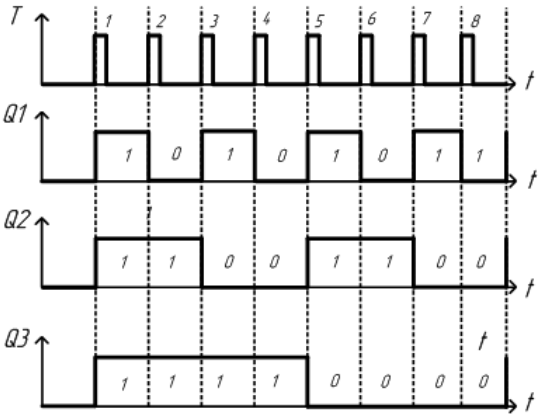


Рисунок 1 – Вычитающий счётчик



Временная диаграмма

Из временной диаграммы видим, что получился вычитающий счётчик. Если информацию снимать с инверсных выходов триггеров, то получится суммирующий счётчик.

Суммирующий асинхронный счетчик на D-триггерах получается, если инверсный выход предыдущего триггера соединить со входом С последующего триггера. При этом инверсный выход D-триггера соединяют с его входом D. Счётный режим возможен только у триггеров динамического типа.

Для построения **вычитающего счетчика на D-триггерах** прямой выход предыдущего триггера соединяют со входом С последующего триггера. При этом инверсный выход D-триггера соединяют с его входом D.

Счётчик с произвольным модулем счёта (с ограниченным модулем счёта)

К счетчику добавляется логическая схема, проверяющая условие: «код на счетчике изображает число равное $К_{сч}$?». В зависимости от результата проверки направляет входной сигнал либо на шину «Установка 0», либо на суммирование к записанному коду.

Это условие можно проверить с помощью n-входовой схемы И, связанной с прямыми выходами тех триггеров, которые при записи в счетчике числа, равного $К_{сч}$ должны находиться в состоянии «1» и с инверсными выходами триггеров, которые в этом случае должны находиться в состоянии «0».

Для построения счётчика с произвольным модулем счета можно использовать двоичный счётчик, у которого модуль счёта М должен быть больше модуля счёта разрабатываемого счётчика с произвольным модулем счёта.

Пример 1:

Пусть нужно сделать счётчик с М= 10 (т.е. счетчик должен иметь 10 состояний: от 0_{10} до 9_{10} или от 0000_2 до 1001_2).

У 4-х разрядного счётчика модуль счёта равен 16 (больше 10).

Схема счётчика представляет собой 4 последовательно включённых счётных триггера, у которых есть вход сброса R (первый триггер содержит младший разряд числа).

Число 10 в двоичной системе счисления представляется 1010. Когда на выходах счетчика будет код 1010, на выходе элемента «И» появится логическая единица, которая запустит схему гашения. Длительность импульса на выходе схемы гашения должна быть достаточна для надёжного сброса всех триггеров счётчика в 0.

Разряды числа 1010, равные 1 подаются на схему «И» с прямых выходов триггеров, а равные 0 - с инверсных. Таким образом, как только счётчик досчитает до 10, произойдёт обнуление всех триггеров и счёт продолжится с кода 0000.

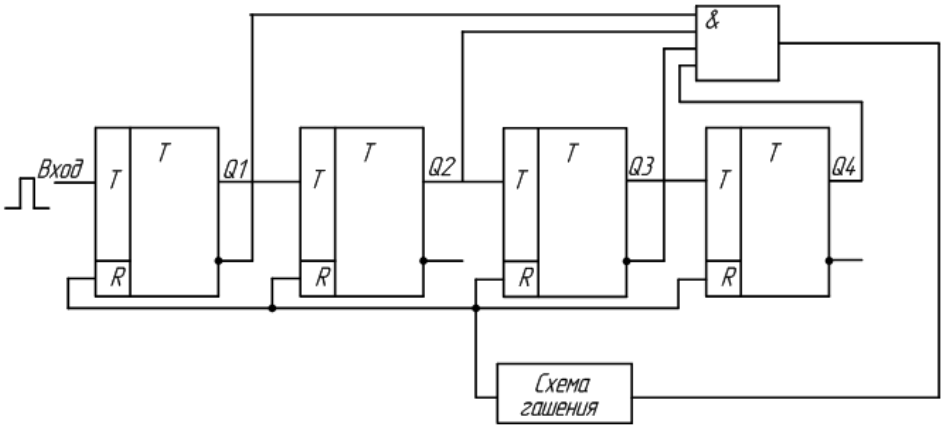
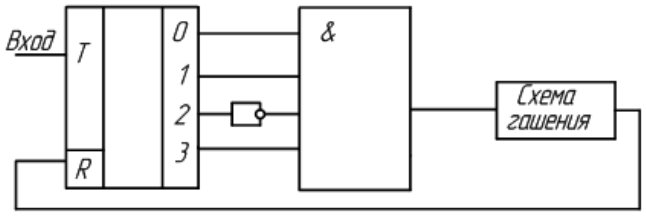


Рисунок 2 – Счётчик с модулем счета М=10 на Т-триггерах

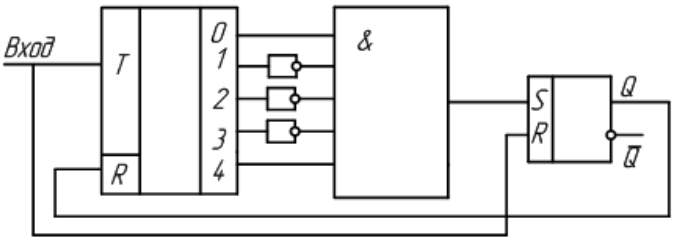
Пример 2: Рассмотрим счётчик с М=11 на основе двоичного счётчика в одной микросхеме (без инверсных выходов).

$11_{10}=1011_2$



Счётчик с модулем счёта М=11

Пример 3: В качестве схемы гашения может быть RS-триггер.



Счётчик с модулем счёта М=17

В этой схеме $М=10001_2 = 17_{10}$

Сигнал на входе К счётчика будет действовать в течение одного периода входных импульсов.

Практические задания:

Задание 1

Варианты 1:

Построить схему асинхронного 4-х разрядного суммирующего счетчика на D-триггерах с R-входом для сброса счетчика. Нарисуйте УГО полученного устройства.

Варианты 2:

Построить схему асинхронного 4-х разрядного вычитающего счетчика на D-триггерах с R-входом для сброса счетчика. Нарисуйте УГО полученного устройства.

Задание 2

Построить схему суммирующего счетчика с модулем счета $К_{сч}$ = вашему варианту

Задание 3

Изобразите временные диаграммы работы суммирующего счётчика.

Задание 4

Изобразите временные диаграммы работы вычитающего счётчика.

Контрольные вопросы:

1. Дайте определение понятию «Счетчик».
2. Приведите классификацию счетчиков.
3. Объясните принцип работы суммирующего счётчика.
4. Объясните принцип работы вычитающего счётчика.
5. Объясните принцип работы счётчика с произвольным модулем счёта.

РАЗДЕЛ VII. Организация устройств ЭВМ

Тема 7.1. Арифметико-логические устройства

ЛАБОРАТОРНАЯ РАБОТА №7

Исследование работы АЛУ экспериментальным путем

Цель: Исследование командного цикла процессора на уровне микрокоманд

Порядок выполнения

1. Изучить теоретическую часть
2. Выполнить практические задания
3. Занести результаты работы программы в таблицу анализа состояния микропроцессора
4. Ответить на контрольные вопросы
5. Оформить отчет по проделанной работе

Теоретическая часть

Арифметико-логическое устройство АЛУ (Arithmetic-Logic Unit – ALU) представляет собой комбинационное устройство на основе сумматора, выполняющее ряд дополнительных функций обработки данных. На рисунке 1 представлено условное обозначение типичного 4-разрядного АЛУ, реализованного в модуле 74181.

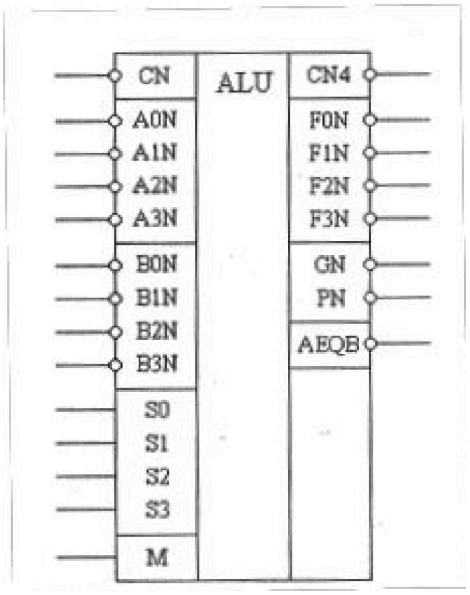


Рисунок 1 – Условное обозначение четырёхразрядного АЛУ (модуль 74181)

Представленное АЛУ содержит две группы входов для двух 4-разрядных операндов (обрабатываемых двоичных чисел) A0N…A3N и B0N...B3N. На выводах F0N…F3N формируется 4-разрядный результат выполненной операции. Вход CN предназначен для подачи входного переноса в младший разряд устройства, если такой перенос существует. Выход CN4 содержит перенос в пятый разряд в случае его возникновения в ходе выполнения операции. Выходы GN и PN предназначены для подачи групповых функций генерации и распространения на внешнюю схему ускоренного формирования переносов (74182). Эта схема может быть использована для построения АЛУ с разрядностью более 4 и параллельным распространением переноса между 4-разрядными блоками. На выходе AEQB формируется сигнал равенства операндов A и B.

Входы SO…S3 и M обеспечивают выбор одной из 32 возможных операций, выполняемых устройством. Список операций представлен в таблице 1. Все операции делятся на две группы: логические и арифметические. Переключение между группами осуществляется с помощью входа M. Приведенная таблица соответствует положительной логике линий данных (1 - высокий уровень). Входные 4-разрядные операнды (данные) и 4-разрядный результат операции обозначены в таблице буквами A, B и F. Бит входного переноса носит имя соответствующего вывода модуля CN.

Логические операции выполняются над операндами побитово (поразрядно), т. е. без взаимодействия разрядов. Эти операции присутствуют и в логическом и в арифметическом режимах. Для отличия от арифметических операций, выполняемых с учетом переносов в разрядах и обозначаемых в таблице обычными символами + и логические операции указаны с помощью кратких обозначений языка AHDL (символом решетка (#) обозначена операция дизъюнкции (ИЛИ), символом доллар ($) – операция сложения по модулю 2 (исключающее ИЛИ)). Если при выполнении операций арифметического режима возникает перенос в пятый разряд, то на выходе CN4 устанавливается 0, в противном случае на нем остается 1.

Внутри центрального процессора (CPU) содержится несколько ячеек памяти, называемых **регистрами.** В данном случае эти регистры занимают один байт (8 бит) памяти. Таким образом, в любой момент времени каждый из этих 8-битных регистров хранит одно значение от 0 до 255, или от $00 до $FF в шестнадцатеричной системе (рисунок 2).

Внутри этого процессора также есть несколько ячеек памяти, называемых **флагами,** каждый из которых занимает один бит памяти и используется для представления булевых значений. Таким образом, в любой момент времени каждый из этих 1-битных флагов имеет значение TRUE, либо FALSE.

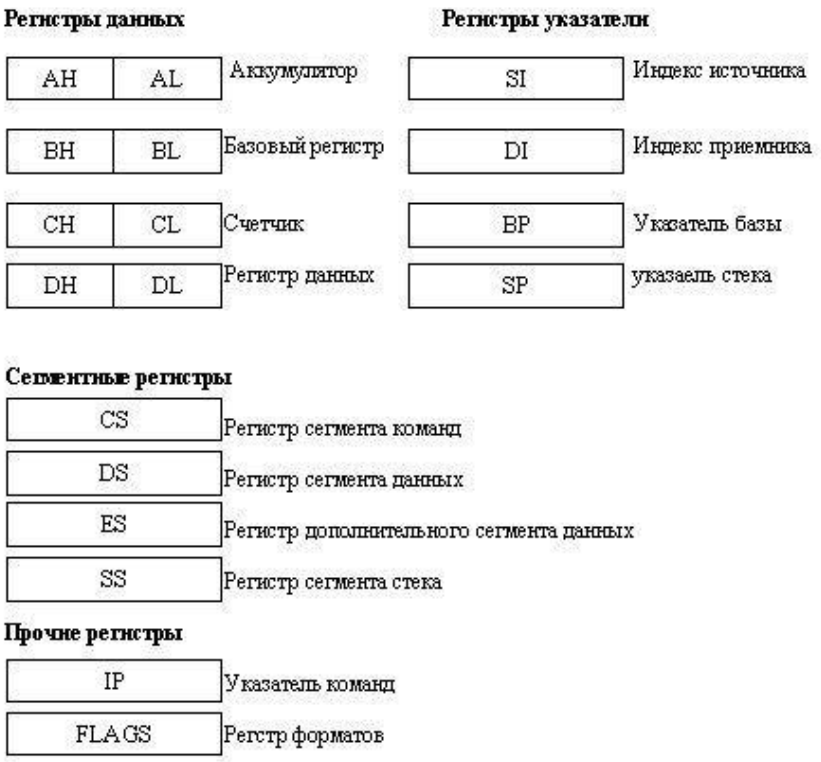


Рисунок 2 – Регистры процессора

Программа – это последовательность инструкций, которые указывают центральному процессору, что делать. Большинство инструкций состоит из операции и одного или нескольких операндов, в зависимости от операции.

Операция – это как функция, встроенная в процессор и предоставляемая программисту для немедленного использования. Каждая

операция имеет короткое запоминающееся имя, называемое мнемоникой. В письменном языке ассемблера операции обозначаются этой мнемоникой.

Операнд – это как аргумент операции. Операндом может быть регистр процессора, ячейка памяти или литеральное значение.

Принципы организации командных и машинных циклов

Команда представляет собой совокупность микрокоманд, которые в виде двоичных кодов хранятся в постоянной памяти (ПЗУ) устройства управления процессора.

Выполнение команды можно рассматривать как процесс считывания из ПЗУ микрокоманд, инициирующих работу некоторых функциональных узлов процессора на отдельных временных интервалах (тактах). Время, затрачиваемое на выполнение команды, называется **командным циклом**.

Команды имеют не одинаковую длительность командных циклов, так как содержат различное число микрокоманд, используют разные способы адресации и другую дополнительную информацию. Это обстоятельство отражается в форматах команд, имеющих длину один, два и более байт.

В командном цикле можно выделить две **основные фазы:**

- **фазу выборки команды**

В этой фазе программный счетчик PC выставляет на адресную шину адрес первого байта команды. Микропроцессор вырабатывает сигнал «чтения памяти», благодаря которому содержимое адресуемой ячейки памяти по шине данных поступает в регистр команд IR (Instruction Register).

Программный счетчика PC формирует адрес, указывающий на следующий элемент объектного кода. Фаза выборки одинакова для всех команд;

- **фазу выполнения команды**

Фаза начинается с дешифрации команды. В результате дешифрации первого байта определяется вид (код) операции и в устройстве управления формируются необходимые для ее выполнения управляющие сигналы.

Действия процессора в фазе выполнения команды и ее продолжительность зависит от вида операции. Для выполнения операции может потребоваться дополнительное обращение к памяти или внешним устройствам за данными, пересылка данных в соответствующие регистры процессора, непосредственное выполнение операции в АЛУ, вывод результатов в память или внешнее устройство и т. д.

Различная длительность командных циклов и неоднородность их отдельных фаз явились причиной организации машинных циклов для выполнения команд.

Каждый машинный цикл представляет собой цикл обращения к системной магистрали. В машинном цикле производится выборка очередного байта (слова) команды или данных из памяти, запись в память, ввод или вывод данных. Командный цикл может содержать различное число машинных циклов. В первом машинном цикле М1 происходит выборка первого байта команды, ее дешифрация и исполнение, если данные находятся во внутренних регистрах процессора. Если выполнение команды требует обращения к памяти или внешним устройствам, то используется два, три и более машинных циклов.

Машинные циклы, как и командные, также не являются однородными. Поэтому на их выполнение затрачивается различное число тактов.

Таким образом, при выполнении команды все действия процессора синхронизированы вложенными друг в друга циклами трех уровней: командными, машинными и тактовыми.

Реализация программы в ЭВМ сводится к последовательному выполнению команд. Каждая команда, в свою очередь, выполняется как последовательность микрокоманд, реализующих элементарные действия над операционными элементами процессора. Для пояснения логики функционирования ЭВМ ее целесообразно представить в виде совокупности узлов, связанных между собой коммуникационной сетью (рисунок 3).

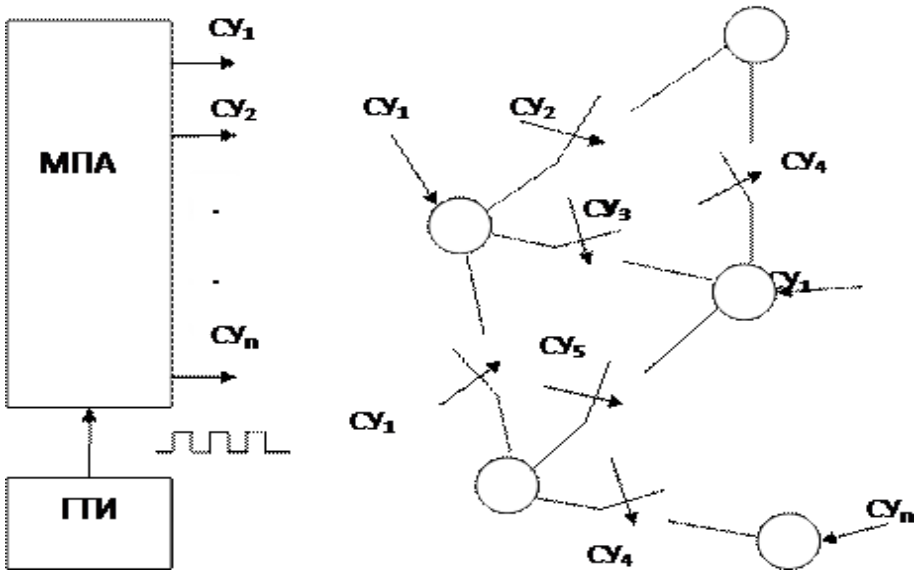


Рисунок 3 – Связь команд микропроцессора

Процесс функционирования вычислительной машины состоит в последовательности пересылок информации между ее узлами и элементарных действий (микроопераций), выполняемых в узлах. Понятие узла здесь трактуется весьма широко: от регистра до АЛУ или основной памяти. Также широко следует понимать и термин «элементарное действие». Это может быть установка регистра в некоторое состояние или выполнение операции в АЛУ.

Любое элементарное действие производится при поступлении соответствующего сигнала управления (СУ) из микропрограммного автомата устройства управления. Возможная частота формирования сигналов на выходе автомата определяется синхронизирующими импульсами, поступающими от генератора тактовых импульсов (ГТИ).

Элементарные пересылки или преобразования информации, выполняемые в течение одного такта синхронизации, называются микрооперациями. В течение одного такта могут одновременно выполняться несколько микроопераций. Совокупность сигналов управления, вызывающих микрооперации, выполняемые в одном такте, называют **микрокомандой.**

Относительно сложные действия, осуществляемые вычислительной машиной в процессе ее работы, реализуются как **последовательность микроопераций**

и могут быть заданы последовательностью микрокоманд, называемой **микропрограммой**.

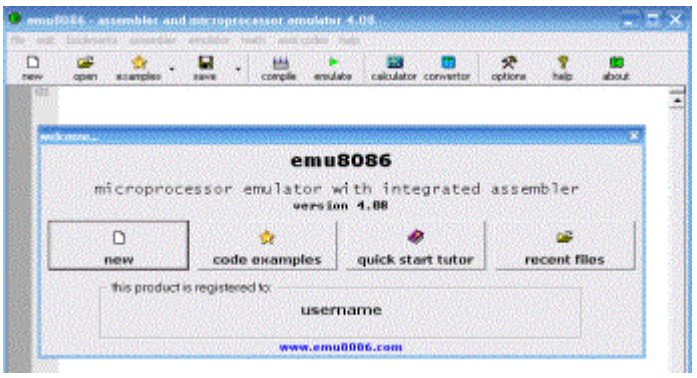
Реализует микропрограмму, то есть вырабатывает управляющие сигналы, задаваемые ее микрокомандами, микропрограммный автомат (МПА).

Практическая часть

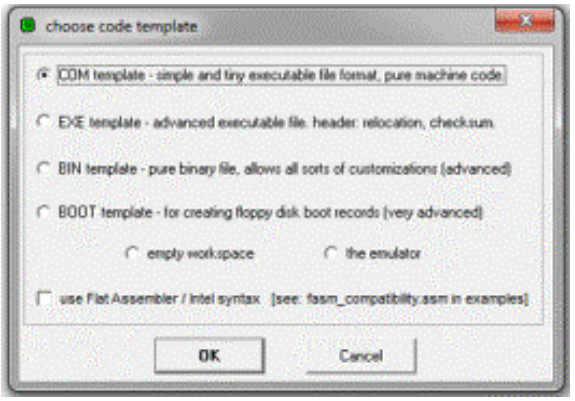
Задание 1.

Исследуйте состояние регистров и памяти во время выполнения программы

1. Запустите эмулятор и щелкните на кнопку new



2. Выберите тип исполняемого файла:



Директивы, определяющие тип исполнимого файла:

#MAKE_COM#

#MAKE_BIN#

#MAKE_BOOT#

#MAKE_EXE#

Вы можете вставить эти директивы в исходный код для определения нужного вам типа исполнимого файла.

Описание типов исполнимых файлов:

#MAKE_COM# – самый старый и самый простой формат исполнимого файла. Такие файлы загружаются с префиксом 100h (256 байтов).

#MAKE_EXE# – более "продвинутый" формат исполнимого файла. Не ограничены размер и количество сегментов.

#MAKE_BIN# – простой исполнимый файл.

#MAKE_BOOT# – эта директива копирует первую дорожку дискеты (загрузочный сектор).

3. Создайте файл, который будет выводить: Hello,world

Для этого введите в строке «add your code here» следующие команды:

№ строки	Команда
1	org 100h
2	begin:
3	mov ah, 9
4	mov dx, offset message
5	int 21h
6	ret
7	message db “Hello word”, 0dh 0ah, ‘$’
8	end begin

```
ORG 100h
begin:
MOV AH, 9
MOV DX, offset message
INT 21h
ret
message db "Hello world", 0dh, 0ah, '$'
end begin
```

Рассмотрим исходный текст программы:

- ***1 строка*** ORG 100h устанавливает значение программного счетчика (IP) в 100h, потому что при загрузке COM–файла в память DOS занимает первые 256 байт (100h) блоком данных PSP и располагает код программы только после этого блока. Все программы, которые компилируются в файлы типа COM, должны начинаться с этой директивы.
- ***2 строка*** Метка BEGIN: располагается перед первой командой в программе и будет использоваться в директиве END (Begin – англ. начало; end – конец), чтобы указать, с какой команды начинается программа. Вообще вместо слова BEGIN можно было бы использовать что-нибудь другое. Например, START:. В таком случае, нам пришлось бы и завершать программу END START.
- ***Строки (3) – (5)*** выводят на экран сообщение “Hello”.

  Команда MOV DX, OFFSET MESSAGE помещает в регистр DX смещение метки MESSAGE относительно начала сегмента данных, который в нашем случае совпадает с сегментом кода. OFFSET (по–английски – это смещение). Когда, при ассемблировании, Ассемблер дойдет до этой строки, он заменит OFFSET MESSAGE на АДРЕС (смещение) этой строки в памяти. Если мы запишем OFFSET MESSAGE (хотя, правильнее будет MOV DX, WORD OFFSET MESSAGE), то в DX загрузится не адрес (смещение), а первые два символа нашей строки (в данном случае "Не"). Так как DX – шестнадцатиразрядный регистр, в него можно загрузить только два байта (один символ всегда один байт).

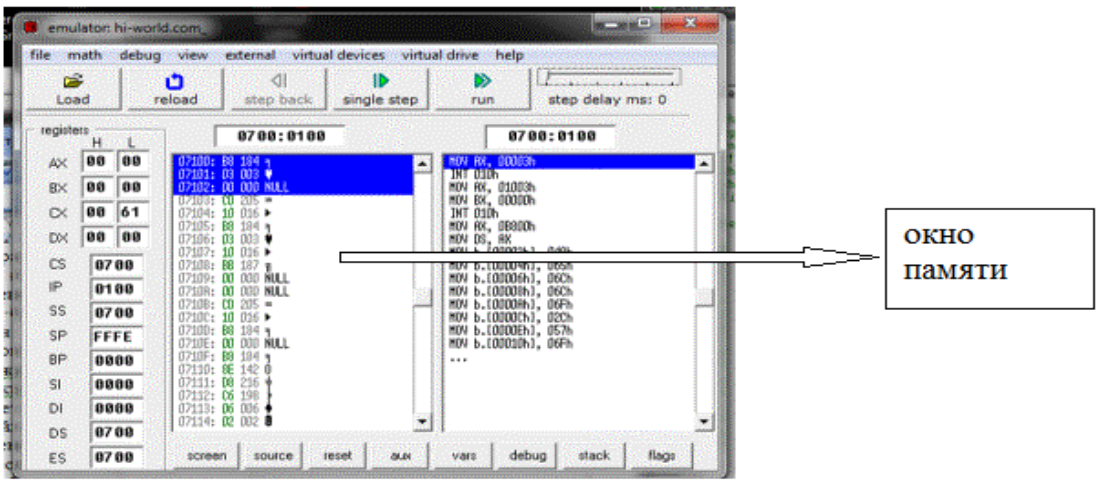
  Команда INT 21H вызывает системную функцию DOS (int от англ. interrupt – прерывание). Прерывание MS–DOS – это своего рода подпрограмма (часть MS–DOS), которая находится постоянно в памяти и может вызываться в любое время из любой программы. Эта команда – основное средство взаимодействия программ с операционной системой. В примере вызывается функция DOS (строка 7) – вывести строку на экран. Эта функция выводит строку от начала, адрес которого задается в регистрах DS: DX, до первого

встречного символа $. При запуске COM–файла регистр DS автоматически загружается сегментным адресом программы, а регистр DX был подготовлен предыдущей командой.

- ***6 строка*** Команда RET пользуется обычно для возвращения из процедуры. DOS вызывается COM–программы так, что команда RET корректно завершает программу.
- ***7 строка*** определяет строку данных, содержащую текст “ Hello ", управляющий символ ASCII возврат каретки с кодом 0Dh, управляющий символ ASCII перевод строки с кодом 0Ah и символ $, завершающий строку. Первое слово (message – сообщение) – название сообщения. Оно может быть любым (например, mess или string и пр). Управляющие символы (0Dh и 0Ah) переводят курсор на первую позицию следующей строки.
- ***8 строка*** директива END завершает программу, одновременно указывая, с какой метки должно начинаться ее выполнение.

4. Щелкните **кнопку [emulate]** (или нажмите клавишу **F5**).

5. Щелкните **кнопку [Single Step] (пошаговый режим)** (или нажмите клавишу **F8**), и наблюдайте за выполнением кода.



Кнопка [Single Step] выполняет команды, один за другим останавливающие после каждой команды.

Кнопка [Run] выполняет команды один за другим с задержкой, установленной задержкой шага между командами.

6. Дважды щелкните на текстовых полях регистра – открывается **окно "Extended Viewer"** со **значением того регистра**, преобразованного ко всем возможным формам. Вы можете изменять значение регистра непосредственно в этом окне.

7. Дважды щелкните на элементе списка памяти – открывается **"Extended Viewer"** со **значением WORD, загруженным** со списка **памяти** в выбранном местоположении.

Менее существенный байт – в младшем адресе: LOW BYTE загружен от выбранной позиции и HIGH BYTE от следующего адреса памяти. Можно изменять значение слова памяти непосредственно в окне "Extended Viewer", можно изменять значения регистров во времени выполнения, печатая по существующим значениям.

Кнопка [Flags] позволяет рассматривать и изменять флажки на времени выполнения.

В окне эмулятора вы можете запустить вашу программу на выполнение целиком (кнопка RUN) либо в пошаговом режиме (кнопка SINGLE STEP). Пошаговый режим удобен для отладки. Ну а мы сейчас запустим программу на выполнение кнопкой RUN.

После этого (если вы не сделали ошибок в тексте программы) вы увидите сообщение о завершении программы (рисунок 4). Здесь вам сообщают о том, что программа передала управление операционной системе, то есть программа была успешно завершена. Нажмите кнопку ОК в этом окне и вы увидите, наконец, результат работы вашей первой программы на языке ассемблера (рисунок 5).

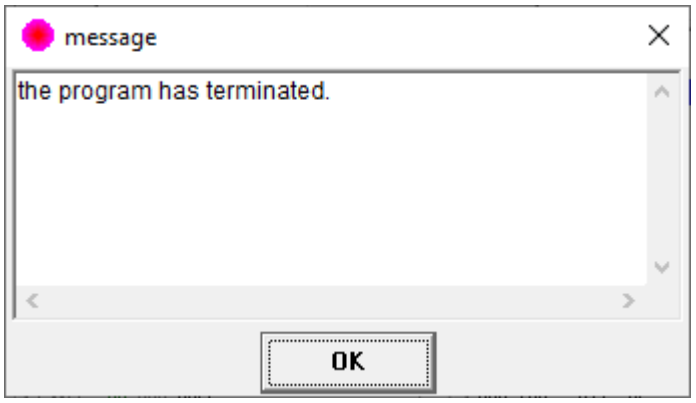


Рисунок 4 – Сообщение о завершении программы

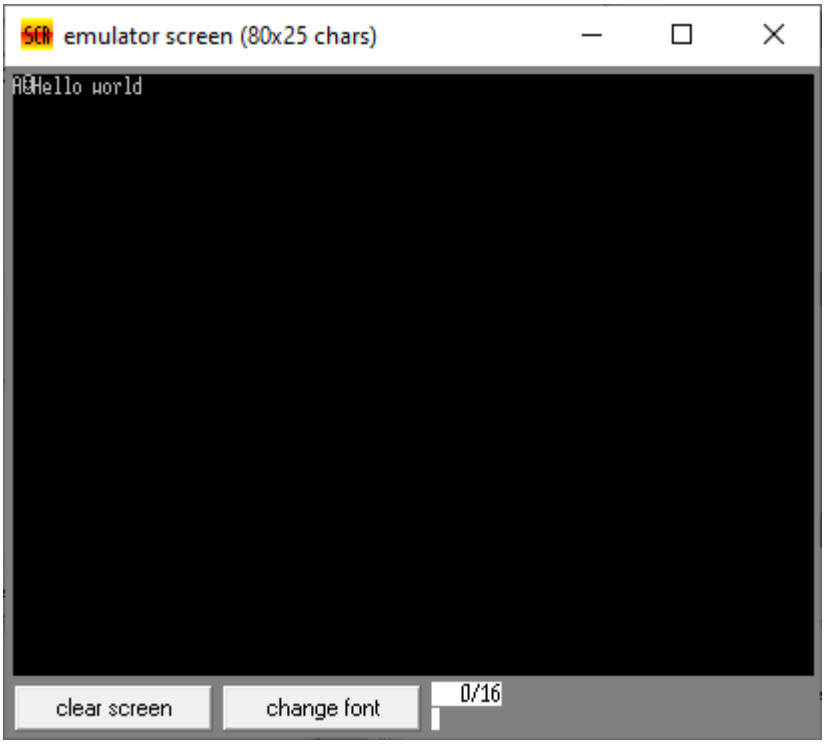


Рисунок 5 – Результат работы программы

Пример 2.

Создание программ, производящих сложение и вычитание.

1. Щелкните на кнопку New, выберите Exe и введите в строке «add your code here» следующие команды для расчета выражения:

((5 + (AL – AH)) – BH) + BL где AL=9, BL=3, AH=4, BH=6.

```
mov al, 9
mov ah, 4
sub al, ah
mov dl, 5
add al, dl
mov bh, 6
sub al, bh
mov bl, 3
add al, bl
```

Для вывода результирующего значения в 10–ричной системе счисления добавьте следующий код:

```
mov bx, 0
mov bl, al
 mov ax, bx
push -1
mov cx,10
1:mov dx,0
div cx
push dx
cmp ax,0
jne 1
mov ah,2h
12:pop dx
cmp dx,-1
je ex
add dl,'0'
int 21h
jmp 12
ex:mov ax,4c00h
int 21h
end start
```

Практическая часть

Задание 1.

Произвести расчет выражений и исследование состояния регистров и памяти при выполнении данных команд:

Вариант 1

1) (№ + AL) – (BH + AH) – BL

Вариант 2

2) ((BH + (№ – AH)) – BL) + AL

где № – порядковый номер по журналу, AL=9, BL=3, AH=4, BH=6.

Задание 2.

Зарегистрировать изменения состояния процессора и памяти в форме таблицы для небольшого кода, выполняющего от 3–5 операций (код придумать самостоятельно).

Таблица 1

Состояние регистров	Окно памяти	Исходный код

Контрольные вопросы:

1. Какова структура ассемблерной программы?
2. Опишите основные фазы командного цикла.
3. Дайте характеристику структуры файла типа *.com?
4. С какой целью в код программы на ассемблере для DOS вводится строка ORG 100h?
5. Опишите назначение прерываний 21h и 20h.
6. Опишете правила применения связки «BEGIN: – END BEGIN».
7. Опишете режимы работы Emu8086.

РАЗДЕЛ VIII. Организация памяти в ЭВМ

Тема 8.2. Оперативные и постоянные запоминающие устройства

ЛАБОРАТОРНАЯ РАБОТА №8

Исследование работы ячейки памяти статического ОЗУ экспериментальным путем

Цель: Изучить принципы работы и архитектуру микропроцессора. Выполнить анализ характеристик микропроцессора

Порядок выполнения

1. Изучить теоретическую часть
2. Выполнить практическое задание
3. Ответить на контрольные вопросы
4. Оформить отчет по проделанной работе

Теоретическая часть

Память и процессор

Среди устройств и узлов, входящих в состав компьютера, наиболее важными для выполнения любой программы катаются оперативная память и центральный микропроцессор, который мы для краткости будем в дальнейшем называть просто процессором.

В оперативной памяти хранится выполняемая программа вместе с принадлежащими ей данными; процессор выполняет вычисления и другие действия, описанные в программе.

Программа загружается в память с жесткого диска, где она хранится, операционной системой в ответ ввод с клавиатуры команды запуска программы. Операционная система, загрузив программу, и при необходимости настроив ее для выполнения в той области памяти, куда она попала, сообщает процессору начальный адрес загруженной программы и инициирует процесс ее выполнения. Процессор считывает из памяти первую команду программы, находит в памяти или в своих регистрах данные, необходимые для ее выполнения (если, конечно, команда требует данных) и, выполнив требуемую операцию, возвращает в память или, возможно, оставляет в регистрах результат своей работы (рисунок 1).

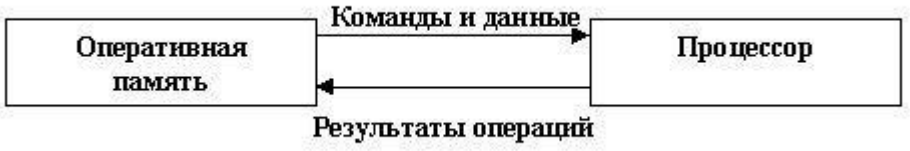


Рисунок 1 – Взаимодействие оперативной памяти и процессора

Выполнив первую команду, процессор переходит к следующей, и так далее до конца программы. Завершив программу, процессор не будет знать, что ему дальше делать, поэтому любая программа должна завершаться командами, передающими управление операционной системе компьютера.

Оперативная память компьютера представляет собой электронное устройство, состоящее из большого числа двоичных запоминающих элементов, а также схем управления ими. Минимальный объем информации, к которому имеется доступ в памяти, составляет один байт (8 двоичных разрядов, или битов).

Все байты оперативной памяти нумеруются, начиная с нуля. Нужные байты отыскиваются в памяти по их номерам, выполняющим функции адресов.

Некоторые данные (например, коды символов) требуют для своего хранения одного байта; для других данных этого места на хватает, и под них в памяти выделяется 2, 4, 8 или еще большее число байтов. Обычно пары байтов называют словами, а четверки – двойными словами (рисунок 2), хотя иногда термином "слово" обозначают любую порцию машинной информации.

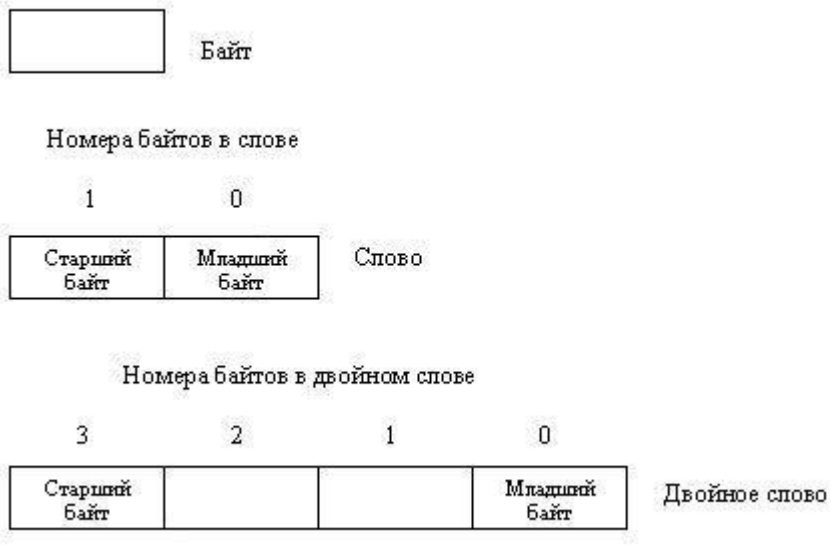


Рисунок 2 – Байт, слово и двойное слово

При обсуждении содержимого многобайтового данного приходится ссылаться на составляющие его байты; эти байты условно нумеруются от нуля и располагаются (при их изображении на бумаге) в порядке возрастания номера справа налево, так что слева оказываются байты с большими номерами, а справа - байты с меньшими номерами. Крайний слева байт принято называть старшим, а крайний справа - младшим. Такой порядок расположения байтов связан с привычной для нас формой записи чисел: в многоразрядном числе слева указываются старшие разряды, а справа - младшие. Следующее число, если его написать за предыдущим, опять начнется со старшего разряда и закончится младшим. Однако в памяти компьютера данные располагаются в более естественном порядке непрерывного возрастания номеров байтов и, таким образом, каждое слово или двойное слово в памяти начинается с его младшего байта и заканчивается старшим (рисунок 3).

Рисунок 3 – Нумерация байтов в многобайтовых данных

Строго говоря, в памяти компьютера можно хранить только целые двоичные числа, так как память состоит из двоичных запоминающих элементов. Для записи иных данных, например, символов или дробных чисел, для них предусматриваются правила кодирования, т.е. представления в виде последовательности битов той или длины. Так, действительное число одинарной точности занимает в памяти двойное слово (32 бит), в котором 23 бит отводятся под мантиссу, 8 бит под порядок и еще один бит под знак числа. Программы, работающие с такого рода данными, должны, естественно, знать правила их записи и руководствоваться ими при обработке и представлении этих данных.

Двоичная система счисления, в которой работают все цифровые электронные устройства, неудобна для человека. Для удобства представления двоичного содержимого ячеек памяти или регистров процессора используют иногда восьмеричную, а чаще - шестнадцатеричную системы счисления. Для процессоров Intel используется шестнадцатеричная система.

Каждый разряд шестнадцатеричного числа может принимать 16 значений, из которых первые 10 обозначаются обычными десятичными цифрами, а последние 6 - буквами латинского алфавита от A до F, где A обозначает 10, B - И, C - 12, D - 13, E - 14, а F - 15. В языке ассемблера шестнадцатеричные числа, чтобы отличать их от десятичных, завершаются буквой h (или H). Таким образом, 100 – это десятичное число, а l00h -

шестнадцатеричное (равное 256). Поскольку одна шестнадцатеричная цифра требует для записи ее в память компьютера четырех двоичных разрядов, то содержимое байта описывается двумя шестнадцатеричными цифрами (от 00h до FFh, или от 0 до 255) , а содержимое слова - четырьмя (от 0000h до FFFFh, или от 0 до 65535).

Помимо ячеек оперативной памяти, для хранения данных используются еще запоминающие ячейки, расположенные в процессоре и называемые регистрами. Достоинство регистров заключается в их высоком быстродействии, гораздо большем, чем у оперативной памяти, а недостаток в том, что их очень мало – всего около двух десятков. Поэтому регистры используются лишь для кратковременного хранения данных.

Для того, чтобы с помощью 16-разрядных чисел адресовать любой байт памяти, в МП 86 предусмотрена сегментная адресация памяти, реализуемая с помощью сегментных регистров процессора.

Суть сегментной адресации заключается в следующем. Обращение к памяти осуществляется исключительно с помощью сегментов – логических образований, накладываемых на те или иные участки физической памяти. Исполнительный адрес любой ячейки памяти вычисляется процессором путем сложения начального адреса сегмента, в котором располагается эта ячейка, со смещением к ней (в байтах) от начала сегмента (рисунок 4). Это смещение иногда называют относительным адресом.

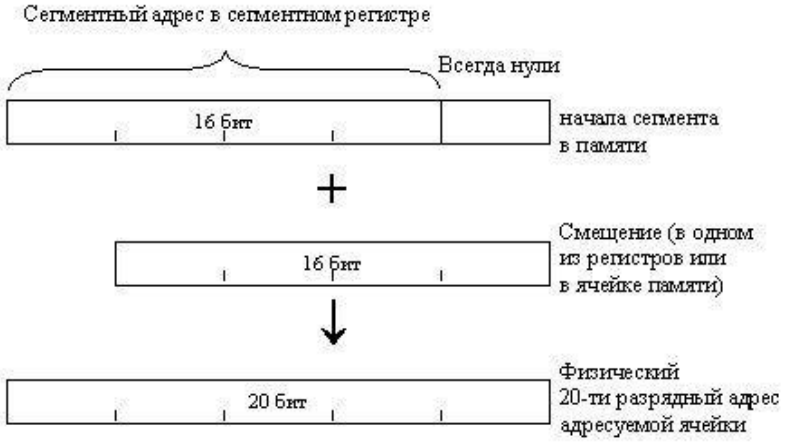


Рисунок 4 – Образование физического адреса из сегментного адреса и смещения

Начальный адрес сегмента без четырех младших битов, т.е. деленный на 16, помещается в один из сегментных регистров и называется сегментным адресом. Сам же начальный адрес хранится в специальном внутреннем регистре процессора, называемом теневым регистром. Для каждого сегментного регистра имеется свой теневой регистр; начальный адрес сегмента загружается в него процессором в тот момент, когда программа заносит в соответствующий сегментный регистр новое значение сегментного адреса.

Процедура умножения сегментного адреса на 16 (или, что то же самое, на 10h) является принципиальной особенностью реального режима, ограничивающей диапазон адресов, доступных в реальном режиме, величиной 1 Мбайт. Действительно, максимальное значение сегментного адреса составляет FFFFh, или 64К-1, из чего следует, что максимальное значение начального адреса сегмента в памяти равно FFFF0h, или 1 Мбайт.

Не следует думать, что термины «адресное пространство» и «оперативная память» эквивалентны.

Адресное пространство – это просто набор адресов, которые умеет формировать процессор; совсем не обязательно все эти адреса отвечают реально существующим ячейкам памяти. В зависимости от модификации персонального компьютера и состава его периферийного оборудования, распределение адресного пространства может несколько различаться. Тем не менее, размещение основных компонентов системы довольно строго унифицировано. Типичная схема использования адресного пространства компьютера приведена на рисунке 5.

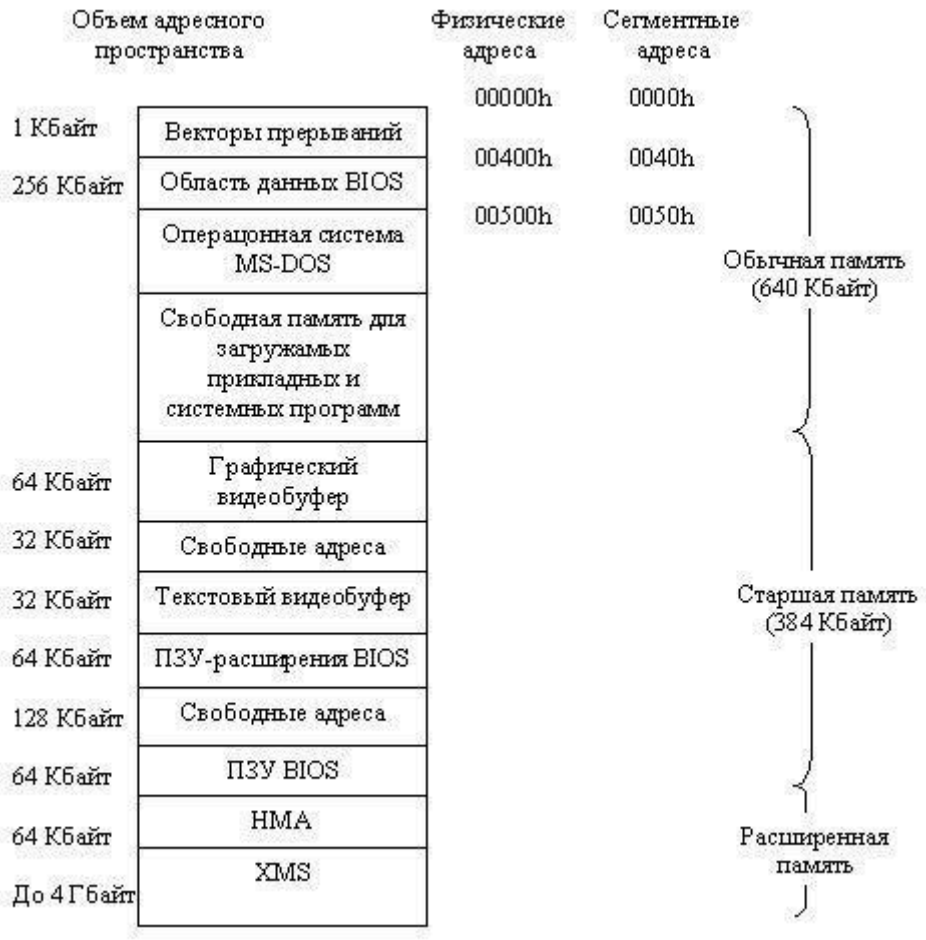


Рисунок 5 – Распределение адресного пространства

Первые 640 Кбайт адресного пространства с адресами от 00000h до 9FFFF11 (и, соответственно, с сегментными адресами от 0000h до 9FFFh) отводятся под основную оперативную память, которую еще называют стандартной (conventional). Начальный килобайт оперативной памяти занят векторами прерываний, которые обеспечивают работу системы прерываний компьютера, и включает 256 векторов по 4 байта каждый. Вслед за векторами прерываний располагается так называемая область данных BIOS, которая занимает всего 256 байт, начиная с сегментного адреса 40h. Сама BIOS (от Basic In-Out System, базовая система ввода-вывода) является частью операционной системы, хранящейся в постоянном запоминающем устройстве. Это запоминающее устройство (ПЗУ BIOS) располагается на системной плате компьютера и является, таким образом, примером встроенного, или

«зашитого» программного обеспечения. В функции BIOS входит тестирование компьютера при его включении, загрузка в оперативную память собственно операционной системы MS-DOS, хранящейся на магнитных дисках, а также управление штатной аппаратурой компьютера - клавиатурой, экраном, дисками и прочим. В области данных BIOS хранятся разнообразные данные, используемые программами BIOS в своей работе.

Так, здесь размещаются:

входной буфер клавиатуры, куда поступают коды нажимаемых пользователем клавиш;

адреса видеоадаптера, а также последовательных и параллельных портов;

данные, характеризующие текущее состояние видеосистемы (форма курсора и его текущее положение на экране, видеорежим, используемая видеостраница и проч.);\

ячейки для отсчета текущего времени и т.д.

Область данных BIOS заполняется информацией в процессе начальной загрузки компьютера, а затем динамически модифицируется системой по мере необходимости. Многие прикладные программы, особенно, написанные на языке ассемблера, обращаются к этой области с целью чтения или модификации содержащихся в них данных.

Практические задания:

Задание 1

В отчёте расписать 6 операций с двумя разными наборами операндов (код взять из лабораторной работы №7). Результаты работы оформить по таблице 1. Указать какие данные хранятся в ячейках памяти.

Таблица 1

Код операции	Операция	Операнд А	Операнд Б	Результат работы

Задание 2

Определить в каких строках кода программы осуществляется работа со значениями ячеек памяти. Записать чему будут равны значения регистров процессора и ячеек памяти после выполнения следующего кода:

```
mov   ax, 6B4Fh
lea   bx, val
mov   [bx], ax
mov   ah, 6
mov   dl,[bx]
int   21h
mov   dl,[bx+1]
int   21h
mov   ah, 4Ch
int   21h
```

Задание 3

Приведите 5 примеров команд, в которых одним из операндов является ячейка памяти. Укажите чему равно ее значение после выполнения команды.

Контрольные вопросы:

1. Дайте определение понятию «Архитектура процессора».
2. Дайте определение понятию «Адресное пространство».
3. Опишите назначение ячеек памяти.
4. Опишите различие в хранении данных в ячейках памяти и регистрах процессора.

Тема 10.2. Форматы команд, способы адресации, система команд однокристальных микропроцессоров

ЛАБОРАТОРНАЯ РАБОТА №9

Разработка и отладка программы с использованием различных форматов команд и методов адресации. Исследование командного цикла МП при выполнении команд с различными форматами и методами адресации

Цель: Изучить принципы построения различных форматов команд и используемых в ассемблере методов адресации.

Порядок выполнения

1. Изучить теоретическую часть
2. Выполнить практическое задание
3. Ответить на контрольные вопросы
4. Оформить отчет по проделанной работе

Теоретическая часть

Команда – это утверждение, которое выполняется процессором во время работы программы. Команда ассемблера ВСЕГДА генерирует машинный код в отличие от директивы. Команды могут быть нескольких типов: передачи управления, передачи данных, арифметические, логические и ввода/вывода. Команды транслируются в машинные коды.

Классификация команд МП представлена на рисунке 1.

Рисунок 1 – Классификация команд МП

С функциональной точки зрения команды разделяются на три большие группы:

- **команды передачи данных** (обмен входами между регистрами процессора, процессора и оперативной памятью, процессора и периферийными установками)
- **команды обработки данных** (команды сложения, умножения, сдвига, сравнения-).
- **команды передачи управления** (команды безусловного и условного перехода).

Рассмотрим подробно основные команды, применяемые в МП, пользуясь классификацией по функциональным признакам. Названия команд

обозначим русскими словами, указывающими на смысл выполняемых операций.

1. Команды передачи данных обеспечивают простую пересылку информации без выполнения каких-либо операций обработки. Команды этой группы делятся на команды связанные с обращением к памяти, команды обращения к регистрам и команды ввода вывода.

2. Команды управления, часто называемые командами перехода, позволяют выполнять различные действия в соответствии со значением внешних сигналов или выработанных внутри системы условий.

Группа команд передачи управления обеспечивает принудительное изменение порядка выполнения команд в программе.

Все команды управления делятся на команды безусловного и условного перехода.

3. Команды обработки данных делятся на арифметические и логические. К арифметическим относятся:

СЛОЖИТЬ содержимое двух регистров или регистра и ячейки памяти;

ВЫЧЕСТЬ из содержимого ячейки памяти или регистра содержимое регистра;

УВЕЛИЧИТЬ НА 1 (ИНКРЕМЕНТ) содержимое ячейки памяти или регистра (указателя стека, индексного регистра, аккумулятора);

УМЕНЬШИТЬ НА 1 (ДЕКРЕМЕНТ) содержимое ячейки памяти или регистра;

СЛОЖИТЬ С УЧЕТОМ ПЕРЕНОСА, по которой выполняется сложение с учетом состояния триггера переноса. Это позволяет легко организовать обработку чисел большой длины;

ВЫЧЕСТЬ С УЧЕТОМ ЗАЕМА;

СДВИГ содержимого ячейки памяти или регистра (обычно на один разряд).

В подгруппу логических команд входят команды:

И (ЛОГИЧЕСКОЕ УМНОЖИТЬ), по которой выполняется операция конъюнкции между содержимым двух регистров или ячейки памяти и регистра;

ИЛИ (ЛОГИЧЕСКИ СЛОЖИТЬ), по которой выполняется операция дизъюнкции между содержимым двух регистров или ячейки памяти и регистра;

НЕРАВНОЗНАЧНОСТЬ, по которой производится поразрядное сравнение содержимого двух регистров или ячейки памяти и регистра;

ИНВЕРСИЯ содержимого ячейки памяти или регистра.

необходимых для размещения одной команды, различают команды длиной в одно, два или три слова. Команды длиной в два или три слова требуют для выборки соответственно два или три цикла обращения к памяти.

Во многих случаях, в частности при сравнении МП со сходной архитектурой, оказывается полезной классификация команд в соответствии с архитектурными характеристиками МП.

К командам связанным с обращением к памяти относятся:

ЗАГРУЗИТЬ (ПРОЧИТАТЬ), по которой содержимое одной из ячеек памяти засылается в регистр;

ЗАПОМНИТЬ (ЗАПИСАТЬ), по которой содержимое регистра засылается в ячейку памяти.

В командах, связанных с пересылкой байта или слова, должны указываться номер конкретного регистра, адрес ячейки памяти и, если необходимо, номер модуля ЗУ.

Команды, связанные с обращением к регистрам, должны указывать номер источника информации и номер регистра результата. В эту подгруппу команд передачи данных входят команды:

ЗАГРУЗИТЬ НЕПОСРЕДСТВЕННО, по которой в регистр записывается константа, указанная в коде команды;

ПЕРЕСЛАТЬ, по которой содержимое одного регистра пересылается в другой.

К командам ввода-вывода относятся:

ВВОД, по которой содержимое устройства ввода засылается во внутренний регистр МП;

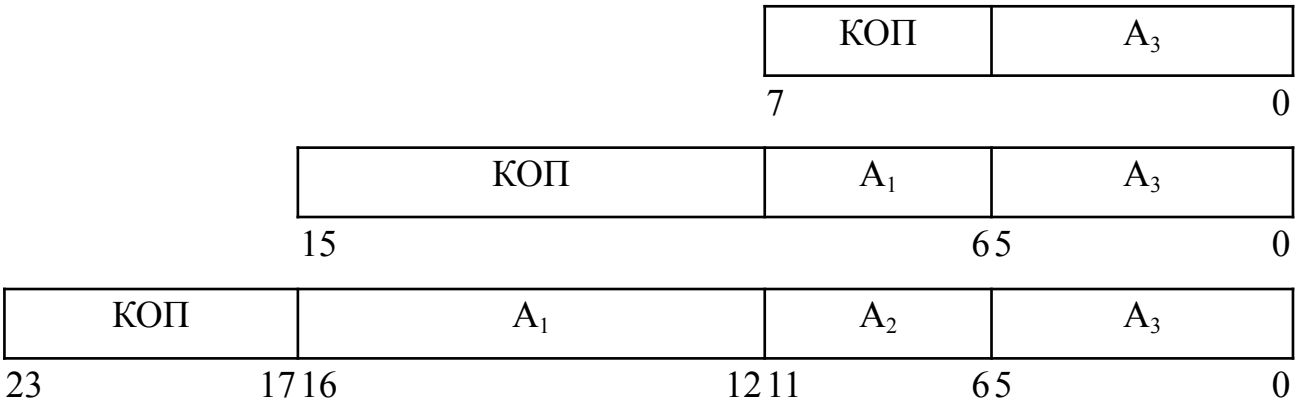
ВЫВОД, по которой содержимое внутреннего регистра МП (обычно аккумулятора) пересылается в устройство вывода.

Формат команд микропроцессора

Алгоритм, написанный пользователем программы, в конечном счете реализуется в виде машинных команд.

Под **командой** понимают совокупность сведений, представленных в виде двоичных кодов, необходимых процессору для выполнения очередного шага.

В ходе команды для сведений о типе операции, адресной информации о нахождении обрабатываемых данных, а также для информации о месте хранения результатов выделяются определенные разряды (поля).



Форматом команды называется заранее обговоренная структура полей в её кодах, позволяющая ЭВМ распознавать составные части кода.

Главным элементом кода команды является **код операции (КОП),** что определяет, какие действия будут выполнены по данной команде. Под него выделяется N старших разрядов формата. В остальных разрядах размещаются A_1 и A_2 – адреса операндов. A_3 – адрес результата.

Распределение полей в формате команды может изменяться при смене способа адресации. Длина команды зависит от числа адресных полей.

По числу адресов <u>команды</u> делятся на:

- безадресные

- одно-,
- двух-,
- трехадресные

Длина кода команды измеряется в машинных словах. Чтобы получить возможность работать с минимальным числом адресных полей, результат, к примеру, можно размещать по месту хранения одного из операндов. Либо предварительно размещают один или несколько операндов в специально выделенных регистрах процессора.

Множество реализуемых машинных действий образует её **систему команд**. Система команд часто определяет области и эффективность применения ЭВМ. Состав и число команд должны быть ориентированы на стандартный набор операций, используемых пользователем для решения своих задач.

Оттранслированные команды записываются в соседние ячейки памяти в порядке их следования в программе.

При естественном порядке выполнения команд в программе, адрес каждой следующей команды определяется по содержимому специального счетчика команд, который входит в состав процессора. Содержимое этого счетчика автоматически наращивается на 1 при выполнении очередной команды.

При организации ветвления цикла или для перехода на подпрограмму в счетчик в счетчик команд принудительно записывается адрес перехода, указанный в ходе команды.

Большинство алгоритмов может быть реализовано небольшим базовым набором команд. Вместе с тем система команд должна быть полной, т.е. содержать все команды, которые необходимы для интерпретации алгоритма в машинных кодах. ЭВМ общего назначения имеет универсальный набор команд и применяется в основном для решения тривиальных (стандартных) задач.

В команде, как правило, содержатся не сами операнды, а информация объект адресах ячеек памяти или регистрах, в которых они находятся. Код команды можно представить состоящим из нескольких полей, каждое из которых имеет свое функциональное назначение.

В общем случае команда состоит из:

- операционной части (содержит код операции);
- адресной части (содержит адресную информацию о местонахождении обрабатываемых данных и месте хранения результатов).

В свою очередь, эти части, что особенно характерно для адресной части, могут состоять из нескольких полей.

Структура команды определяется составом, назначением и расположением полей в коде.

Форматом команды называется заранее оговоренная структура полей ее кода с разметкой номеров разрядов (бит), определяющих границы отдельных полей команды, или с указанием числа разрядов (бит) в определенных полях, позволяющая ЭВМ распознавать составные части кода.

Пример формата команды процессора i486

<table>
<tr><td rowspan="2">КОП</td><td colspan="4">Байты адресации</td><td rowspan="2">Смещение</td><td rowspan="2">Операнд</td></tr>
<tr><td rowspan="2">mod r/m</td><td colspan="3">sib</td></tr>
<tr><td>SS</td><td>index</td><td>base</td><td></td><td></td></tr>
<tr><td>1 или 2 байта</td><td colspan="4">0 или 1 байт</td><td>0,1,2, или 4 байта</td><td>0,1,2, или 4 байта</td></tr>
</table>

КОП — код операции;

mod r/m — спецификатор режима адресации;

r/m — регистр памяти;

SS — масштабный множитель для режима масштабирования индексной адресации;

index — определяет индексный регистр;

base — определяет базовый регистр.

Способы адресации

Способы адресации классифицируют:

- по наличию адресной информации в команде (явная и неявная адресация).
- по кратности обращения в оперативную память.
- по способу формирования адресов ячеек памяти.

По наличию адресной информации в команде

При **явной** адресации операнда в команде есть поле адреса этого операнда. При **неявной** адресное поле в команде отсутствует, а адрес операнда подразумевается кодом операции. Например, из команды может быть исключен адрес приемника адресата, при этом подразумевается, что результат записывается на месте второго операнда.

По кратности обращения в оперативную память различают:

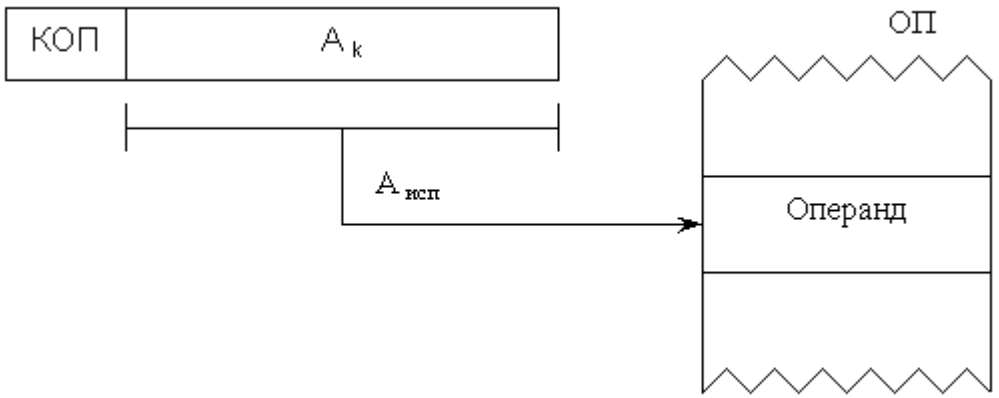
- непосредственную адресацию (direct addressing)
- прямую адресацию (immediate addressing)
- косвенную адресацию (indirect addressing)

Непосредственная адресация

При непосредственной адресации операнд располагается непосредственно в адресном поле команды.

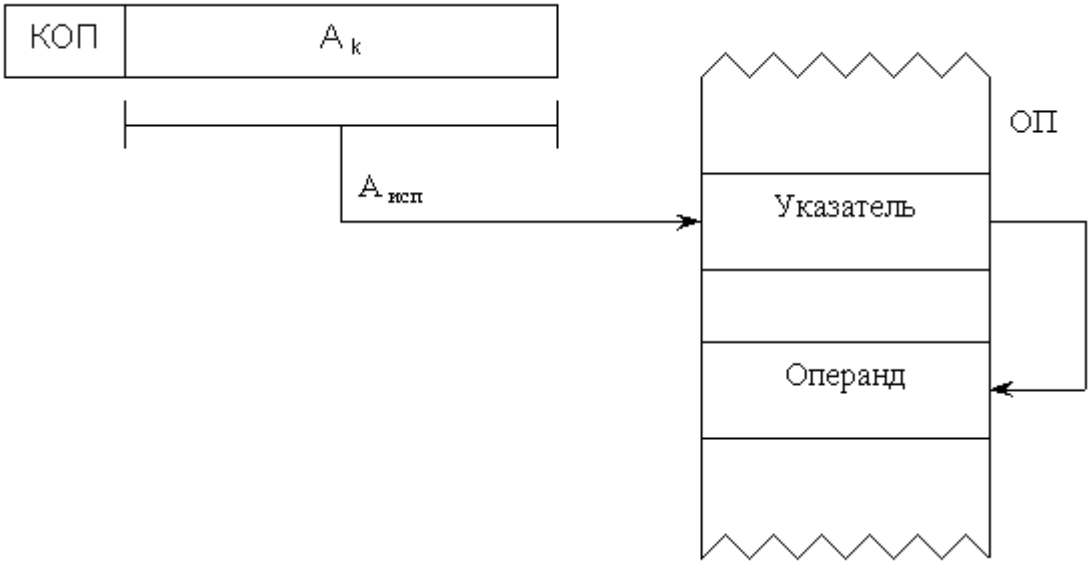
Прямая адресация

При прямой адресации обращение за операндом производится по адресному коду в поле команды. При этом исполнительный адрес совпадает с адресом кода команды.



Косвенная адресация

При косвенной адресации код команды указывает адрес ячейки памяти, в которой находится не сам операнд, а его адрес, называемый **указателем**.



Способы формирования адресов ячеек памяти можно разделить на:

- **Абсолютные способы формирования** предполагают, что двоичный код адреса ячейки памяти может быть целиком извлечен либо из адресного поля команды, либо из какой-нибудь другой ячейки в случае косвенной адресации.
- **Относительные способы формирования** предполагают, что двоичный код адресной ячейки памяти образуется из нескольких составляющих:

Б – код базы,

И – код индекса,

С – код смещения.

Эти составляющие используются в различных сочетаниях.

Относительная адресация

При относительной адресации применяется способ вычисления адреса путем суммирования кодов, составляющих адрес.

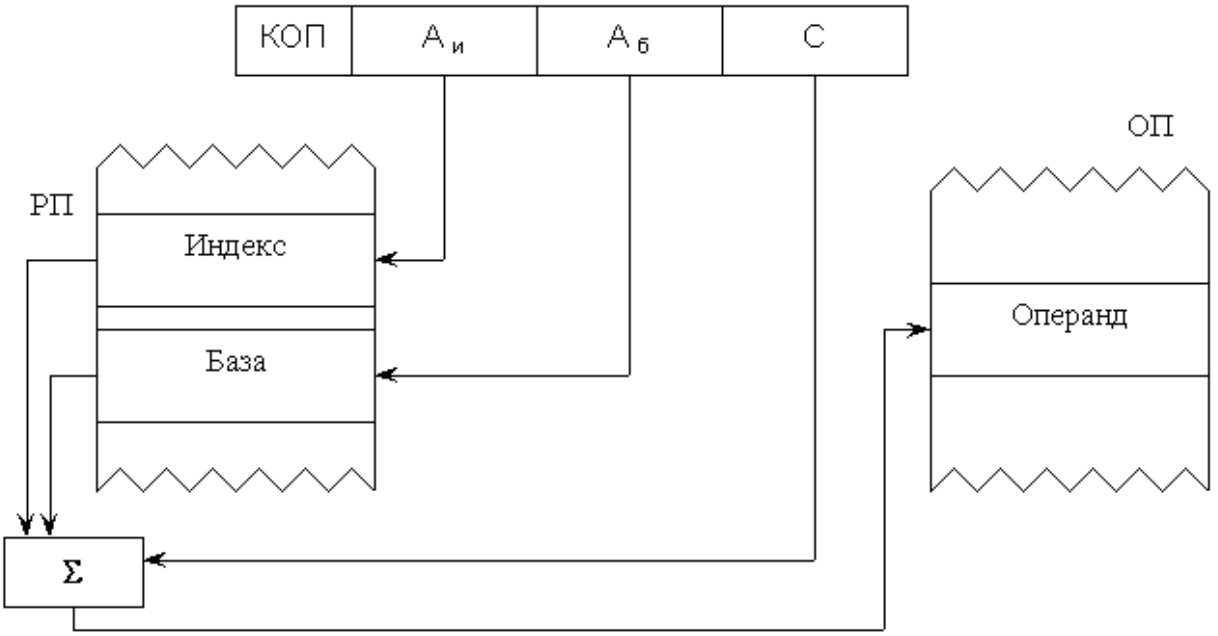
А = Б + И + С

А = Б + С

А = И + С

Индексная адресация

Для работы программ с массивами, требующими однотипных операций над элементами массива, удобно использовать индексную адресацию.



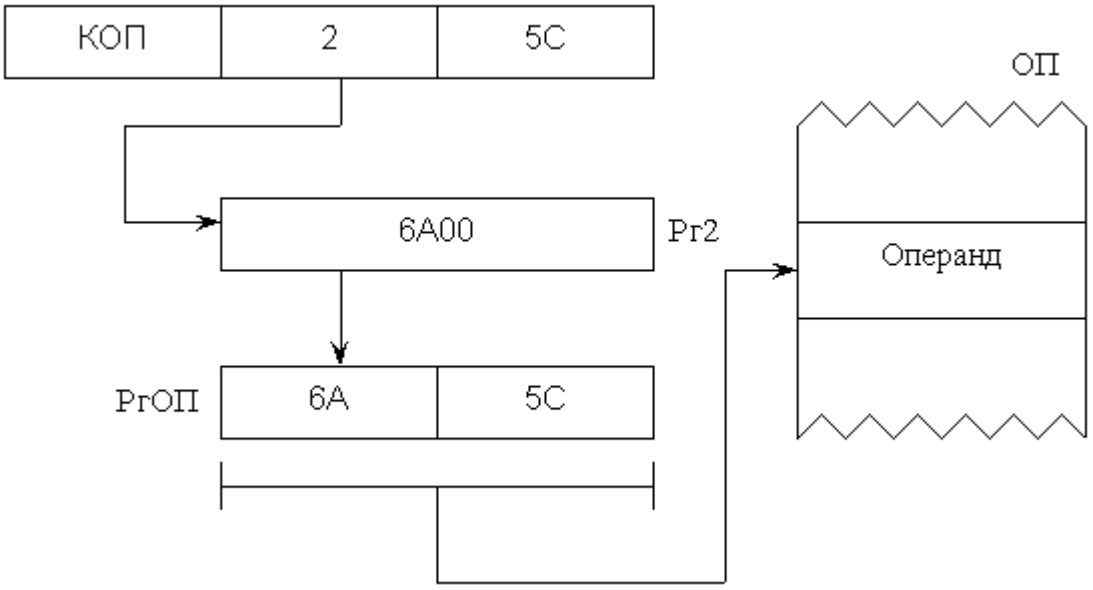
Адрес i-того операнда в массиве определяется как сумма начального адреса массива операнда, задаваемого смещением S, и индекса I , записанного

в одном из регистров регистровой памяти, называемым **индексным регистром**.

Адрес индексного регистра задается в команде полем адреса индекса $A_и$.

В каждом i-том цикле содержимое индексного регистра изменяется на постоянную величину, как правило, это 1.

В некоторых моделях ЭВМ относительная адресация выполняется без суммирования по следующей схеме:

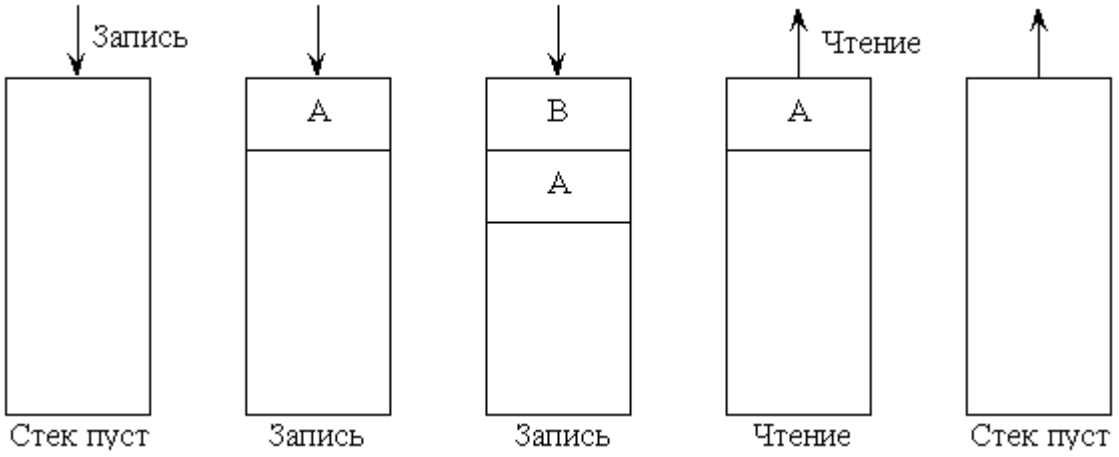


Автоиндексная адресация

При автоиндексации косвенный адрес, находящийся в регистре РП, автоматически увеличивается (автоинкрементная адресация), или уменьшается (автодекрементная адресация) на постоянную величину до или после выполнения операции.

Стековая адресация

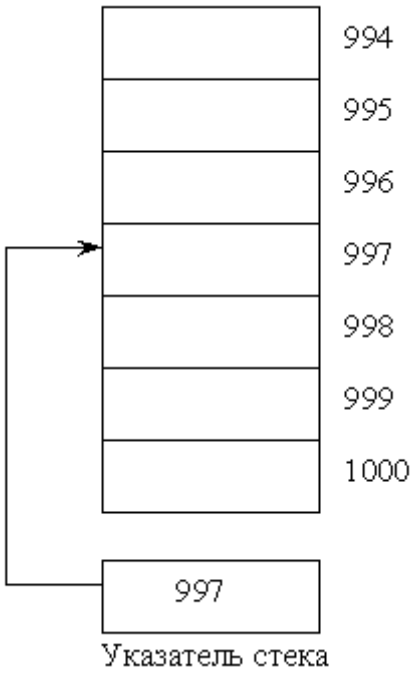
Стековая память широко используется в современных ЭВМ. Хотя адрес обращения в стек отсутствует в команде, он формируется схемой управления:



Для чтения записи доступен только один регистр v вершина стека. Этот способ адресации используется, в частности, системой прерывания программ при вложенных вызовах подпрограмм.

Стековая память реализуется на основе обычной памяти с использованием указателя стека и автоиндексной адресации.

Запись в стек производится с использованием автодекрементной адресации, а чтение – с использованием автоинкрементной адресации.



Примеры:

```
mov al,10;          загружаем в регистр AL число 10
add al,15;          al = 25; al – приемник, 15 – источник

mov al,10
sub al,7;           al = 3; al – приемник, 7 – источник
mov ax,25000
sub ax,10000;       ax = 15000; ax - приемник, 10000 - источник
```

Практические задания:

Задание 1.

Рассмотрите приведенные примеры команд. Определите какой тип адресации был использован, каково количество адресов команды (безадресные, одно-или двух-адресные).

Задание 2.

Запишите 20 команд с различным типом адресации, добавив комментарий, что выполнят команда, результат ее работы, а также какой тип адресации был использован.

Задание 3.

```
B DB 'a'
W DW 2
D DD 1
Q DQ 3
```

Среди перечисленных команд указать неправильные; объяснить, в чём ошибка.

a) MOV B, AX

б) MOV W, AX

в) MOV AX, W

г) XCHG ESI, B–1

д) MOV EAX, BL

е) MOV B, 80h

ж) XCHG AH, AL

з) MOV SI, W+1

и) MOV B, W–B

к) XCHG D, ECX

л) MOV Q, word ptr 2

м) MOV dword ptr B, 12345h

Контрольные вопросы:

1. Дайте определения понятиям: «команда», «система команд МП», «исполнительный адрес», «адресация», «формат команд».
2. Охарактеризуйте формат команд по числу адресов команды.
3. Охарактеризуйте способы адресации по наличию адресной информации в команде.
4. Охарактеризуйте способы адресации по кратности обращения в оперативную память.
5. Охарактеризуйте способы адресации по способу формирования адресов ячеек памяти.
6. В чем заключается особенность стековой адресации?

Раздел X. Универсальные микропроцессоры

Тема 10.2. Форматы команд, способы адресации, система команд однокристальных микропроцессоров

ЛАБОРАТОРНАЯ РАБОТА №10

Разработка и отладка программы с использованием арифметических команд. Исследование командного цикла МП при выполнении арифметических команд

Цель: Изучить процесс разработки и отладки программ с использованием арифметических команд

Порядок выполнения

1. Изучить теоретическую часть
2. Разработать алгоритм задачи (согласно варианту)
3. Выполнить практические примеры
4. Реализовать полученный алгоритм на языке Ассемблера
5. Выполнить тестирование программы
6. Ответить на контрольные вопросы
7. Оформить отчет по проделанной работе

Теоретическая часть

Арифметические команды

Рассмотрим классификацию арифметических команд ассемблера (рисунок 1).

После выполнения все арифметические команды изменяются ФЛАГИ, по которым можно определить характеристики результата:

1. Флаг *CF* устанавливается, если при сложении произошёл перенос из старшего разряда. Для беззнаковых чисел это будет означать, что произошло переполнение и результат получился некорректным.
2. Флаг *OF* обозначает переполнение для чисел со знаком.
3. Флаг *SF* равен знаковому биту результата (для чисел со знаком, а для беззнаковых он равен старшему биту и особо смысла не имеет).
4. Флаг *ZF* устанавливается, если результат равен 0.

5. Флаг *PF* – признак чётности, равен 1, если результат содержит нечётное число единиц.

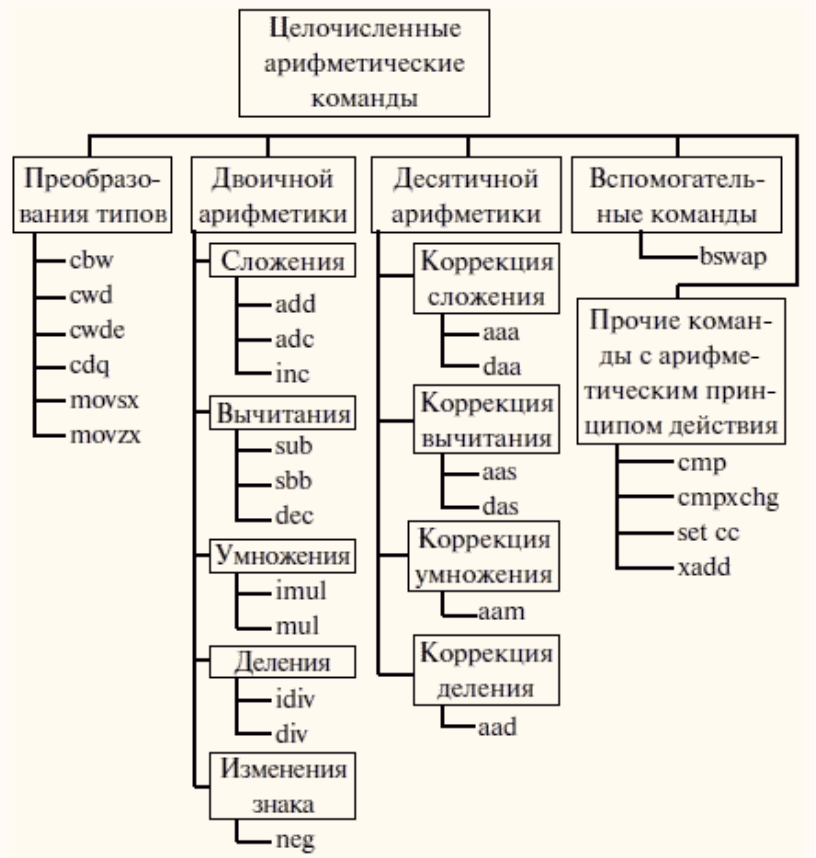


Рисунок 1 – Классификация арифметических команд

ADD – команда для сложения двух чисел (работает как с числами со знаком, так и без знака).

ADD *Приемник*, *Источник*

Логика работы команды:

<*Приемник*> = <*Приемник*> + <*Источник*>

- Операнды должны иметь одинаковый размер.
- Результат помещается на место первого операнда (предыдущее значение теряется).
- Возможны сочетания:

  регистр – регистр

  память – регистр

регистр – память

регистр – непосредственный операнд

непосредственный операнд – регистр

- Операция память - память выполняется через промежуточный регистр

Примеры:

1) add ax,5 ;AX = AX + 5
2) add dx,cx ;DX = DX + CX
3) add dx,cl ;Ошибка: разный размер операндов

Особые ситуации, которые могут возникать при сложении:

1) значение результата превышает размерности поля операнда для фиксирования ситуации выхода за разрядную сетку результата в микропроцессоре предусмотрено специальное средство: флаг переноса CF (curry flag). Он располагается в бите 0 регистра флагов eflags/flags. Именно установкой этого флага фиксируется факт переноса единицы из старшего разряда операнда. (CF=1)

Например, при сложении операндов размером в байт результат не должен превышать число 255. Если это происходит, то результат оказывается неверным.

Например, выполним сложение: 254 + 5 = 259 в двоичном виде. 11111110 + 0000101 = 1 00000011. Результат вышел за пределы восьми бит и правильное его значение укладывается в 9 бит, а в 8-битовом поле операнда осталось значение 3.

Т.е. если при сложении двух 8-битовых чисел результат занимает 9 битов, то значение старшего 9 бита запоминается во флажке CF.

Необходимо предусматривать возможность такого исхода операции сложения – включать участки кода после операции сложения, в которых анализируется флаг CF.

2) При сложении чисел со знаком может произойти другая особая ситуация: результат выходит из диапазона допустимых значений

Переполнение регистрируется с помощью флага переполнения OF. Дополнительно к флагу OF при переносе из старшего разряда устанавливается в 1 и флаг переноса CF.

SUB – команда для вычитания одного числа из другого (работает как с числами со знаком, так и без знака).

SUB *Приемник*, *Источник*

Логика работы команды:

<Приемник> = *<Приемник>* - *<Источник>*

- Операнды должны иметь одинаковый размер.
- Результат помещается на место первого операнда.
- Возможны сочетания:

  регистр – регистр

  память – регистр

  регистр – память

  регистр – непосредственный операнд

  непосредственный операнд – регистр
- Операция память - память выполняется через промежуточный регистр
- После команды вычитания чисел без знака нужно анализировать состояние флага CF. Если он установлен в 1, то это говорит о том, что произошел заем из старшего разряда и результат получился в дополнительном коде.

На самом деле вычитание в процессоре реализовано с помощью сложения. Процессор меняет знак второго операнда на противоположный, а затем складывает два числа.

Примеры:

1) sub ax,13 ;AX = AX - 13
2) sub ax,bx ;AX = AX + BX
3) sub bx,cl ;Ошибка: разный размер операндов.

Сложение и вычитание с переносом (с учётом флага переноса CF)

В системе команд имеются специальные команды сложения и вычитания с учётом флага переноса (CF).

Для сложения с учётом переноса предназначена команда **ADC**, а для вычитания – **SBB**. В общем, эти команды работают почти так же, как ADD и SUB, единственное отличие в том, что к младшему разряду первого операнда прибавляется или вычитается дополнительно значение флага CF.

Они позволяют выполнять сложение и вычитание многобайтных целых чисел, длина которых больше, чем разрядность регистров процессора (в нашем случае 16 бит).

Принцип программирования таких операций очень прост – длинные числа складываются (вычитаются) по частям. Младшие разряды складываются (вычитаются) с помощью обычных команд ADD и SUB, а затем последовательно складываются(вычитаются) более старшие части с помощью команд ADC и SBB. Так как эти команды учитывают перенос из старшего разряда, то мы можем быть уверены, что ни один бит не потеряется. Этот способ похож на сложение(вычитание) десятичных чисел в столбик.

На рисунке 2 показано сложение двух двоичных чисел командой ADD:

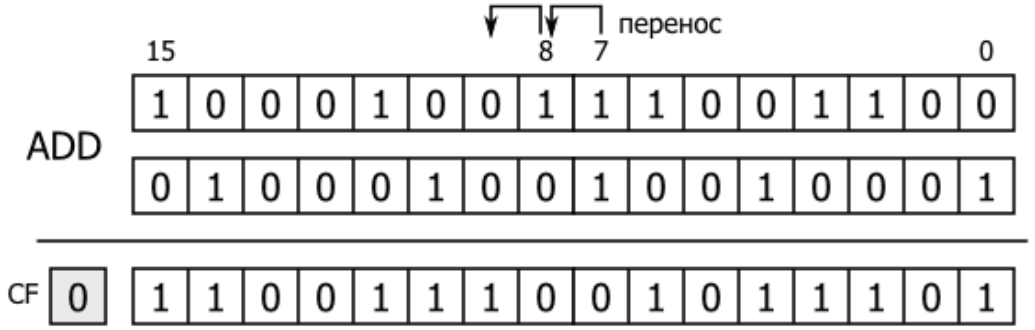


Рисунок 2 – Сложение двоичных чисел

При сложении происходит перенос из 7-го разряда в 8-й, как раз на границе между байтами. Если мы будем складывать эти числа по частям командой ADD, то перенесённый бит потеряется и в результате мы получим ошибку. Но перенос из старшего разряда всегда сохраняется в флаге CF. Чтобы прибавить этот перенесённый бит, достаточно применить команду ADC (рисунок 3):

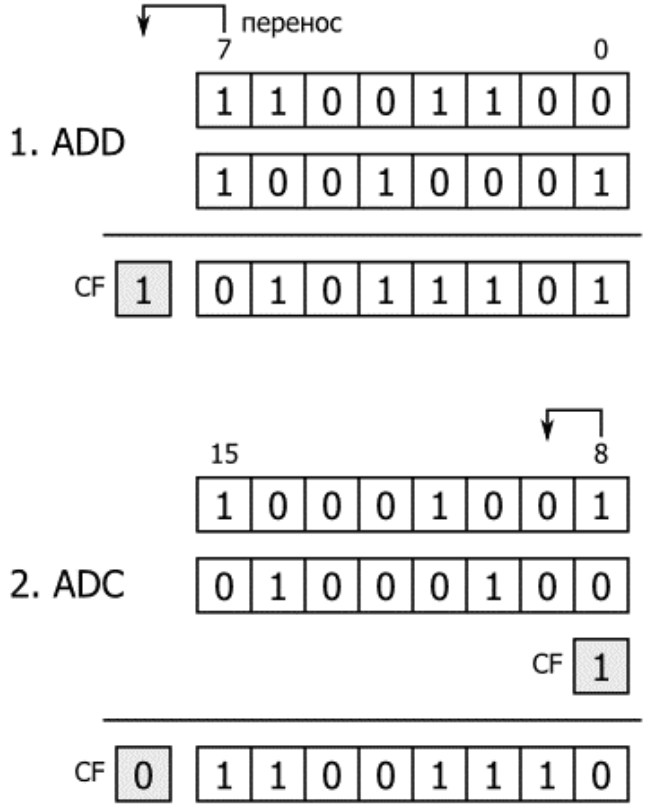


Рисунок 3 – Механизм работы команды ADC

INC - инкремент (прибавление единицы) содержит один операнд

DEC – декремент (вычитания единицы) содержит один операнд

INC *Операнд*

DEC *Операнд*

Логика работы команд:

INC: <*Операнд*> = < *Операнд* > + 1

DEC: <*Операнд*> = < *Операнд* > - 1

- В качестве операнда допустимы регистры и память: reg, mem.
- Операнд не может быть imm (непосредственным операндом)
- Эти команды не изменяют значение ФЛАГА *CF*.

Примеры:

1) inc ax ;AX = AX + 1

2) dec ax ;AX = AX – 1

3) Mov ax,slag1 **; Увеличение числа в** slag1 **на 1**

inc ax

mov rez,ax

NEG – команда для изменения знака операнда.

NEG *Операнд*

Логика работы команды:

<*Операнд*> = 0 – < *Операнд* >

В качестве декремента допустимы регистры и память: reg, mem.

Пример:

neg ax ;AX = -AX

MUL – команда умножения чисел без знака.

У этой команды только один операнд – второй множитель, который должен находиться в регистре или в памяти. Местоположение первого множителя и результата задаётся неявно и зависит от размера операнда:

Размер операнда	Множитель	Результат
1 байт	AL	AX
2 байта	AX	DX:AX
4 байта	EAX	EDX:EAX

Отличие умножения от сложения и вычитания в том, что разрядность результата получается в 2 раза больше, чем разрядность сомножителей.

Примеры:

```
mul bl  ;AX = AL * BL
mul ax  ;DX:AX = AX * AX
```

IMUL – команда умножения чисел со знаком. Эта команда имеет три формы, различающиеся количеством операндов:

1) С одним операндом – форма, аналогичная команде MUL. В качестве операнда указывается множитель. Местоположение другого множителя и результата определяется по таблице.

2) С двумя операндами – указываются два множителя. Результат записывается на место первого множителя. Старшая часть результата в этом случае игнорируется. Эта форма команды не работает с операндами размером 1 байт.

3) С тремя операндами – указывается положение результата, первого и второго множителя. Второй множитель должен быть непосредственным значением. Результат имеет такой же размер, как первый множитель, старшая часть результата игнорируется. Это форма тоже не работает с однобайтными множителями.

Примеры:

```
1)  imul cl      ;AX = AL * CL
2)  imul bx,ax    ;BX = BX * AX
3)  imul cx,-5    ;CX = CX * (-5)
4)  imul dx,bx,134h  ;DX = BX * 134h
```

ФЛАГИ CF = OF = 0, если произведение помещается в младшей половине результата, содержимое другого регистра (старшей части) является расширением знака – все его биты равны старшему биту (знаковому разряду) младшей части результата.

Если cf = of = 1, знаком результата является знаковый бит старшей части результата, а знаковый бит младшей части является значащим битом двоичного кода результата.

Для второй и третьей формы команды CF = OF = 1 означает, что произошло переполнение.

DIV – команда деления чисел без знака. У этой команды один операнд – делитель, который должен находиться в регистре или в памяти. Местоположение делимого, частного и остатка задаётся неявно и зависит от размера операнда:

Размер операнда (делителя)	Делимое	Частное	Остаток
1 байт	AX	AL	AH
2 байта	DX:AX	AX	DX
4 байта	EDX:EAX	EAX	EDX

При выполнении команды DIV может возникнуть прерывание:

- если делитель равен нулю;
- если частное не помещается в отведённую под него разрядную сетку (например, если при делении слова на байт частное больше 255).

Примеры:

1) div cl ;AL = AX / CL, остаток в AH
2) div di ;AX = DX:AX / DI, остаток в DX

IDIV – команда деления чисел со знаком. Единственным операндом является делитель. Местоположение делимого и частного определяется также,

как для команды DIV. Эта команда тоже генерирует прерывание при делении на ноль или слишком большом частном.

Пример. (4 + 8) / (2 * 3)

```
mov bx, 4 //BL = 4
add bx, 8 //BL = BL + 8 = 12
mov al, 2 //AL = 2
mov cl, 3 //CL = 3
mul cl  //AX = AL * CL = 6
xchg bx, ax // обменять содержимое двух операндов, т.е. AX = 12, BX = 6
mov dx, 0
div bx
```

Практическое задание:

Вычислить значение выражения

1. $X = (A-1)*3-(B+2)/2$
2. $A=(B^2+5^2-8)/2$
3. $X=4*C-(A+B)/2$
4. $X=100-(A+2B)/3$
5. $C=10+(X/2)+D*4$
6. $X=A^2+5-B/2$
7. $Y=X^2+5C-1$
8. $Y=(A+B-1)/(2*C)$
9. $A=(20-C)+X*4$
10. $A=100-(10+B*2-B/2)$
11. $X=5*C+10-Y/2$
12. $Y=(D+50)/(C*2-4)$
13. $C=(A-B)*2/(A+B)$
14. $D=(A^2+10)-B/2$
15. $Y=200-((2*A+4)/2)$
16. $Y=1+(X*5-5)/2$

Контрольные вопросы:

1. Опишите назначение арифметических команд.
2. Приведите классификацию арифметических команд ассемблера.
3. Опишите назначение и структуру регистра флагов. Приведите примеры.

Тема 10.2. Форматы команд, способы адресации, система команд однокристальных микропроцессоров

ЛАБОРАТОРНАЯ РАБОТА №11

Разработка и отладка программы с использованием команд пересылки и сравнения кодов. Исследование командного цикла МП при выполнении команд пересылки и сравнения кодов

Цель: Изучить процесс разработки и отладки программ с использованием команд пересылки и передачи управления

Порядок выполнения

1. Изучить теоретическую часть
2. Разработать алгоритм задачи (согласно варианту)
3. Выполнить практические примеры
4. Реализовать полученный алгоритм на языке Ассемблера
5. Выполнить тестирование программы
6. Ответить на контрольные вопросы
7. Оформить отчет по проделанной работе

Теоретическая часть

КОМАНДЫ ПЕРЕСЫЛКИ ДАННЫХ можно разбить на следующие группы:

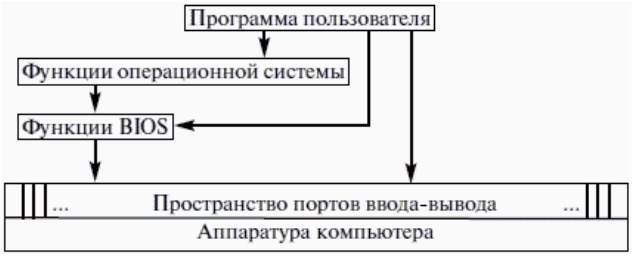
1) пересылки данных общего назначения:

mov – это основная команда пересылки данных

xchg –двунаправленная пересылка данных: операнды должны иметь один тип; не допускается обменивать между собой содержимое двух ячеек памяти.

2) ввода-вывода в порт:

Упрощенная, концептуальная схема управления оборудованием компьютера:



Самым нижним уровнем является уровень BIOS, на котором работа с оборудованием ведется напрямую через порты (сведения о номерах портов, их разрядности, формате управляющей информации приводятся в техническом описании устройства).

in аккумулятор, номер_порта – ввод в аккумулятор из порта с номером номер_порта;

out порт, аккумулятор – вывод содержимого аккумулятора в порт с номером номер_порта.

3) работы с адресами и указателями:

lea назначение, источник – загрузка эффективного адреса;

Команда lea похожа на команду mov тем, что она также производит пересылку. Однако команда lea производит пересылку не данных, а эффективного адреса данных (т. е. смещения данных относительно начала сегмента данных) в регистр, указанный операндом назначение.

Ids назначение, источник – загрузка указателя в регистр сегмента данных ds;

les назначение, источник – загрузка указателя в регистр дополнительного сегмента данных es;

lgs назначение, источник – загрузка указателя в регистр дополнительного сегмента данных gs;

lfs назначение, источник – загрузка указателя в регистр дополнительного сегмента данных fs;

lss назначение, источник – загрузка указателя в регистр сегмента стека ss.

4) преобразования данных:

xlat [адрес_таблицы_перекодировки] -

команда замещает значение в регистре al другим байтом из таблицы в памяти, расположенной по адресу, указанному операндом адрес_таблицы_перекодировки.

5) работы со стеком:

Для работы со стеком предназначены три регистра:

ss – сегментный регистр стека;

sp/esp – регистр указателя стека;

bp/ebp – регистр указателя базы кадра стека.

В каждый момент времени доступен только один стек, адрес сегмента которого содержится в регистре SS. Этот стек называется текущим. Для того чтобы обратиться к другому стеку («переключить стек»), необходимо загрузить в регистр ss другой адрес. Регистр SS автоматически используется процессором для выполнения всех команд, работающих со стеком.

push источник – запись значения источник в вершину стека

pop назначение – запись значения из вершины стека по месту, указанному операндом назначение (значение при этом «снимается» с вершины стека).

pusha – команда групповой записи в стек (в стек последовательно записываются регистры ax, cx, dx, bx, sp, bp, si, di).

pushf – сохраняет регистр флагов в стеке.

Команды передачи управления

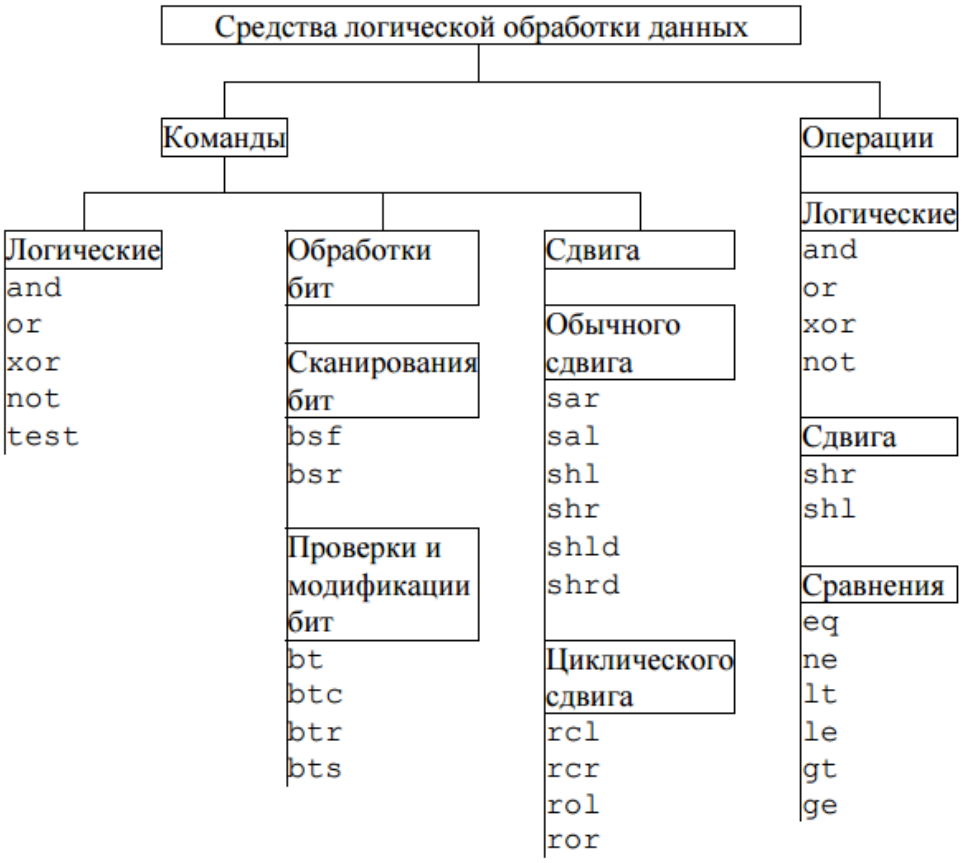


Рисунок 1 – Команды передачи управления

ЛОГИЧЕСКИЕ КОМАНДЫ

Команды выполняют логические операции над **битами** операндов, начиная с младшего (нулевого). Размерность операндов должна быть одинакова.

and – операция логического умножения.

Логика работы команды:

< операнд_1> = < операнд_1> И < операнд_2>

```
Mov al,00110011b
And al,11101110b   ; al=0010010b
```

or – операция логического сложения.

Команда выполняет поразрядно логическую операцию ИЛИ (дизъюнкцию) над битами операндов операнд_1 и операнд_2.

Логика работы команды:

< операнд_1> = < операнд_1> ИЛИ < операнд_2>

```
mov al,00110011b
or al,11101110b     ; al=11111111b
```

xor – операция логического исключающего сложения.

Команда выполняет поразрядно логическую операцию исключающего ИЛИ над битами операндов операнд_1 и операнд_2.

Логика работы команды:

< операнд_1> = < операнд_1> ⊕< операнд_2>

Если один из сравниваемых битов равен 0, а другой равен 1, то результат равен 1; если сравниваемые биты одинаковы (оба - 0 или оба - 1) то результат - 0

```
mov al,00110011b
xor al,11101110b            ; al=11011101b
```

test– операция «проверить» (способом логического умножения).

Команда выполняет поразрядно логическую операцию И над битами операндов операнд_1 и операнд_2. Состояние операндов остается прежним, изменяются только флаги ZF, SF, PF, что дает возможность анализировать состояние отдельных битов операнда без изменения их состояния;

test операнд_1, операнд_2

test действует как AND-устанавливает флаги, но не изменяет биты.

```
mov al,00110011b
test al,11101110b   ; al=0011011b SF=0
```

not– операция логического отрицания.

Команда выполняет поразрядное инвертирование (замену значения на обратное) каждого бита операнда.

not операнд

С помощью логических команд возможно выделение отдельных битов в операнде с целью их установки, сброса, инвертирования или просто проверки на определенное значение.

•Для установки определенных разрядов (бит) в 1 применяется команда
or операнд_1,операнд_2.
В этой команде операнд_2, выполняющий роль маски, должен содержать единичные биты на месте тех разрядов, которые должны быть установлены в 1 в операнд_1.
or eax,10b; установить 1-й бит в регистре eax

•Для сброса определенных разрядов (бит) в 0 применяется команда
and операнд_1,операнд_2.
В этой команде операнд_2, выполняющий роль маски, должен содержать нулевые биты на месте тех разрядов, которые должны быть установлены в 0 в операнд_1.
and eax,fffffffdh; сбросить в 0 1-й бит в регистре eax

•Команда xor операнд_1,операнд_2 применяется:
1)для выяснения того, какие биты в операнд_1 и операнд_2 различаются;
2)для инвертирования состояния заданных бит в операнд_1.
xor eax,10b; инвертировать 1-й бит в регистре eax

bsf, bsr - поиск первого установленного в 1 бита операнда.

bsf операнд_1,операнд_2 - сканирование битов вперед.

Команда просматривает (сканирует) биты операнд_2 от младшего к старшему (от бита 0 до старшего бита) в поисках первого бита, установленного в 1. Если таковой обнаруживается, в операнд_1 заносится номер этого бита в виде целочисленного значения. Если все биты операнд_2 равны 0, то флаг нуля ZF устанавливается в 1, в противном случае флаг ZF сбрасывается в 0.

```
mov al,02h
bsf bx,al       ;bx=1
```

bsr операнд_1,операнд_2 — сканирование битов в обратном порядке. Команда просматривает (сканирует) биты операнд_2 от старшего к младшему (от старшего бита к биту 0) в поисках первого бита, установленного в 1. Если таковой обнаруживается, в операнд_1 заносится номер этого бита в виде целочисленного значения.

При этом важно, что позиция первого единичного бита слева отсчитывается все равно относительно бита 0. Если все биты операнд_2 равны 0, то флаг нуля ZF устанавливается в 1, в противном случае флаг ZF сбрасывается в 0.

bt, bts, btr, btc - доступ к одному конкретному биту операнда (операнд может находиться как в памяти, так и в регистре общего назначения).

bt – проверка бита.

bt операнд, смещение_бита

Команда переносит значение бита во флаг CF.

```
bt ax,5         ;проверить значение бита 5
```

bts – проверка и установка бита.

bts операнд, смещение_бита

Команда переносит значение бита во флаг CF и затем устанавливает проверяемый бит в 1.

```
mov ax,10
bts pole,ax; проверить и установить 10-й бит в pole
```

btr – проверка и сброс бита.

btr операнд, смещение_бита

Команда переносит значение бита во флаг CF и затем устанавливает этот бит в 0.

btc – проверка и инвертирование бита.

btc операнд, смещение_бита

Команда переносит значение бита во флаг CF и затем инвертирует значение этого бита.

КОМАНДЫ СДВИГА

Все команды сдвига перемещают биты в поле операнда влево или вправо в зависимости от кода операции. При выполнении команд сдвига флаг CF всегда содержит значение последнего выдвинутого бита.

Структура команд сдвига: КОП операнд, счетчик сдвигов

По принципу действия команды сдвига можно разделить на два типа:

•команды линейного сдвига;

•команды циклического сдвига.

SHR - Логический (беззнаковый) сдвиг вправо

SHL - Логический (беззнаковый) сдвиг влево (правые биты заполняются нулями)

SAR - Арифметический сдвиг вправо (для заполнения левого бита используется знаковый бит, положительные и отрицательные величины сохраняют свой знак).

SAL - Арифметический сдвиг влево (правые биты заполняются нулями)

1) Mov cl,03
   Mov ax,10110111b
   Shr ax,1 ; 01011011 - Сдвиг вправо на 1
   Shr ax,cl ; 00001011 - Сдвиг вправо на 3
2) Mov cl,03
   Mov ax,10110111b
   Sar ax,1 ; 11011011 ;Сдвиг вправо на 1

Sar ax,cl ; 11111011 ;Сдвиг вправо на 3

Практические задания:

Задание 1. Используя команды сдвига, сложения и вычитания, составить и отладить программу для вычисления заданного арифметического выражения. Команды умножения и деления не использовать.

№	Выражение	№	Выражение	№	Выражение
1	$y=\frac{8(a-b)+5}{4}$	11	$y=\frac{8(a+b)-2b}{4}$	21	$y=\frac{2a+4b-3}{8}$
2	$y=\frac{2(a-3)+7b}{8}$	12	$y=\frac{2a+16b-5}{2}$	22	$y=\frac{4(a+b)-2b}{8}$
3	$y=\frac{7a+4b-8}{8}$	13	$y=\frac{2(4a-b)+6}{4}$	23	$y=\frac{4(8a-b)+2a}{4}$
4	$y=\frac{16(a-4b)+7}{4}$	14	$y=\frac{2(8a+b)-3}{4}$	24	$y=\frac{8(4a-b)+9}{4}$
5	$y=\frac{8a-2b+4}{2}$	15	$y=\frac{2(a+8b)-3}{4}$	25	$y=\frac{4(a-16b)+4}{8}$
6	$y=\frac{16a+8b-3}{4}$	16	$y=\frac{4(a+8b)+6}{4}$	26	$y=\frac{2a-8b}{4}$
7	$y=\frac{8(a-2b)}{4}$	17	$y=\frac{8a+4b-5}{2}$	27	$y=\frac{8b+2a-7}{16}$
8	$y=\frac{2(a-4b)+6}{8}$	18	$y=\frac{4(a-8b)+5}{4}$	28	$y=\frac{2(a-8b)+7}{4}$
9	$y=\frac{4(a-2b)-5b}{2}$	19	$y=\frac{8(a-2b)+b}{4}$	29	$y=\frac{4(a+2b)-9}{8}$
10	$y=\frac{2(8b+a)-1}{4}$	20	$y=\frac{4(16a+b)-6}{8}$	30	$y=\frac{2(a+8b)-b}{4}$

Пример выполнения задания: $y=\frac{2a+8b-7}{4}$.

```
dseg segment para public 'data'
    a db 3
    b db 2
    y db ?
    mesa db 10,13,'a=$'
    mesb db 10,13,'b=$'
    mesy db 10,13,'y=$'
dseg ends
sseg segment para stack 'stack'
    db 256 dup (0)
sseg ends
extrn disp:near
```

```
cseg segment para public 'code'
main proc near
assume cs:cseg, ds:dseg, ss:sseg
    mov ax,dseg
    mov ds,ax
    mov al,a        ;al=a
    sal al,1   ;al=2a
    mov bl,b        ;bl=b
    sal bl,3   ;bl=8b
    add al,bl       ;al=2a+8b
    sub al,7  ;al=2a+8b-7
    sar al,2   ;al=(2a+8b-7)/4
    mov y,al        ;y=al
    lea dx,mesa
    mov ax,0900h
    int 21h
    mov al,a
    cbw
    call disp
    lea dx,mesb
    mov ax,0900h
    int 21h
    mov al,b
    cbw
    call disp
    lea dx,mesy
    mov ax,0900h
    int 21h
    mov al,y
    cbw
```

```
    call disp
    mov ax,4C00h
    int 21h
main endp
cseg ends
end main
```

Задание 2. В соответствии с вариантом напишите программу на языке ассемблера с полным описанием сегментов для вычисления значения y.

1	$y = y1 + y2;\ y1 = \begin{cases} a + x, & \text{если } x > a \\ 2a - x, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a * x, & \text{если } x > 10 \\ x, & \text{если } x <= 10 \end{cases}$
2	$y = y1 - y2;\ y1 = \begin{cases} x - 2, & \text{если } x >= 2 \\ 8, & \text{если } x < 2 \end{cases};\ y2 = \begin{cases} 4, & \text{если } x = 0 \\ a - x, & \text{если } x <> 0 \end{cases}$
3	$y = y1 * y2;\ y1 = \begin{cases} x - a, & \text{если } x > a \\ 5, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a, & \text{если } a > x \\ a * x, & \text{если } a <= x \end{cases}$

4	$y = y1 + y2;\ y1 = \begin{cases} 2 - x, & \text{если } x < 2 \\ a + 3, & \text{если } x >= 2 \end{cases};\ y2 = \begin{cases} a - 1, & \text{если } x < a \\ a * x - 1, & \text{если } x >= a \end{cases}$
5	$y = y1 - y2;\ y1 = \begin{cases} \lvert x \rvert, & \text{если } x < 0 \\ x - a, & \text{если } x >= 0 \end{cases};\ y2 = \begin{cases} a + x, & \text{если } x \bmod 3 = 1 \\ 7, & \text{в остальных случаях} \end{cases}$
6	$y = y1 + y2;\ y1 = \begin{cases} x \bmod 4, & \text{если } x > a \\ a, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a * x, & \text{если } x / a > 3 \\ x, & \text{если } x / a <= 3 \end{cases}$
7	$y = y1 + y2;\ y1 = \begin{cases} 4 - x, & \text{если } \lvert x \rvert < 3 \\ a + x, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} 2, & \text{если } x \text{ четное} \\ a + 2, & \text{в остальных случаях} \end{cases}$
8	$y = y1 + y2;\ y1 = \begin{cases} 4 * x, & \text{если } x <= 4 \\ x - a, & \text{если } x > 4 \end{cases};\ y2 = \begin{cases} 7, & \text{если } x \text{ нечетное} \\ x / 2 + a, & \text{в остальных случаях} \end{cases}$
9	$y = y1 * y2;\ y1 = \begin{cases} a * x, & \text{если } x \bmod 3 = 2 \\ 9, & \text{в остальных случаях} \end{cases};\ y2 = \begin{cases} a - x, & \text{если } a > x \\ a + 2, & \text{если } a <= x \end{cases}$
10	$y = y1 - y2;\ y1 = \begin{cases} a + \lvert x \rvert, & \text{если } x > a \\ a - 7, & \text{если } x <= a \end{cases};\ y2 = \begin{cases} a * 3, & \text{если } a > 3 \\ 11, & \text{если a <= 3} \end{cases}$
11	$y = y1 \bmod y2;\ y1 = \begin{cases} 10 + x, & \text{если } x > 1 \\ \lvert x \rvert + a, & \text{если } x <= 1 \end{cases};\ y2 = \begin{cases} 2, & \text{если } x > 4 \\ x, & \text{если } x <= 4 \end{cases}$
12	$y = y1 / y2;\ y1 = \begin{cases} 15 + x, & \text{если } x > 7 \\ \lvert a \rvert + 9, & \text{если } x <= -7 \end{cases};\ y2 = \begin{cases} 3, & \text{если x > 2} \\ \lvert x \rvert - 5, & \text{если } x <= 2 \end{cases}$
13	$y = y1 * y2;\ y1 = \begin{cases} 3 + x, & \text{если } x = a \\ a - x, & \text{если } x <> a \end{cases};\ y2 = \begin{cases} \lvert a \rvert, & \text{если } a < x \\ \lvert a \rvert - x, & \text{если ```a >= } x \end{cases}$
14	$y = y1 - y2;\ y1 = \begin{cases} 2 * x + a, & \text{если } x > 2 \\ 2 * x + 1, & \text{если } x <= 2 \end{cases};\ y2 = \begin{cases} \lvert x \rvert + 1, & \text{если x > 0} \\ a - 1, & \text{если } x <= 0 \end{cases}$
15	$y = y1 \bmod y2;\ y1 = \begin{cases} 8 + \lvert x \rvert, & \text{если } x < 1 \\ \lvert a \rvert * 2, & \text{если } x >= 1 \end{cases};\ y2 = \begin{cases} 3, & \text{если } x = a \\ a + 1, & \text{если } x <> a \end{cases}$

Пример: Необходимо ввести с клавиатуры значение двух переменных a и x. Если a<x, то сложить их значения, а иначе из x отнять a.

```
data segment
    a db ?
    x db ?
    per db 10,13,'$'
    mesa db 10,13,'Input a: $'
    mesx db 10,13,'Input x: $',10,13
data ends
s segment
```

```
    stack db 128 dup(?)
s ends
code segment
main:
    assume ss:s,ds:data,cs:code
    mov ax,data
    mov ds,ax
    mov dx,offset mesa
    mov ah,9            ;Приглашение на ввод a
    int 21h
    mov ah,1            ;Считывание нажатого символа
    int 21h
    mov a,al            ;Запись считанного символа в a
    mov dx,offset mesx
    mov ah,9            ;Приглашение на ввод x
    int 21h
    mov ah,1            ;Считывание нажатого символа
    int 21h
    mov x,al            ;Запись считанного символа в x
    mov dx,offset per
    mov ah,9            ;Перевод строки int 21h
    mov al,x
    cmp a,al
    jl Lower            ;Если a<x,то перейти на метку Lower. Иначе на метку
    Higher
    Higher:
        mov al,a
        sub al,x
        add al,30h      ;Коррекция по вычитанию
        jmp short l1
```

```
    lower:
        mov al,x        ;В регистр al записываем результат сложения a и x
        add al,a
        sub al,30h      ;Корекция по сложению
    l1:
        mov dl,al
        mov ah,2        ;Вывод содержимого dl на экран
        int 21h
        mov ah,0        ;Ожидание нажатия клавиши
        int 16h
        mov ah,4ch
        int 21h
        code ends
end main
```

Контрольные вопросы:

1. На какие категории делятся команды пересылки данных? Опишите каждую из них.
2. Приведите классификацию команд передачи управления.
3. На какие группы могут быть разделены команды передачи управления по принципу действия команды микропроцессора?

РАЗДЕЛ X. Универсальные микропроцессоры

Тема 10.2. Форматы команд, способы адресации, система команд однокристальных микропроцессоров

ЛАБОРАТОРНАЯ РАБОТА №12

Разработка и отладка программы с использованием команд переходов. Исследование командного цикла МП при выполнении команд переходов

Цель: Изучить при помощи отладчика работу процессора по выполнению команд ветвлений. Выяснить, каким образом вычисляется смещение при переходе от выполняемой команды к команде, обозначенной меткой. Научиться работать с логическими адресами команд.

Порядок выполнения

1. Изучить теоретическую часть
2. Разработать алгоритм задачи (согласно варианту)
3. Выполнить практические задания
4. Реализовать полученный алгоритм на языке Ассемблера
5. Выполнить тестирование программы
6. Ответить на контрольные вопросы
7. Оформить отчет по проделанной работе

Теоретическая часть

БЕЗУСЛОВНЫЕ ПЕРЕХОДЫ

Все команды безусловного перехода обозначаются одинаково: **JMP op**, но в зависимости от типа операнда **op** ассемблер формирует разные машинные команды. Команда передает управление в указанную точку того же или другого программного сегмента, может совершать переход из текущей процедуры, от одной процедуры к другой, от одного сегмента к другому, полностью выходя за пределы текущей программы или в любое место памяти RAM или ROM. Не воздействует на флаги CPU. Адрес точки вызова теряется и **вернуться к ней нельзя**!

Команда JMP заставляет процессор продолжать выполнение с нового места программы**.** Новое место может быть отмечено меткой, которую

процессор преобразовывает в адрес. Если переход происходит в текущий сегмент, то смещение метки загружается непосредственно в регистр IP. Если метка находится в другом сегменте, адрес сегмента дополнительно загружается в регистр CS.

Если изменяется **только IP**, то такой переход называется **внутрисегментным** или **ближним** (управление остается в том же сегменте команд), а если меняются оба регистра **CS и IP**, то это **межсегментный** или **дальний** переход (начинают выполняться команды из другого сегмента команд).

1. **Прямой короткий (short) переход (внутрисегментный)**:

JMP <offset8> ; IP = IP + <offset8>

Прямым называется переход, в команде которого в явной форме указывается метка, на которую нужно перейти.

В команде **JMP** указывается метка той команды, на которую надо передать управление, и **ассемблер сам вычисляет сдвиг.** Команда **прибавляет это число к текущему значению регистра IP**, получая в нем адрес (смещение от начала сегмента команд) той команды, которая должна быть выполнена следующей. Значение регистра CS при этом не меняется. Помните, что величина <offset> прибавляется к содержимому регистра IP, хранящего уже адрес команды следующей за JMP. Поэтому, например, команда JMP 0 – это переход на следующую команду программы.

По короткому переходу можно передать управление только на ближайшие команды программы – отстоящие от команды, следующей за командой перехода, до 128 байтов назад (-128) или до 127 (+127) байтов вперед.

JMP SHORT L1 ; переход к команде с меткой L1 и явно указывается, что используется прямой короткий переход в пределах – 128 … + 127 байт

JMP L1 **;** прямой короткий переход к команде с меткой L1 (определяется смещение)

2. **Прямой ближний (near) или внутрисегментный переход:**

JMP <offset16> ; IP = IP + <offset16>

<offset16> обозначает смещение размером в слово, который рассматривается как знаковое целое от –32768 до +32767.

Встретив команду перехода с меткой, которой была помечена одна из предыдущих команд программы, **ассемблер вычисляет разность между адресом этой метки и адресом команды перехода** и по этому сдвигу определяет, какой переход – короткий или ближний – надо сформировать.

Но если метка еще не встречалась в тексте программы, т.е. делается переход вперед, тогда ассемблер, не зная еще адреса метки, не может определить, какую именно машинную команду прямого перехода формировать, поэтому он на всякий случай выбирает команду ближнего перехода. Однако эта машинная команда занимает 3 байта, а команда короткого перехода – 2 байта. Поэтому, если известно, что должна выполняться команда короткого перехода, с целью экономии памяти необходимо заранее сообщить об этом ассемблеру, чтобы он сформировал команду короткого перехода. Такое указание дается с помощью оператора SHORT. Для переходов назад оператор SHORT не нужен, т.к. уже зная адрес метки, ассемблер сам определит вид команды прямого перехода.

3. **Косвенный ближний (внутрисегментный) переход:**

JMP Reg16 ; IP = [reg]

или

JMP Mem16 ; IP = [mem16]

Здесь **Reg16** обозначает любой из 16-разрядных регистров AX, BX, CX, DX, SI, DI, SP, BP, а Mem16 – адрес слова памяти, задаваемый одним из режимов адресации. В этом регистре (или слове памяти) должен находиться адрес, по которому и будет произведен переход. Например, по команде **JMP BX** осуществляется переход по адресу, находящемуся в регистре BX.

MOV AX, 1400h ; переход ближний косвенный по содержимому регистра AX

JMP AX **;** в IP заносится число 1400h

M1 dw 1400h **;** переменная M1 описана как слово

JMP M1 **;** эта команда работает так же, как и предыдущая.

JMP word ptr A **;** косвенный ближний переход, в переменной A находится 16-разрядный адрес, который становится содержимым регистра IP.

4. Прямой дальний (far), или межсегментный переход:

JMP seg:ofs; CS=seg, IP = ofs

Здесь **seg** – начало (старшие 16 разрядов начального адреса) сегмента памяти, а **ofs** – смещение в этом сегменте.

В командах эта пара **всегда задается конструкцией FAR PTR <метка>**, которая указывает, что надо сделать переход по указанной метке, причем эта метка – "дальняя", т.е. находится в другом сегменте. Ассемблер сам определяет какой нужен сегмент. И сам подставляет в машинную команду его начало, т.е. значение **seg**.

JMP far ptr A **;** прямой дальний переход по метке. Метка A расположена в другом сегменте кодов.

5. Косвенный дальний (межсегментный) переход:

JMP mem32 ; CS=[mem32+2], IP = [mem32]

Здесь **mem32** – адрес двойного слова памяти, в котором находится пара **seg:ofs,** задающая абсолютный адрес, по которому данная команда должна выполнить переход. Помните, что в ПК величины размером в двойное слово хранятся в "перевернутом" виде, поэтому смещение **ofs** находится в младшем слове двойного слова mem32, а смещение **seg –** в старшем слове (по адресу mem32+2).

JMP dword ptr A ; косвенный дальний переход, в 32-разрядной переменной A находится базовый адрес нового сегмента кодов seg (старшее слово) и новое содержимое регистра IP – ofs (младшее слово).

Команды межсегментного перехода используются тогда, когда команды программы размещены не в одном сегменте памяти, а в нескольких (например,

команд столько, что они не помещаются в 64 Кбайтах). При переходе из одного такого сегмента в другой необходимо менять не только счетчик команд **IP**, но и содержимое регистра **CS**, загружая в последний начальный адрес второго сегмента. Такое одновременное изменение обоих этих регистров и делают команды межсегментного перехода.

УСЛОВНЫЕ ПЕРЕХОДЫ

При выполнении перехода по условию выполняется **два шага**:

1. сначала выполняются **арифметические команды и команды сравнения**, по результату которых процессор устанавливает определенные флаги;
2. затем **команды условного перехода**, при которых процессор принимает решение на основе состояния флагов.

Команда перехода по условию передает управление по указанному адресу, когда признак условия установлен, в противном случае перехода не происходит, и процессор продолжает работу со следующей команды.

Синтаксис: Jcc метка,

где cc определяет признак условия, идентифицируя состояние одного или нескольких флагов.

Условия по каждой мнемонической записи даны ниже в таблице. Термины меньше и больше используются для сравнения знаковых целых операндов, а выше и ниже – без знаковых целых.

Переходы на основе чисел без знака: например, сравниваются такие числа, как 7FFFh и 8000h, и предполагается, что первое число меньше последующих.

Переходы, основанные на сравнении чисел со знаком: например, сравниваются такие числа, как 80h (-128) и 7Fh (+127), и предполагается, что первое число меньше последующих.

В командах перехода непосредственно рассчитывать или определять смещение или сам адрес перехода программисту нет необходимости, программа ассемблера сделает это автоматически и, при необходимости, выдаст предупреждающее сообщение, если это сделать невозможно по тем или иным причинам. Переход может осуществляться как вперед, так и назад в диапазоне -128 …+127 байтов.

Команда **Jcc** не оказывает влияния на флаги.

Команда	Перейти, если	Условие перехода
ja	выше	CF=0 и ZF=0
jae	выше или равно	CF=0
jb	ниже	CF=1
jbe	ниже или равно	CF=1 и ZF=1
jc	перенос	CF=1
jcxz	CX=0	CX=0
je	равно	ZF=1
jg	больше	ZF=0 или SF=OF
jge	больше или равно	SF=OF
jl	меньше	SF не равно OF
jle	меньше или равно	ZF=1 или SF не равно OF
jna	не выше	CF=1 или ZF=1
jnae	не выше и не равно	CF=1
jnb	не ниже	CF=0
jnbe	не ниже и не равно	CF=0 и ZF=0
jnc	нет переноса	CF=0
jne	не равно	ZF=0
jng	не больше	ZF=1 или SF не равно OF
jnge	не больше и не равно	SF не равно OF

jnl	не меньше	SF=OF
jnle	не меньше и не равно	ZF=0 и SF=OF
jn	нет переполнения	OF=0
jnp	нет четности	PF=0
jns	знаковый бит равен 0	SF=0
jnz	не нуль	ZF=0
jo	переполнение	OF=1
jp	есть четность	PF=1
jpe	сумма битов четная	PF=1
jpo	сумма битов нечетная	PF=0
js	знаковый бит равен 1	SF=1
jz	нуль	ZF=1

Примеры команд:

```
CMP CX, 0        ; CX=0 ?
JE   L1          ; если да (т.е. если ZF=1), перейти на метку L1

CMP AX, 1000          ; пусть AX=8000h= 32768 (= - 32768h)
JA   L2          ; 32768 > 1000, переход будет, т.к. CF=ZF=0.
```

Перед командами условных переходов обычно используется команда сравнения операндов – **CMP**.

CMP операнд1, операнд2

Действие: операнды операнд1 и операнд2 сравниваются методом вычитания (операнд1 – операнд2), при этом сами операнды не изменяются. Флаги: OF, SF, ZF, AF, PF, CF.

Необходимо помнить, что при сравнении операндов:

А) если операнды равны, то ZF=1, иначе ZF=0;

Б) для без знаковых операндов:

CMP		CF	ZF
операнд1 операнд2	<	1	0
операнд1 операнд2	=	0	1
операнд1 операнд2	>	0	0

В) для знаковых операндов:

CMP		ZF	SF, OF
операнд1 операнд2	<	?	SF<>OF
операнд1 операнд2	=	1	?
операнд1 операнд2	>	0	SF=OF

Практические задания:

Задание 1:

1. Даны три числа. Записать в регистр SI меньшее, а в регистр DI - большее из них.
2. Ввести символ с клавиатуры (ah=01h, int 21h, введенный символ в al). Если это цифра, занести ее в регистр BL.
3. Дано 4-ех значное число. Оно хранится в регистре BX. Найти в этом числе самую большую цифру и занести ее в регистр DX.

4. Переменной K присвоить 10, если среди цифр числа (4-х значного) есть одинаковые цифры.

5. Дано целое число. Если оно чётное, то записать в регистр DI значение FFFFh, если нечётное - FF00h.

Задание 2

Предположим, что команда CMP сравнивает два операнда. Для каждой из команд условного перехода из таблицы 4 определено состояние 4-х флагов. Отметьте в последнем столбце будет ли осуществляться переход.

Команда	OF	SF	ZF	CF	Переход
1. JNZ	0	0	1	0	
2. JA	1	0	1	0	
3. JNB	1	0	1	0	
4. JBE	0	0	1	0	
5. JO	1	1	0	0	
6. JNLE	0	1	0	1	
7. JNS	0	0	1	0	
8. JNG	1	1	0	1	
9. JE	1	0	1	0	
10. JNAE	1	0	1	0	

Контрольные вопросы:

1. На какие флаги действует команда JMP?
2. В чем особенности ближнего и дальнего переходов?
3. В чем отличие между коротким и ближним переходом?
4. Какое различие между командами JB и JL?

5. Можно ли при переходе по условию сделать переход на метку в любом месте сегмента?
6. После выполнения команды JB какие флаги устанавливаются?

Тема 10.2. Форматы команд, способы адресации, система команд однокристальных микропроцессоров

ЛАБОРАТОРНАЯ РАБОТА №13

Изучение возможностей системы команд МП для разработки циклической программы

Цель: изучить возможности системы команд МП для разработки циклической программы; научиться разрабатывать и осуществлять отладку циклической программы.

Порядок выполнения

1. Изучить теоретическую часть.
2. Разработать алгоритм задачи (согласно варианту).
3. Выполнить практические задания.
4. Реализовать полученный алгоритм на языке Ассемблера.
5. Выполнить тестирование программы.
6. Ответить на контрольные вопросы.
7. Оформить отчет по проделанной работе.

Теоретическая часть

Как и в любом языке программирования, в языке Assembler существует несколько способов организации циклического повторения фрагмента программы. Каждый из способов имеет свои особенности, поэтому для

эффективной реализации конкретной задачи следует использовать наиболее подходящий способ.

Рассмотрим особенности способов организации циклов.

***Команда LOOP* – цикл с известным числом повторений:**

Команда *LOOP* требует, чтобы в качестве счётчика цикла использовался регистр ECX. Команда вычитает единицу именно из этого регистра, сравнивает полученное значение с нулём и осуществляет переход на указанную метку, если значение в регистре ECX больше 0.

```
mov  ecx, n
L: ...
...
 loop L
```

Команды LOOPE/LOOPZ и LOOPNE/LOOPNZ

Эти команды похожи на команду *LOOP*, но позволяют также организовать и досрочный выход из цикла.

LOOPE `<метка>; Команды являются синонимами` *LOOPZ* `<метка>`

Действие этой команды можно описать следующим образом: `ECX = ECX - 1; if (ECX != 0 && ZF == 1) goto <метка>;`

До начала цикла в регистр ECX необходимо записать число повторений цикла. Команда *LOOPE*/*LOOPZ*, как и команда *LOOP* ставится в конце цикла, а перед ней помещается команда, которая меняет флаг ZF (обычно это команда сравнения *CMP*). Команда *LOOPE*/*LOOPZ* заставляет цикл повторяться ECX раз, но только если предыдущая команда фиксирует равенство сравниваемых величин (вырабатывает нулевой результат, т.е. ZF = 1).

По какой именно причине произошёл выход из цикла надо проверять после цикла. Причём надо проверять флаг ZF, а не регистр ECX, т.к. условие ZF = 0 может появиться как раз на последнем шаге цикла, когда и регистр ECX стал нулевым.

Команда *LOOPNE*/*LOOPNZ* аналогична команде *LOOPE*/*LOOPZ*, но досрочный выход из цикла осуществляется, если ZF = 1.

С помощью команд перехода можно реализовать любые разветвления в программе:

Пример 1:

```
   ; if (x > 0) S
    cmp  x, 0
    jleL
  ...                          ; S
L:
```

Пример 2

```
  ; if (x) S1 else S2
    cmp  x, 0
  je  L1
  ...                          ; S1
    jmp  L2
  L1: ...                              ; S2
L2:
```

Пример 3

```
; if (a > 0 && b > 0) S
    cmp  a, 0
  jle  L
    cmp  b, 0
  jleL
  ...                          ; S
L:
```

Практическая часть

Задание 1.

Для данных ниже команд определить тип перехода (прямой или косвенный), для неверных команд объяснить ошибку. Считать, что выше команды перехода были описаны имена: A — байт; B — двойное слово; K — константа со значением 4000FFFFh; L, M — метки.

а) JMP L

б) JMP A

в) JMP BX

г) JMP B

д) JMP ECX

е) JMP M

ж) JMP EIP

з) JMP K

Задание 2.

X DB 206; (–50)

Определить, будет ли сделан переход на метку MET при выполнении следующих команд:

а) CMP X,100 JA MET

б) CMP X,100 JG MET

в) CMP X,210 JA MET

г) CMP X,210 JE LAB JB MET LAB:

д) CMP X,–40 JG MET

е) CMP X,–40 JL MET

ж) CMP X,216 JL MET

Задание 3.

Реализовать следующие последовательности операторов (числа со знаком)

а) if ESI>20 then AL:= 1 else AL:= 0;

б) AL:=0; if ESI>20 then AL:=1

Сравнить полученные фрагменты кода.

Контрольные вопросы:

1. Что представляет собой цикл на языке ассемблера?

2. Какие команды могут быть использованы для создания циклических конструкций на языке ассемблера?

3. В чем заключается особенность отладки циклической программы?

РАЗДЕЛ X. Универсальные микропроцессоры

Тема 10.1. Организация и программная модель однокристальных микропроцессоров

ЛАБОРАТОРНАЯ РАБОТА №14

Изучение возможностей системы команд мп для разработки программы с использованием подпрограмм и стека

Цель: Исследование командного цикла процессора на уровне микрокоманд при выполнении программ с использованием стека

Порядок выполнения

8. Изучить теоретическую часть.
9. Выполнить практические задания.
10. Занести результаты работы программы в таблицу анализа состояния микропроцессора.
11. Ответить на контрольные вопросы.
12. Оформить отчет по проделанной работе.

Теоретическая часть

Стек – упорядоченный набор ячеек с последовательным доступом, для временного хранения произвольных данных. Данные можно сохранять и в сегменте данных, но в этом случае для каждого временно сохраняемого данного надо заводить отдельную именованную ячейку, что не целесообразно, т.к. увеличивает размер программы.

Стек обычно использует для временного хранения адресов возврата из подпрограмм или прерываний, чтобы можно было продолжить выполнение программы с прерванной точки.

Элементы стека располагаются в области памяти, специально отведенной под стек, начиная со **дна стека**, т.е. с его **максимального адреса.** При этом адрес ячейки памяти хранящей последний занесенный элемент меньше любого из адресов, хранящих ранее занесенные элементы.

Для адресации такой области служит **SS**(сегментный регистр). В нем хранится начальный адрес этого сегмента памяти. И **SP –** в нем хранится смещение по отношению к началу этого сегмента.

Адрес верхнего, доступного, элемента стека хранится в **SP,** следовательно, пара регистров **SS:SP** описывает адрес доступной ячейки стека. Важно помнить, что в исходном состоянии указатель стека, регистр **SP,** указывает на ячейку лежащую под дном стека и не входящую в него.

Стек – это область памяти для хранения временных данных. Стек используется командой **CALL** для хранения адреса, чтобы программа могла вернуться к тому месту, откуда была вызвана процедура. Команда **RET** получает этот адрес из стека и возвращает управление по этому смещению. то же самое происходит, когда команда **INT** вызывает прерывание, она записывает в стек регистр флагов, сегмент и смещение кода. Команда **IRET** используется для возвращения после вызова прерывания.

Вы можете использовать стек для хранения любых данных. Для работы со стеком имеются две команды:

- **PUSH** – записывает 16-ти битовое значение в стек
- **POP** – получает 16-ти битовое значение из стека

Процесс записи в стек называется «заталкиванием» (инструкция **PUSH), а** процесс чтения – «выталкивание» (инструкция **POP**).

Синтаксис для команды **PUSH**:
PUSH REG
PUSH SREG

PUSH memory
PUSH immediate
REG: AX, BX, CX, DX, DI, SI, BP, SP.

SREG: DS, ES, SS, CS.

memory: [BX], [BX+SI+7], 16-ти битовая переменная и т.п...

immediate: 5, -24, 3Fh, 10001101b, и т.п...

Синтаксис для команды **POP**:
POP REG
POP SREG
POP memory
REG: AX, BX, CX, DX, DI, SI, BP, SP.

SREG: DS, ES, SS, (кроме CS).

memory: [BX], [BX+SI+7], 16-ти битовая переменная и т.п...

Примечания:

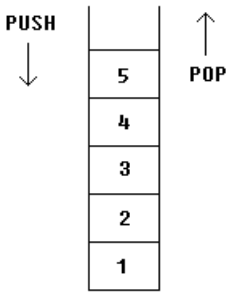
- Команды **PUSH** и **POP** работают только с 16-ти битовыми значениями!
- Примечание: **PUSH immediate** работает только на процессорах 80186 и выше!

Принцип работы стека

Стек использует алгоритм **LIFO** (Last In First Out - Последним пришел - первым ушел), это значит, что если мы поместим эти значения одно за другим в стек:

1, 2, 3, 4, 5

то первым значением, которое мы можем получить из стека, будет **5**, затем **4**, **3**, **2** и только потом **1**.



Очень важно применять равное количество команд **PUSH** и **POP**, иначе стек может быть нарушен и невозможно будет вернуться в операционную систему. Как вы уже знаете, мы используем команду **RET** для возвращения в операционную систему. Когда программа запускается, ее адрес записывается в стек (обычно это 0000h).

Команды **PUSH** и **POP** чрезвычайно полезны, т.к. для хранения данных обычно недостаточно только регистров. Вот выход из ситуации:

- Записать значение регистра в стек (используя **PUSH**).
- Использовать этот регистр в своих целях.
- Восстановить предыдущее значение регистра из стека (используя **POP**).

Перед загрузкой в стек содержимое регистра указателя стека SP инкрементируется, а после извлечения из стека декрементируется. По сигналу системного сброса в SP заносится начальное значение 07H. Для переопределения SP можно воспользоваться командой MOV SP, #d. Таким образом, стек может располагаться в любом месте резидентной памяти данных.

Стек используется для организации обращений к подпрограммам и при обработке прерываний, может быть использован для передачи параметров подпрограммам и для временного хранения содержимого регистров специальных функций. Подпрограмма должна сохранить в стеке содержимое тех регистров, которые она сама будет использовать, а перед возвратом в прерванную программу должна восстановить их значения.

Область памяти стека устанавливается при помощи регистров **SS** (StackSegment – сегмент стека) и **SP** (StackPointer – указатель стека). Обычно операционная система устанавливает значения этих регистров на начало программы.

Команда "**PUSH источник**" делает следующее:

- Вычитает **2** из регистра **SP**.
- Записывает значение **источника** по адресу **SS:SP**.

Команда "**POP приемник**" делает следующее:

- Записывает данные, размещенные по адресу **SS:SP** в **приемник**.
- Увеличивает на **2** значение регистра **SP**.

Текущий адрес указателя в **SS:SP** называется **вершиной стека**.

Для **COM**-файлов сегмент стека - это обычно и сегмент кода, а указатель стека установлен в значение **0FFFEh**. По адресу **SS:0FFFEh** записывается адрес возврата для команды **RET**, которая выполняется в конце программы.

Вы можете наблюдать за работой стека, щелкнув по кнопке [**Stack**] в окне эмулятора. Вершина стека отмечена знаком "<".

Примеры использования стека:

- для хранения локальных переменных подпрограмм;
- для сохранения содержимого регистров используемых подпрограммой;
- перед вызовом подпрограммы, которая тоже использует эти регистры;
- при возврате из подпрограммы исходное содержимое регистров восстанавливается.

Примеры:

```
ORG    100h

MOV    AX, 1234h
PUSH   AX; записать значение из AX
в стек.

MOV    AX, 5678h ;  изменить
значение регистра AX.

POP    AX; восстановить
первоначальное значение AX.

RET

END
```

Стек можно также использовать для того, чтобы поменять местами значения в регистрах:

```
ORG    100h

MOV    AX, 1212h; записать в AX
число 1212h.
MOV    BX, 3434h ; записать в BX
число 3434h.

PUSH   AX ;  записать значение AX в
стек.
PUSH   BX ; записать значение BX в
стек.

POP    AX;  установить в AX значение
BX.
POP    BX;  установить в BX значение
AX.

RET

END
```

Обмен данными происходит потому, что стек использует алгоритм **LIFO** (Последним пришел - первым вышел), поэтому когда мы помещаем в стек число **1212h**, а затем - **3434h**, то при обращении к стеку мы сначала получим число **3434h**, и только потом - **1212h**.

Практическая часть

Задание:

Занести в стек последовательность чисел от N до M, используя регистры и команды. А затем извлечь данную последовательность из стека в обратном порядке. При выполнении действий использовать команды ADD и MOV. Результат выполнения программы отследить в пошаговом режиме выполнения.

Варианты

№ варианта	N	M
1	1	3
2	2	4
3	5	7
4	7	9
5	11	13
6	12	14
7	3	5
8	4	6
9	12	14
10	8	10

Контрольные вопросы:

1. Что представляет собой стек?
2. Опишите область применения стека.

3. Опишите принцип работы стека. В чем заключается особенность использования алгоритма **LIFO** (Last In First Out)?

СПИСОК ЛИТЕРАТУРЫ

1. *Иванов, В.В.* Микропроцессорная техника : учеб. пособие / В.В. Иванов. Самара : Изд-во Самарского ун-та, 2019. 80 с.
2. *Келим, Ю.М.* Вычислительная техника. / Ю.М. Келим. М. : Академия, 2019. 368 с.
3. *Кобайло, А. С.* Арифметические и логические основы цифровых вычислительных машин : учеб. -метод. пособие для студентов специальности «Информационные системы и технологии (издательско-полиграфический комплекс)» заочной формы обучения / А. С. Кобайло. – Минск : БГТУ, 2014. – 76 с.
4. *Куль, Т.П.* Основы вычислительной техники : учеб. пособие / Т.П. Куль. Минск : РИПО, 2018. 241 с.
5. *Кушнер, Д.А.* Основы автоматики и микропроцессорной техники : учеб. пособие / Д.А. Кушнер, А.В. Дробов, Ю.Л. Петроченко. Минск : РИПО, 2019. 245 с.
6. *Луцик, Ю. А.* Арифметические и логические основы вычислительной техники : учебное пособие / Ю. А. Луцик, И. В. Лукьянова. – Минск : БГУИР, 2014. – 174 с. : ил.

7. *Сенкевич, А.В.* Архитектура ЭВМ и вычислительные системы : учеб. / А.В. Сенкевич. М. : Академия, 2018. 320 с.

8. *Фоминых, Е. И.* Арифметико-логические основы вычислительной техники : учебное пособие / Е. И. Фоминых, Т. Е. Фоминых, Ю. Л. Пархоменко. - Минск : РИПО, 2021. - 223 с.