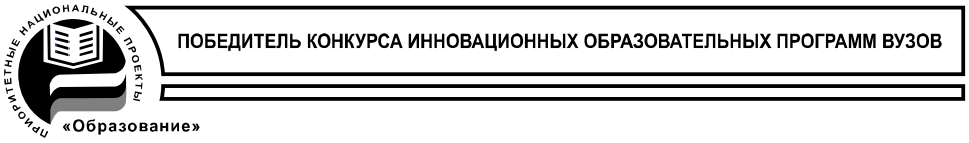
02.11.19
0,,,,,0,0,МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ



Организация ЭВМ как исполнителя алгоритма

Санкт-Петербург

2019

1

Введение.

Пособие содержит введение в алгоритмизацию как когнитивный процесс, относящийся к психологии мышления. Алгоритмы формируются в сознании на основе знаний, в том числе – фундаментальных. Интуитивное решение задачи формализуется в явных формах, позволяющих актуализировать мышление и , по существу тестируя алгоритм, сформулировать решение в некоторой теории и реализовать его исполнителем. Рассматривается интуитивная алгоритмизация, формализация и исполнение алгоритма компьютером. Для демонстрации логики алгоритмов используется программирование для ЭВМ общего назначения, программируемый логический контроллер (ПЛК), микропрограммирование.

В лабораторном практикуме задачи решаются интуитивно и формализованы с использованием алгоритмического языка Си с средствами моделирования. Показательна и эффективна в образовании система Keil (Интегрированная система проектирования (Integraited Development Environment), включающая графику Логического Анализатора для тестирования и временных измерений , имитатор внешних событий в виде Сигнальных функций при вводе.

Глава 1. Знакомство с когнитивной алгоритмизацией.

Глава 2. Обзор моделей вычислителй как исполнителей алгоритмов.

Принстонская и Гарвардская модели ЭВМ.

Глава 3. Алгоритмические задачи, основанные на вычислениях: ввод и вывод численных данных, арифметика, вычисление рекурсивных функций, булевские функции, задачи на графах.

Глава 4. Управление интерфейсами ввода/вывода. Система прерывания. Таймеры. Широтно-импульсная модуляция. Клавиатура. Аналого-цифровое преобразование.

Содержание

2.2.2.

1/ Память с произвольным доступом (Random Acsess Memory-Ram)

1)Память **Data.**

2)Регистры специальных функций - *SFR*

3) Битовая память Bdata.

2/ Постоянная (энергонезависимая) память (Read Only Memory – ROM[V]) .

3/ Расширенная память данных Xdata.

2.2.2.Управление программой.

2.2.3. Микропрограммная модель

III. Алгоритмические вычисления.

3.1. Ввод-вывод численных данных.

3.1.1. Преобразование целых чисел при вводе и выводе данных.

3.1.2.Преобразование дробных чисел при вводе и выводе данных.

Задания.

3.2. Машинная арифметика.

3.2.1. **Умножение.**

3.2.2. Деление.

Задания.

3.3. Вычисление функций.

3.3.1. Вычисление с числами с плавающей точкой.

3.3.2. Вычисление функций с числами фиксированной точкой.

1) Вычисление функций с десятичным масштабом 100.

2) Вычисление функций с двоичным масштабом $m=2^8$

Задания .

3.4. Булева алгебра и Вероятностная логика.

Задания.

3.5. Табличные вычисления.

Задания.

IV. Управление вводом и выводом в ПЛК.

4.1. Алгоритмическое управление прерываниями.

4.2. Алгоритмическое управление таймерами.

4.3. Ввод с клавиатуры.

Задания .

4.4. Широтно-импульсная модуляция (ШИМ).

Задания.

4.5. Аналого-цифровое преобразование .

3

I. Алгоритмическое мышление.

Мышление –**процесс познания** внешней **информации** с преобразованием ее в сознании как представление об объективной **реальности, что позволяет получать новую информацию в виде знаний, сохранять** их как опыт, способствуя осознанию **условий** для решения задач и формирования новых знаний. Мыслительная деятельность целенаправленна на решение какого-либо задания. Процесс мышления заключается в целенаправленном и целесообразном преобразовании действительности[1]

Человеку свойственно **алгоритмическое мышление**.

Алгоритм – набор инструкций и логика, определяющие операции с информацией и их порядок **исполнения** для получения новой информации.

Как метод решения задач алгоритм определяется общими свойствами:

1)**Дискретность** – последовательность выполнения **команд** (шагов) некоторым **исполнителем**.

2)Детерминированность – однозначнse по смыслу команды исполнителя

3)Завершаемость – исполнение алгоритма за конечное число шагов и достижение конечного результата

4)Массовость – применимость к множеству наборов значений исходных данных.

Подразумевается существование некоторой области применения с участием **человека** как исполнителя алгоритма. Для определения этого участия используется понятие **алгоритмическое мышление (алгоритмическая когнитивность** связана с психологией сознания [1,2,3]).

В определении алгоритма используются понятия – **информация, дискретность, алгоритмизация как процесс**.

Информация – подразумеваются конкретные осознанные сведения из некоторой области знаний. Для представления информации используются **дискретизация** и упорядочение ее во **времени**.

Человек живет в реальном непрерывном окружающем мире – содержание (смысл) внешних наблюдаемых событий или процессов не меняется, либо меняется непрерывно условно – **во времени**.

Далее для того, чтобы принять и осознать события или процессы необходимо время (периодичность, задержка,) и точность **(дискретность**) оценки (слышимость, видимость, разрешимость инструмента).

Если оценка (**измерение**) осуществляется периодически, то измерима **задержка времени** как информация: **есть или стало** (тепло, жарко, прохладно, холодно, ...), (много, мало), (0v, +2v, +3v, …) и др.

Если нас интересуют изменения смысла событий, то множество дискретных событий аппроксимируется как непрерывный процесс в непрерывном времени с дискретными значениями в реальном времени начала **процесса** и начала **измерения**. Предыдущие измерения в процессе упорядочены по времени и образуют упорядоченное множество измерений. Информацию, представленную числами, называем **данными.**

Если при исполнении алгоритма необходимо сравнение с предыдущими значениями, то их необходимо **запоминать, хранить и вспоминать.** Для этого требуется **функция памяти.**

Алгоритмизацию можно представить следующей диаграммой.

Рис.1.1. Диаграмма алгоритмизации.

В любой области приложения алгоритмическое мышление – интуитивное осознание информации как знания (изображение, вербальное описание), косвенным через измерения в численной форме, преобразованием в

5

графические и математические модели, которые допускают представление в естественно доступных интуитивно формах,

Необходимым элементом алгоритмизации **является исполнитель**, который выполняет некоторую совокупность детерминированных операций с знаниями, представленными данными (**мышление, тестирование, калькулятор, компьютер,схема, когнитивная система, киберсистема,..)**

Алгоритм и исполнитель объединены конструктивно и алгоритмизация – этап мыслительного процесса и **формализация**, позволяющая использовать модели и **теорию** (логика,графы,математика,графика , лингвистика) как форму записи алгоритма .

На схеме представлены основные блоки исполнителя алгоритма в виде программы , которая хранится в основной памяти и загружается в виде кодированной двоичным объектным кодом из блока ввода.

Исполнитель программы – быстрый процессор с поддерживающей высокую скорость вычислений быстрой памятью.

Результаты вычислений выводятся через блок вывода в устройства визуального отображения.

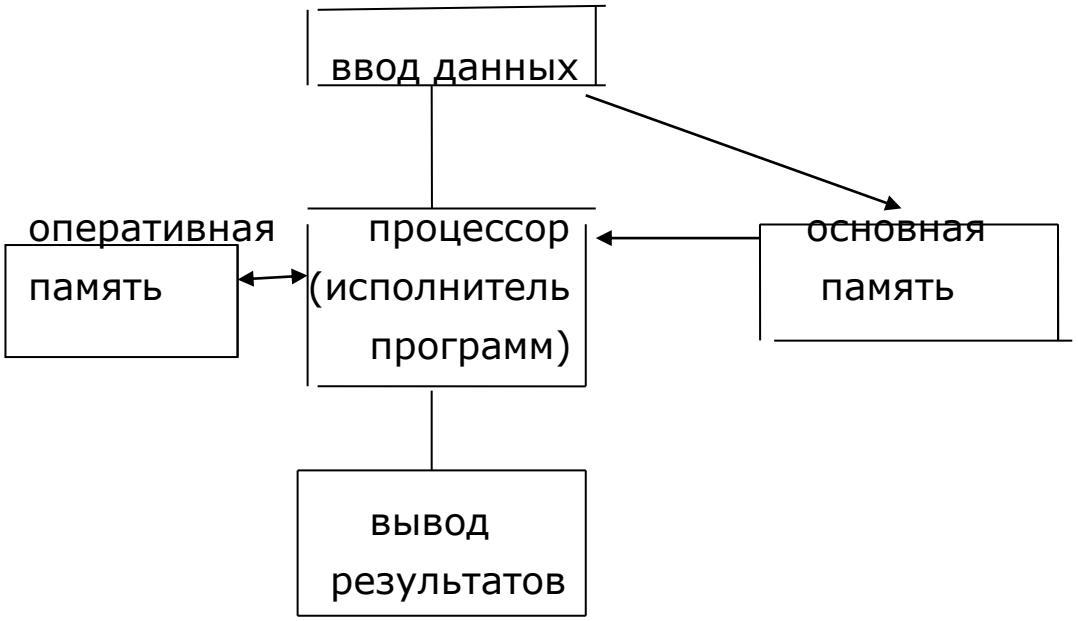


Рис. 1.2. Структурная схема компьютера как исполнителя алгоритмов (программ)

II.Компьютер как алгоритмическая исполнительная машина

Дж. фон Нейманом сформулирована **программная модель** универсального компьютера. ЭВМ конструктивно реализует в одной схеме с исполнителем и управляющую программу, что обеспечивает автоматическое исполнение алгоритма без участия человека.

Определение свойств вычислительной машины Неймана:

1. Вычислительные машины работают с числами, представленными в **двоичной форме**.

2. Вычислительный процесс, контролируется **управляющей программой**, представляющей собой формализованную последовательность исполняемых команд.

3. **Память** вычислительной машины осществляет **хранение данных и программ** в двоичном коде. Доступ к программам в памяти аналогичен доступу к данным. По типу данных, как целые числа в байтах, команды и данные не различимы, однако их различает назначение и информация, которые они представляют.

Ячейки памяти ЭВМ **адресуемые**. Таким образом функционируют в программировании переменные.

4. Предусмотрен уникальный порядок выполнения команд **условными операторами**. При этом они будут выполняться не в естественном порядке своей записи, а следуя указанным адресом.

Историческая классификация ЭВМ по направлениям [4,5]– Гарвардская и Принстонская архитектуры. В настоящее время ЭВМ различаются скорее по приложениям и назначению – персональные, вычислительные системы, вычислительные сети, встроенные применения(контроллеры)

2.1.ЭВМ общего назначения с Принстонской Неймановской)архитектурой

Наиболее близкой из современных ЭВМ по архитектуре является портативная Персональная мини ЭВМ (ПВМ)

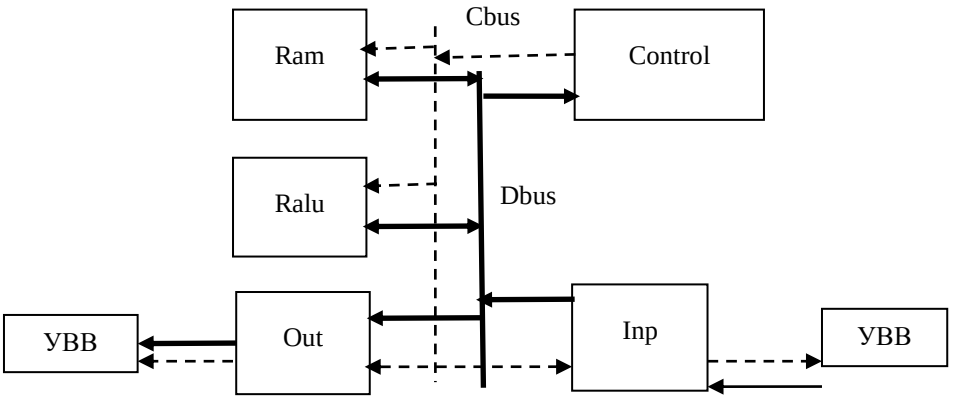


Рис. 1.3. Неймановская архитектура.

Память **Ram (Random Acsess Memory)** с адресным доступом общего назначения предназначена для записи, хранения и чтения двоичных кодов данных и программ. Режимы адресации не акцентируются – на уровне алгоритмического языка используется **прямой доступ** к памяти.

7

Блок **RALU** исполняет арифметические и логические микрооперации в командах процессора.

Блок микропрограммного управления **Control** реализует микропрограммы управления архитектурой по блокам.

Блок управления вводом **Inp (канал ввода)** осуществляет чтение (ввод) данных с внешних усройств ввода УВВ.

Блок вывода **Out (канал вывода)** управляет записью (выводом) данных во внешние устройства вывода УВВ.

Блоки управления Inp,Out осуществляют преобразования данных в соответствии с протоколами обмена данными по стандартным интерфейсам.

Необходимые информационные связи (интерфейсы) между блоками представлены шиной данных **Dbus,** по которой происходит также чтение команд из Ram, и шиной микропрограммного управления **Cbus**..

Порядок исполнения команд (операторов языка) определяет **Логика**[1] алгоритмов. В каждый дискретный момент времени, следующий шаг алгоритма однозначно определяется **состоянием памяти** исполнителя при выполнении текущей операции с данными и новой целью преобразования на следуюшем шаге. Логика определяет **последовательное**, **разветвленное**, **циклическое** и **рекурсивное** (иерархическое) исполнение команд зависит от выбранной при интерпретации информационной модели.

В процессе **алгоритмизации** могут быть использованы формальные модели преобразования данных, удобные для ручного и машинного тестирования и определены ресурсы исполнителя в виде программной модели. Эта информация обобщается языками программирования высокого уровня.

В частности, модель может быть представлена: численными или логическими формулами, алгоритмической схемой, графикой, функциональными схемами. Алгоритм **универсален** и применим к **множеству** значений наборов входных данных.

Промышленные технологии алгоритмизации подтверждают, что процедурность, используемая в **языке программирования C++** как обращение к стандартным библиотечным функция, эквивалентна исполнению алгоритмических команд - эффективна, естественна и равнодоступна.

Алгоритмы и алгоритмизация в различных интерпретациях предполагают разнообразные методы демонстрации преобразований данных и в обзоре Д. Э. Кнута, по существу, число их не ограничено.

Следуя общей классификации Кнута[2] (т.1 – Основные численные алгоритмы, т.2 – Получисленные алгоритмы, т.3 – Сортировка и поиск, т.4 –

---

[1] *Замятин, А. П.* Математическая логика и теория алгоритмов: учеб. пособие / А.П. Замятин. – УрГУ: Екатеринбург, 2008.

[2] *Кнут Д. Э.* Искусство программирования. Том 1. — 3-е изд. — М.: Вильямс, 2017.
Кнут Д. Э. Искусство программирования. Том 2. — 3-е изд. — М.: Вильямс, 2017.
Кнут Д. Э. Искусство программирования. Том 3. — 3-е изд. — М.: Вильямс, 2017.
Кнут Д. Э. Искусство программирования. Том 4. — 3-е изд. — М.: Вильямс, 2018.

комбинаторные алгоритмы). В пособии рассматриваются решения отдельных алгоритмических задач из этих классов с акцентом на алгоритизацию.

2.2. ЭВМ с Гарвардской архитектурой

Основное отличие ЭВМ – иерархическая организация памяти с различными режимами адресации и прямое управление периферией УВВ.

Память различается как постоянная (долговременная) память ROM для хранения и исполнения программного кода и память данных типа Ram, предполагающая режимы записи, хранения и чтения.

Соответствующие архитектуры относятся к **программируемым логическим контроллерам (PLC, ПЛК)** и отличаются также:

1) относительно простой и доступной для алгоритмизации схемотехникой, прямое обращение к которой в ряде примеров упрощает алгоритмизацию.

2) многообразием расширений и модификаций, сохраняющих ядро ПЛК, что позволяет демонстрировать исполнение алгоритмов в различных конфигурациях ввода-вывода

3) наличием эффективных средств программирования и исполнения алгоритма в симуляторе с возможностью демонстрации состояния памяти в числах и временных диаграмм функциональных зависимостей .

4) наличием средств макетирования внешних схем ввода-вывода.

Библиотека Keil содержит сотни модификаций ПЛК прототипа mcs51 фирмы Intel, разработанные десятками фирм разных стран еще в 80-е годы прошлого века. ПЛК совершенствуются и выпускаются в виде прототипов для оперативного применения.

В современных ПЛИС-технологиях доступны **программный и микропрограммный уровни** исполнения и алгоритмизации .

В компьютерах каждая команда реально исполняется схемами с многотактным **микропрограммным** управлением. Что определяется как двухуровневое программное и микропрограммное исполнение алгоритма.

Программный уровень– исполнение алгоритма архитектурой компьютера программой в системе команд ЭВМ.

Микропрограммный уровень – исполнение алгоритма функциональной схемой ЭВМ, где каждая команда ЭВМ представлена **микропрограммой.**

Среда программирования ПЛК и возможность проектирования его модификаций в САПР на основе прототипа позволяют организовать **заказное проектирование** с микропрограммным управлением.

9

Для демонстрации задач алгоритмизации на микропрограммном уровне и при работе с битами данных используется в качестве прототипа mcs51[3] и язык C51 в IDE Keil[4].

Архитектуру контроллера в C51 определяем, как **программную модель высокого уровня**. Детали построения и работы ПЛК в системе команд доступны на уровне **Ассемблера A51,** в котором программная модель позволяет использовать как исполнителя структурную схему ЭВМ **.**Основным методом **тестирования** алгоритмов является исполнение программы в симуляторе.

В ПЛК используется иерархическая организация памяти с косвенной и неявной адресацией с учетом типа памяти и типа данных.

Применение стандартного языка C++ к ПЛК имеет ограничения (не доступна специфическая адресация в неоднородной и совмещенной памяти в применении к mcs51), поэтому используется **эмуляция** разных типов памяти, заменяющая их структуру в C51 и A51.

Диаграмма mcs51 (рис.2.3.) представляет в виде программной модели доступную иерархию памяти и интерфейсы ввода-вывода в C51.

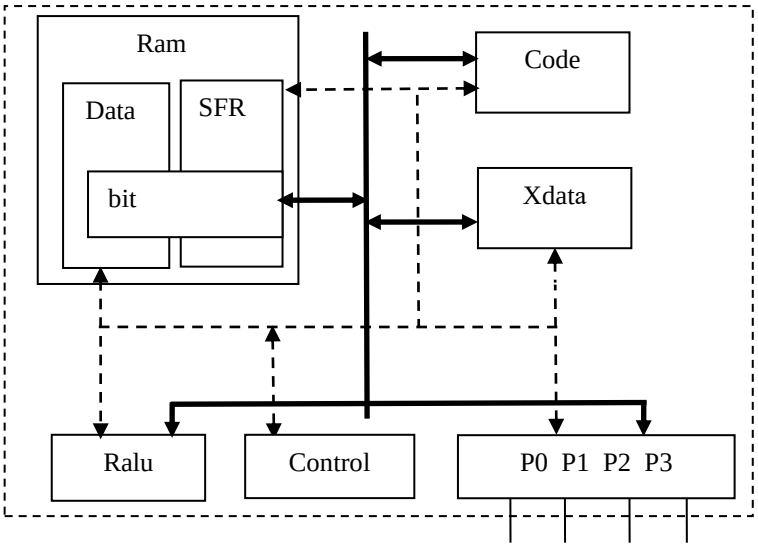


Рис. 1.4. Программная модель в C51

В отличие от Гарвардской модели в C++ разделены адресные пространства памяти программ **Code** и данных **(Data, Xdata**), ввод-вывод представлен внешними параллельными интерфейсами (портами P0, P1, P2, P3).

1. Организация памяти

1) Память с произвольным доступом (Random Acsess Memory-Ram)

Адресуемая память (с раздельными входом/выходом или с общей шиной данных)

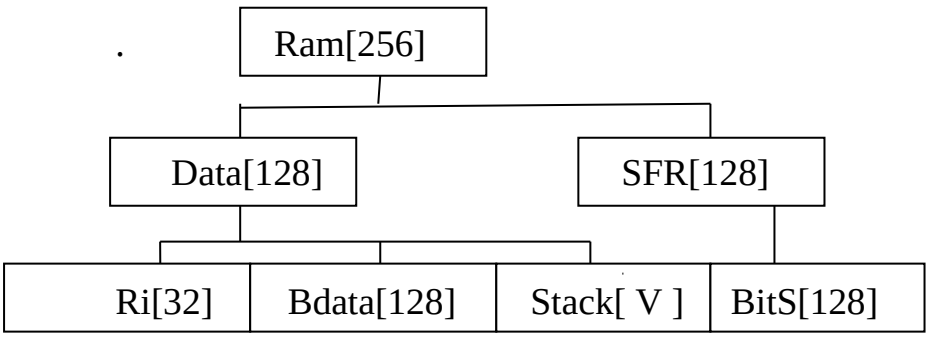
[3] *Магда Ю.С.* Микроконтроллеры серии 8051: практический подход. — М.: ДМК Пресс, 2008.
[4] Keil µVision MDK-ARM 5.20.

В С51 не акцентируется адресный доступ - используется **прямой** доступ к данным в памяти ЭВМ. Идентификатор переменной обозначает доступ к формату данных безотносительно к режиму **(адресации)** доступа.

Свойства Ram[V] :

n -разрядность слова памяти, m-разрядность адреса V -объем памяти (число доступных слов памяти)

Две области памяти **Data и SFR** можно рассматривать как непрерывную Ram[256] **,** которая имеет иерархическую организацию и может быть представлена в виде диаграммы.



1)Память **Data** неявно подразумевается в С51 как общезначимая

Типы и форматы данных определяются стандартно для языка С:

char x, int y, long z, unsigned char,

type definition unsigned char uchar //определение типа

Применимы стандартные арифметические операции {+,-,*,/,^,>>}

char S[128]; // n=8, V=128 байт

char *x= S[0]**;** //x **у**казатель-адрес **pointer** доступа к сегменту в памяти **Data**

Операции с указателем

x++; x=x+5; //модификация адреса

*x++; *x=*x+5; //модификация значения по адресу

Переменные в памяти Data

int y; //формат 2 байта

long z; //формат 4 байта со знаком

При размещении многобайтовых форматов данных в памяти используется последовательное со старших байтов к младшим **(BigEndian ~ BE**) - соответственно с младшего адреса к старшему.

Long S 0x048FF4EA ~ char S[4]

	a3	**a2**	**a1**	**a0**
	04	**8F**	**F4**	**EA**
адреса в Data	**8**	**9**	**A**	**B**
	S[0]	**S[1]**	**S[2]**	**S[3]**

Доступ к адресу переменной с прямым доступом

char x; char y=&x;

1

Преобразование прямого адреса переменной в указатель

char x; char *y=&x;

Преобразование форматов

int *xx= S[10]; char *y=(*char) S;

Все другие сегменты памяти Ram (SFR, Ri,BitD,Stack) в С51 доступны только через адресацию по указателю.

2)Регистры специальных функций - *SFR* 128 байт

Реализуют прямой доступ к аппаратным регистрам.

В их числе 8-разрядные прямо доступные по именам порты ввода-вывода **{P0, P1,P2,P3}.**

P2=0x55; P2=x; //интерпретируем как ввод и вывод при работе в симуляторе

P3=P2+P0; //P2,P0-ввод и P3-вывод

Регистры RALU (**ACC, B)** не доступны и не могут быть переопределены.

Управляющие регистры прямо доступны.

Доступ определяется загрузкой адресного файла

#include <reg51.h> { reg515.h, ADuC812.h,..из каталога C51/INC}

Свободные регистры в SFR могут быть определены

sfr TT=0x95 – свободный адрес в SFR

sfr16 y=0xA1 определяет адрес двух смежных регистров в SFR(Big Endian-размещение).

3) Битовая память Bdata – поле из 128 битов. Биты 0….127 хранятся в ячейках Data. В С51 применимы логические операции с прямым доступом к битам **(&,|,~,^)**,

bit x1,x2; //биты с прямым доступом по идентификатору

char bdata mem,meme ; //доступ к битам в именованных форматах char, int,long

sbit y1= mem^0; //0-ой бит 0-байта **mem** в сегменте **bdata**

sbit z2= **meme^2;** //2-ой бит 1-байта mem

В регистрах SFR прямо доступны резервированные управлением биты

PSW=C.AC.F0.RS1.RS0.OV.-.P - резервированные имена битов – признаки арифметических операций в регистре состояний PSW с прямым доступом, кроме С и АС.

Логическая функция y1=P | OV&x2;

4)Адресуемая постоянная (энергонезависимая) память (Read Only Memory –ROM[V]) .

В программной модели mcs51 память **Code** типа Rom объемом до 65 кб используется для записи и хранения программного кода и таблиц с константами.

Определения констант в С51

char code x= 0x55;

char code y[]=”text”;

Доступ по **i**-индексу **char xx= x[i];**

int yy=y[i++]; //чтение двух байтов

Указатель-адрес **pointer** к константе в памяти **code**

char code *x= 0x77**;**

char code *S[]=”text”;

Операции с указателем

x++: x=x+5; xx++;

чтение константы

char y=*x;

int y=*(int)x;

5) Расширенная память данных Xdata типа **Ram,** до 65 кбайт адресное пространство

Сегмент памяти **xdata с прямым лоступом**

char xdata mem[0x200];

int x= mem[0x100] ;

Указатель-адрес **pointer** к сегменту в памяти **xdata**

char xdata *x= &S[0]**;**

char xdata *S[10];

int xdata *xx= &S[0]**;**

Доступ к адресу переменной

char xdata x; int xdata y=(int)&x;

Преобразование прямого адреса переменной в указатель

char xdata x; char *y=(int)&x;

Преобразование форматов

int *xx= S[10]**; char *y=(*char) S[0];**

Страничное размещение данных

char pdata x; //переменные размещаются со смещением в странице

P2=0x50; //задается адрес страницы

x= 0x55;

6) Параллельные цифровые порты ввода-вывода.

В mcs51 прямой доступ к ввод-выводу представлен цифровыми 8-битовыми портами **P0-P3** .

Порт **P0 двунаправленный** , может в реальной схеме использоваться для ввода и вывода и не требуют настройки.

Порты **P1,P2,P3** в реальных схемах включены как **однонаправленные** и могут настраиваться на соответствующий режим обмена побитно.

Порты (Рис.1.4) включают регистр с прямым доступом при записи (вывод). При этом состояние регистра отображается в состояниях контактов Pin7..0

1

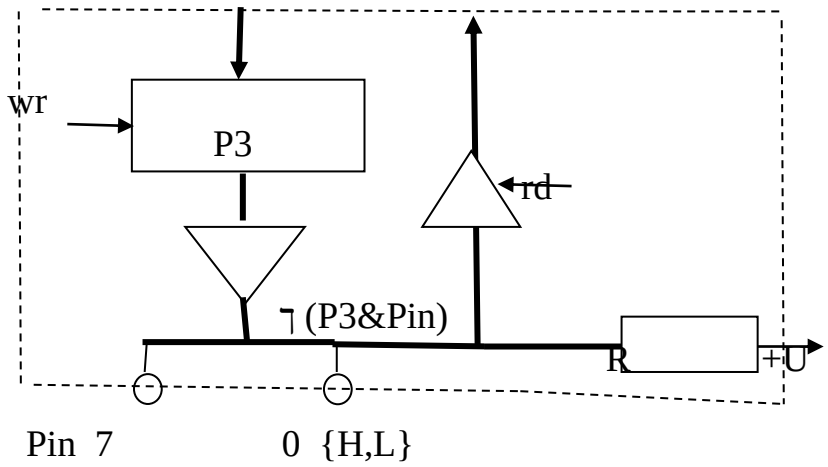


Рис.1.5. Схема портов P1,P2,P3

В состоянии ввода контакты Pin7..0 отключены от регистра и находятся в состоянии, определяемом значением сигналов на внешних линиях, подключены по схеме с открытым коллектором. При этом ввод через порт осуществляется сигнала ┐(P3&Pin)=(┐P3 v ┐ Pin) - несовместимы ввод и вывод. Вывод осуществляется записью в регистр P3= 0x55; и ввод в схеме невозможен -линии связи доступны включением только для вывода

P2=0x55; //вывод двоичного константы из памяти Code через порт P2

Программа ввода байта данных

char bb;

//состояние готового к вводу порта P1=0xFF подразумевается

bb=P1; //чтение**(**ввод) с контактов порта и сохранение в памяти Data

Двоичный код интерпретируется в положительном кодировании сигналами (H~1, L~0). Порт хранит беззнаковый двоичный код и приобретает смысл типа данных при записи в памяти Data.

Через порты также передаются сигналы управления периферией (rd,wr), адреса и данные внешней памяти Xdata.

Схема внешнего интерфейса PLC

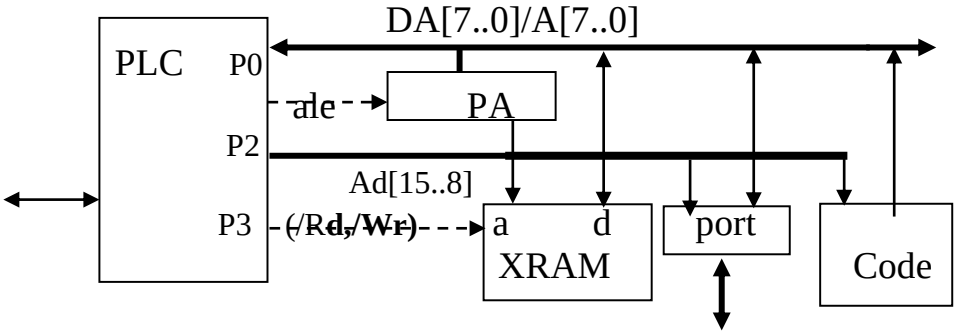


Рис.1.6. Мультиплексированная системная шина

Передача младшего байта адреса по шине P0 синхронизируется сигналом **ale,** который записывает адрес в регистр адреса PA.

Для управления внешними расширенными портами совмещаются адреса портов с адресацией внешней памяти Xram ..

Управление внешней памятью Xram и программной Code разделены (логически и электрически различимы) и, соответственно, разделены и независимы адресные пространства памяти данных и программ (Принстонская архитектура)

7) Управление программой.

Операторы управления программой стандартные

goto метка;

if (условие) оператор ; **else** оператор **;**

while(условие) оператор **;**

do { оператор **} while (условие) ;**

for (i=0;i<N; i++) оператор **;**

```
switch (ss) {    //декодирование двоичного кода ss
            case  0x55 of :   [      ]; break;  //ss=0x55
                        ……..

            case  0x66 of :   [      ]; break;
                          default: [   ];
                        }
```

В С51 могут быть определены **рекурсивные (рекуррентные) алгоритмы**.

Reentrant (реентранная) функция декларируется как возможная для многократного (повторного) обращения рекурсивно.

Вывод формулы в математике является прямым доказательством правильности алгоритма.

Например, вычисление факториала определяется рекурсивной функцией

F(0)=1; F(i) = F(i-1)*i

рекурсивная программа

```
        int f=1;     char i=0;
    fact(char n) reentrant
      {  if(i<n) {i++;f*=i;
            return fact(i);}}
   main( ){  fact(5);}
```

Эквивалентная **циклическая программа** вычислений

```
char n,i=0;
int f=1;
main()
{ n=5;
  while(i<=n)
     {i++; f*= i; } }
```


1

2.3.Микропрограммная модель .

В алгоритмизации на микропрограммном уровне исполнителем предполагается функциональная схема, включающая управляемые элементы памяти (Ram, Rom, регистры, триггеры), функциональную логику, арифметику (+,-,*,/). При этом можно использовать максимально доступное параллельное исполнение и, соответственно, миниальное время исполнения алгоритма.
При алгоритмизации в С алгоритм рассматривается как одна функциональная микропрограмма с необходимыми ветвлениями , обозначаемыми в языке С метками.
Функциональная микропрограмма может быть записана на алгоритмическом языке С51 или более общим исполнением в С++ .

III. Алгоритмические вычисления.

3.1. Алгоритмические преобразования данных при вводе и выводе.

Входные численные данные могут быть представлены **в естественной текстуальной форме в ASCII** при вводе с клавиатуры или текстовыми файлами, двоично-десятичными кодами, двоичными кодами с датчиков. Цель алгоритмического преобразования – перевод из естественных форм записи чисел в машинные двоичные коды и обратно.

3.1.1. **Форматы машинных данных.**

Форматирование машинных данных в фиксированных форматах ЭВМ затрудняет анализ погрешностей вычисления и работу в широком диапазоне значений данных . В больших компьютерах первого/второго поколений (IBM 360/370) предусмотрены различные системы арифметических команд для различных типов данных.

1)**В научных расчетах** используются форматы с плавающей точкой (**float**.double). Точность вычисления в этих форматах определяется как относительная погрешность в процентах **d=(x*100)/D**, где x-текущее значение в десятичной системе и D-диапазон значений в формате. Погрешность возникает при выравнивании порядков и переводом в формат с естественной запятой. В вычислениях с изменяемыми масштабами в фиксированных форматах погрешность необходимо оценивать (применимы все арифметические операции +,-,*,/).

2)**Стандартные вычисления** в форматах с фиксированной точкой (целые числа **char,int,long**) “стандартная система команд”. Вычисления в целых числах в двоичном масштабе(применимы все арифметические операции +,-,*,/).
Операции (+,-,*) точные, но деление чаще всего дает погрешность в виде остака и, соответственно, вносит заметную погрещность в вычислениях. При вводе и выводе дробных чисел в десятичном масштабе погрешность определяется остатком при делении и многократно возрастает при последующих вычислениях –необходимо оценивать и, по возможности, компенсировать.

3)В **экономических расчетных задачах**, связанных с бухгалтерией и бизнесом , актуально прямое использование переменных форматов с **естественной запятой в десятичной системе счисления и положительных чисел без знака**, где удобен как исполнитель обычный **калькулятор.** Машинные данные кодируются строками в ASCII или в стандартных двоично--десятичных кодах с заданной длиной L строки и естественной запятой. Точность вычислений при фиксированной запятой абсолютная .
Основные арифметические операции (+,-,*,/) выполняются со строками 4-х битовых 2/10 кодов десятичных цифр, операции (*, /) имеют смысл, если соответствующие сомножитель и делитель целые.

Примером такой распространенной бухгалтерской системой является 1С.

3.1.2.Ввод и вывод.

Таким образом, **ввод** предполагает два этапа и две алгоритмические задачи:
1) чтение данных в исходных форматах с входных портов и сохранении их в памяти ЭВМ,
2)преобразование во внутренние машинные двоичные форматы

Соответственно, **вывод** – две алгоритмические задачи :
1) преобразование данных в машинных форматах двоичного кодирования в исходные для устройств вывода .
2)вывод данных обычно посимвольный во внешние устройства визуализации

Исполнителем алгоритмов ввода и вывода в даном случае является ПЛК, связанный с внешним миром цифровыми портами.

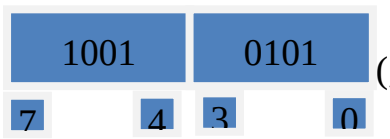
Десятичная и двоичная системы счисления позиционные однородные являются формальным способом записи информации о количестве(целые) и результатов измерений с заданной точностью могут быть представлены дробями.

Алгоритмизация ввода/вывода использует на этапе формализации стандартные форматы хранения данных в ЭВМ.

1.Ввод и вывод целых чисел.

1) Форматы двухразрядного целого десятичного числа в 8-разрядном двоичном порте

(,)



(,)

2)Преобразование целых 10/2 выполняется при вводе пересчетом **количества N** в двоичной системе по формуле определения чисел в позиционной однородной системе счисления
При вводе десятичного числа $N=a_1a_0$

$$\mathbf{N=B_2=(a_1 *10 + a_0)_2 = (x>>4)*10 + (x\&0x0f);\ \}}$$

1

Преобразование 2/10 целых при выводе выполняется делением двоичного числа на основание 10.

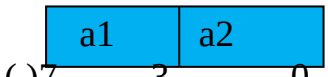
Если $\mathbf{B_2=(a_1 *10 + a_0)_2}$, то обратное преобразование целая часть $\mathbf{a_1 = B_2/10}$ в остатке $\mathbf{a_0= B_2\%10}$ и

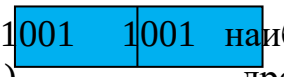
P1=((B2/10)<<4) | (B2%10);

2. Ввод и вывод дробных чисел.

1)Информация (**количество N**) в **записи** 2-разрядного дробного десятичного числа $\mathbf{A=0,a_1a_2}$

$$\mathbf{N= \sum a_i d^{-i} = a_1 10^{-1} + a_2 10^{-2}}$$

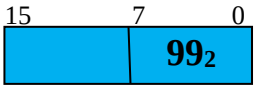




наибольшее десятичное дробное 0,99

2)Десятичное дробное число A= $\mathbf{0,a_1a_2 =0,99}$ рассматривается как целое $\mathbf{N= a_1a_0 = 99}$ с масштабом 10^2

$$N= \mathbf{a_1a_0 = (a_1 10^{-1} + a_2 10^{-2}) *100 =}$$

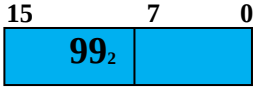


Применяя метод преобразования целых в двоичную систему , получим **8**-разрядное двоичное целое число $\mathbf{N_2 =(A*10^2)_2}$

Для получения дробного двоичного 8-разрядного числа необходимо разделить целое N_2 на масштаб $\mathbf{10^2}$ или $\mathbf{Q=N_2 /10^2}$ **в** двоичной системе, но это невыполнимо, так как $\mathbf{N_2 < 100}$ и Q=0 .

3}В целочисленной арифметике требуется сначала выполнить масштабирование с двоичным масштабом ($\mathbf{N_2*2^8}$) ,

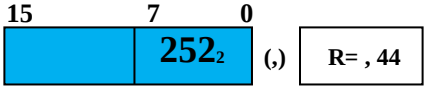
$$\mathbf{N_2*2^8 =}$$



$\mathbf{= 25\,244}$

4) $\mathbf{Q=(N_2*2^8)_2 /100 = Q_2*2^8}$ и результат дробное двоичное 8-разрядное в масштабе $\mathbf{2^8}$ как целое в 16-разрядном формате.

Q=

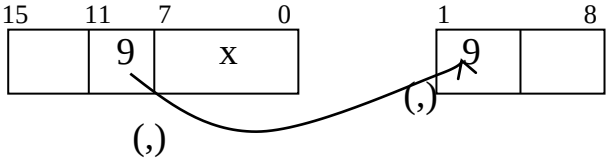


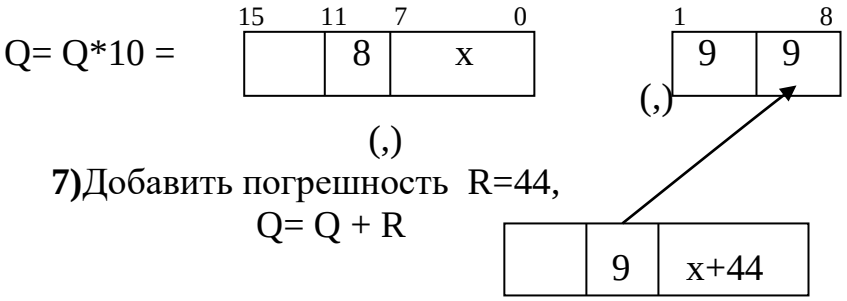
остаток

5)Вывод дробных чисел с преобразованием 2/10

Дробное в формате байта интерпретируется в масштабе 2^8 как целое двоичное и преобразуется последовательным умножением на 10 для получения десятичных цифр дробного числа в масштабе 2^8

Q= Q*10 =





7)Добавить погрешность R=44,
Q= Q + R

(,)

Программа ввода и вывода дробных чисел в С51.

```
#include <reg51.h>
unsigned int N,Q,S;
unsigned char R,Z;
main(){  //  ввод P0=0,99
  N=P1=(((P0&0xf0)>>4))*10 + (P0&0x0f);   //шаги 1 и2 – дробное как целое
  N<<=8;    //масштаб  2^8
  P2=Q=N/100; P3=R=N%100;  //преобразование в дробное с масштабом 2^8
  S =Q*10; Z=(S&0x0f00)>>4;  //старшая десятичная цифра  дроби  a1=9
     Q=S&0xff;
  S=((Q*10+R)&0x0f00)>>8;  //добавление остатка в масштабе 2^8
                                  //младшая цифра дроби  a2=9
  P1=Z=Z|S;  //вывод P1=0,99
  while(1);
   }
```

Таким образом, к дробным или к смешанным формам чисел с использованием масштабирования применимы одни и те же арифметические операции. При выводе учитывается масштабирование для дробных и смешанных исходных данных.

В вычислениях в форматах с фиксированной точкой накапливается абсолютная погрешность, которую трудно контролировать. Достоинством алгоритмов вычислений с фиксированной точкой (в форматах целых чисел) является высокая скорость вычислений.

По этой причине в общем случае используются форматы с плавающей точкой, в которых при вычислениях контролируется погрешность автоматическим изменением масштаба (порядком числа).

1

3.1.3.Суммирование десятичных чисел .

Схема сложения битов строится по таблице истинности и расширена для различных форматов в схемы с последовательным или ускоренным переносом многоразрядных сумматоров. Любая Эвм выполняет суммирование двоичных кодов и соответственно, шестнадцатеричных чисел одной командой.

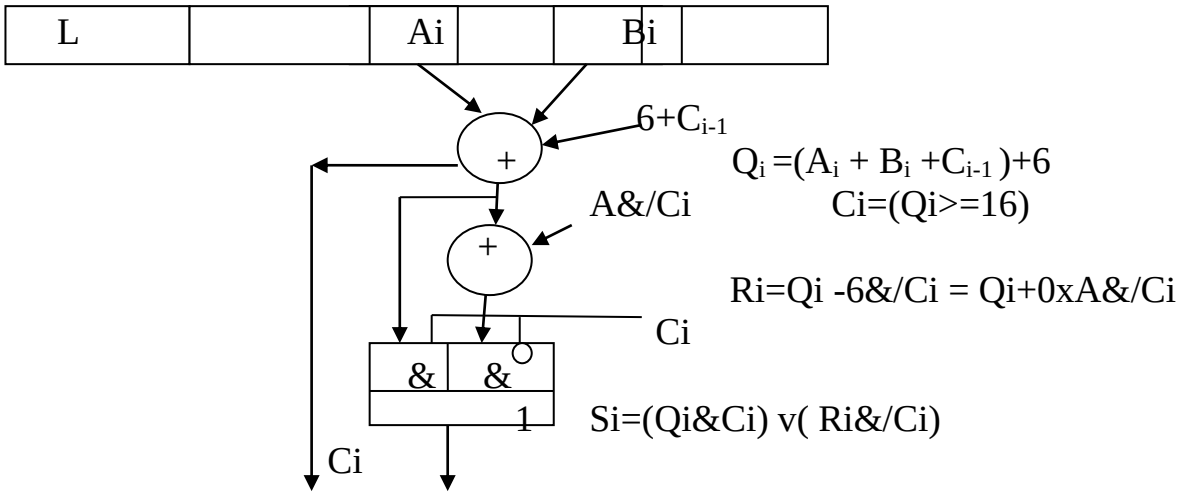
В схеме десятичного сумматора с использованием двоичного должна быть использована коррекция с учетом различия переносов.

Перенос в HEX-коде имеет вес 16, а в десятичной вес 10. Разность весов компенсируется добавлением при суммировании двоично-десятичных кодов константы (6).

Если формируется перенос, то естественно вычитается из суммы константа 16 и результат совпадает с двоично-десятичным кодом цифры результата.

Если перенос отсутствует, то необходимо вычитание избыточной константы 6 в сумме.

Схема десятичного сумматора двух цифр с входным переносом C_{i-1} формируется результат S_i и выходной перенос C_i



В схеме используются два последовательных 4-хразрядных двоичных сумматоров, которые используются для коррекции результата двоичного сложения с преобразованием в десятичный результат.

Алгоритм последовательного суммирования десятичных чисел. Алгоритмизация как интуитивный процесс с одновременным тестированием ориентирована на последовательное поразрядное суммирование по предлагаемой схеме исполнителя.

1.Выравнивание размещения запятых в форматах сдвигом одного из чисел вправо. В совмещенных форматах при вычислениях позиция запятой фиксируется . Абсолютная точность сохраняется Переполнение интерпретируется как расширение формата.

2. Последовательное суммирование по схеме с коррекцией предполагаемого HEX-переноса . Десятичные цифры представлены двоично-десятичными HEX-кодами

Q1=(A_i + B_i +C_{i-1})+6= **0x9,9 +0x9,9+0x6,6= 0x13,2+0x6,6= 0x19,8**

C_1C_0 интуитивно учитываются Hex-переносы в разрядах Коррекция 2/10 результата не требуется.

Q2= Q_i =(A_i + B_i +C_{i-1})+6=**0x2,5+0x5,5+0x6,6= 0x7,A+0x6,6=0xE,0**,

интуитивно HEX перенос C0=1 в младщем разряде и в старшем C1=0 требуется коррекция избыточного кода в младшем разряде.

3. Для Q2 требуется коррекция **Ri = Qi+0xA&/Ci** .

R2= **0xE,0-0x6,0=0xE,0+ 0xA,0= 0x8,0**

где 0xA=0x10-0x6 дополнительный код (-6)

4. Признак Ci (перенос в старшем разряде суммы Q) обозначает расширение формата значением неравным нулю (увеличение длины формата и смещение запятой в формате с переменной длиной L+Ci).

В примере C1=1, S=0x198, занимает 3 разряда , длина числа L=3 цифры.

Формат целого двоичного

```
unsigned int Q1,R2;
  Q1= (99+99)/10       //=19.0
  Q1= (99+99)%10     //=0.8

  R2= (25+55)/10   //=8.0
```

Формат с плавающей точкой

```
float  Q1,R2;
Q1= 9,9+9,9  //=19,8
R2= 2,5+5,5   //=8.0
```

3.1.4.Вычитание десятичных чисел .

В экономических расчетах знаки операндов всегда положительные. Знак результата сложения или вычитания в экономике должен быть положительным. Вычитание без знака выполняет схема прямого десятичного вычитания, которая может быть построена как схема последовательного или параллельного (с ускоренным переносом) вычитания двоичных разрядов по таблице истинности.

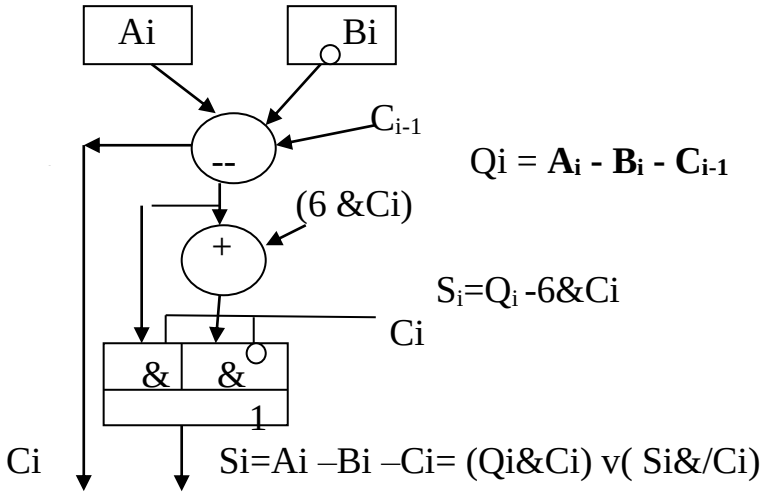
1)Вычитание тетрад эквивалентно в двоичной системе счисления соответствующим операциям в шестнадцатеричной системе счисления с формированием межразрядного HEX-заема .

2) HEX-заем имеет вес 16, а в десятичный вес 10. Разность весов компенсируется при вычитании десятичных цифр в HEX-коде добавлением константы 6 , если был заем. Схема HEX-вычитания десятичных цифр с заемом C_{i-1}

Q_i =A_i -B_i - C_{i-1}

2

$S_i=Q_i$ -0x6&Ci, если заем Ci=1



Алгоритм последовательного вычитания десятичных чисел

1. Содержание первого шага совпадает с этим же шагом алгоритма сложения
2. Последовательное интуитивное вычитание по схеме с HEX-заемами .

Qi= Ai-Bi=9,0-4,5 =0x9,0-0x4,5 =0x4,B

3.Интуитивно заем $C_0=1$

Si=Qi -0x0,6=0x4,B-0x0,6=0x4,5

Формат целого двоичного

```
unsigned int Q1,R2;
  Q1= Ai-Bi= (90-45)/10;  //=4.
 R1=  (90-45)%10;    //=.5
```

Формат с плавающей

```
float  Q1,Ai,Bi,R2;
Q1= (90-45)/10    //=19,8
R2= (25+55)/10   //=8.0
```

Задания. Разработать программу последовательного суммирования или вычитания с естественной запятой в ASCII коде. Выполнить интуитивное тестирование при исполнении в переменном формате. Тестировать исполнение также в форматах float, int.

1. **56,921 + 44,22**
2. **57,921 - 44,22**
3. **42,52+156,24**
4. **156,24- 42,52**
5. **21,155 + 33,99**
6. **33,99-21,155**
7. **253,52 + 55,91**
8. **253,52 - 55,91**

9. 95,77 + 24,52
10. 95,77 - 24,52
11. 38,67+29,50
12. 38,67-29,50

3.2. Задача преобразования кодирования текстовых строк

. Алгоритмизация в С++ и С51 с исполнителем ЭВМ.

- определить форматы данных
- организация памяти данных
- функциональная схема преобразования
- программирование в С++ и С51

Пример. Постановка задачи вербальная.

1) **“**Строка данных (десятичное число в ASCII-коде) загружена при вводе в память данных . Для выполнения преобразования число считывается посимвольно, переводится в двоичную систему счисления и в формате целого 16-разрядного числа без знака преобразуется побитно в текстовую строку“

“125” → “01111100”.

Исходный массив цифр в ASCII y[5]=”0125” – 4 цифры и байт с символом конца строки при вводе размещается в памяти Data[128].

1.Определить форматы данных и размещение в памяти ЭВМ

2.Преобразование десятичного числа в Big Endean-записи в двоичную запись при вводе пересчетом в двоичной системе

3.Преобразование для вывода и контроля ввода в символьную запись в ASCII последовательным чтением бита кода и преобразованием в ASCII ***B[i] = S(i) | 0x30.***

Алгоритмизация интуитивно сопровождается формализацией, необходимой для восприятия и интуитивного тестирования.

В данном случае для формализации используется **алгоритмическая схема** преобразования, отображающая интуитивно выбираемые форматы данных, элементы памяти, функциональные преобразования и логику доступа к данным в преобразованиях данных (рис. 2.5). Алгоритмическая схема учитывает организацию памяти в контроллере mcs51 и эмулируется в С++

1.Форматы данных

Входная строка- ASCII-код с адресом y[i] 4байта, доступ к тетрадам i=0,1,2,3, со стороны старших разрядов

Максимальное двоичное целое число $2^{13} < 10^4-1 < 2^{14}$ занимает 16 бит стандартный формат

2

2.Перевод в двоичное число. по схеме Горнера, используя рекуррентную формулу пересчета

N = “125” = $a_2a_1a_0$ = $a_2*10^2 + a_1*10^1 + a_0 = \sum a_i 10^i$ =

= ((((0*10 + a_2)*10 + a_1)*10 + a_0. →

$S(i+1) = S(i)*10 + 0x0f \,\&\, a_{n-i}$ $S(0) = 0$, $i = 0, \ldots, 3$ – позиция разряда

3. Преобразование двоичного S в символьную запись в ASCII последовательным чтением бита кода и ***B[i] = S(i) | 0x30.***

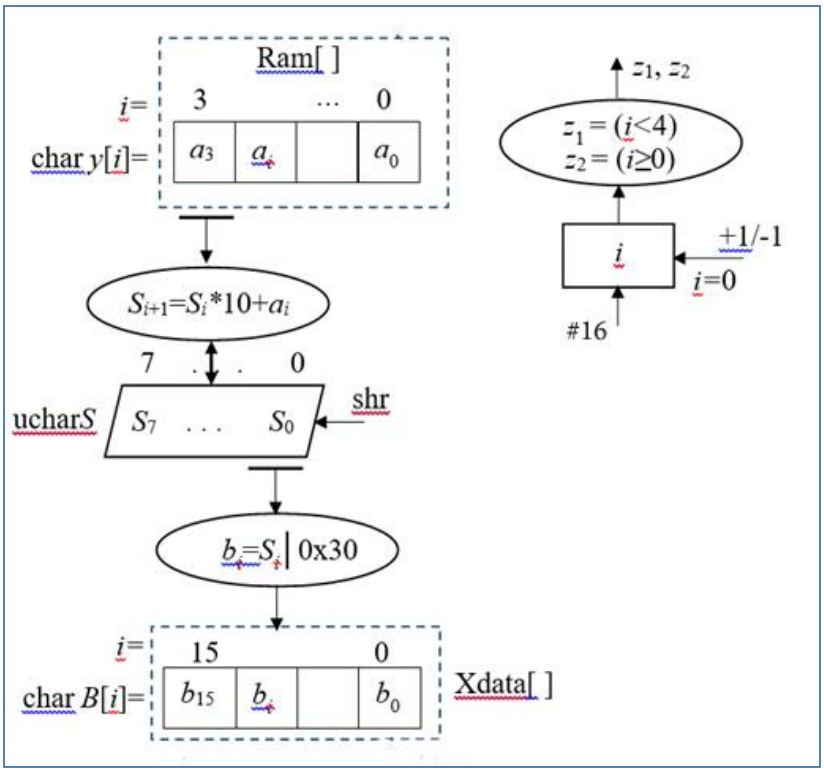


Рис.2.5. Алгоритмическая схема преобразования текста

Интуитивный алгоритм ввода.

1)Исходные данные- строка размещается в выделенном блоке памяти типа Ram . Формат данных – текстовая строка в стандартном 2/10 4-хразрядном двоичном коде. В процессе преобразования потребуется сдвиговый регистр двоичного кода . Результат преобразования - строка в памяти Xdata. Требуется также счетчик разрядов форматов.

2)Индексация (*i*) определяет для целых чисел порядок доступа к разрядам. Предполагается BigEnd порядок доступа по индексу к байтам массива *y*[i]/i=0,1,..7

3) LittleEnd к разрядам формата двоичного кода S[i]/i=7,6,..0 и байтам массива *B*[i]/i=15,14,..0.

В определяемом программой формате текста учитывается признак конца строки *B*[0]=0.

Логика алгоритма здесь очевидна и возможна запись программы и отладка в алгоритмических языках, минуя блок-схему.

Для отладки алгоритма управления алгоритмической схемой

1) Программа для исполнения в С++.

```
char dy[5] = "0125";                  //массив Code
unsigned char S;                      //переменная в Data
char xB[17];                          //массив Xdata
main() {   printf(" %s", dy);
             S = 0;
           for(i = 0; i<=4); i++)              //Si+1 = Si*10 +ai
              {S = S*10 + ( dy[i]&0x0f);}
           for(i = 16;  i>=0); i--)             //bi=Si | 0x30
              {xB[i] = (S&1) | 0x30; S = S>>1;}
               xB[16] = 0;
               printf(" %s", xB);
             while(1);}}
```

В программе на С++ используется прямой доступ к данным по идентификатору, адресность подразумевается.

2) Программа в С51

```
#include <reg51.h>

 char code  dy[5] = "0125";                 //массив Code
unsigned char S;                      //переменная в Data
char  xdata xB[17];                          //массив Xdata
main() {
             S = 0;
           for(i = 0; i<=4); i++)              //Si+1 = Si*10 +ai
              {S = S*10 + ( dy[i]&0x0f);}
           for(i = 16;  i>=0); i--)             //bi=Si | 0x30
              {xB[i] = (S&1) | 0x30; S = S>>1;}
               xB[16] = 0;
             while(1);}}
```

Исходные данные и результат контролируется в Симуляторе в окнах Watch и Memory. Измерить время исполнения и привести объем требуемой памяти Code.

3) Алгоритмическую схему можно рассматривать как **функциональную** схему исполнителя с прямым доступом к элементам памяти и функциональным блокам.

Функциональная микропрограмма в С++, условно разделяемая на микрокоманды (m1, m2, ...).

2

```
char dy[ ] ="0125";
unsigned char S;
char xB[9];
main() {
        m1:   {i = 0; S = 0;}
        m2:  {S = S*10 + ( y[i++]&0x0f);}
        m3:   {if(i< 4) goto m2;}
        m4:   {i = 16; B[i] = 0;}
        m5:   {xB[i--] = (S&1) | 0x30; S = S>>1;}
        m6:   {if (i >= 0) goto m5;}}
      }
```

4) **Программирование в С51 с указателем**.

Техника работы с указателем эквивалентна косвенному доступу к данным по адресу, определяемому символической ссылкой.

```
      #include  <reg51.h>
      unsigned char x,i; //переменная в Data
char  code * y="125";  //указатель на текстовую константу, имя переменной
      обозначает адрес
char  xdata  * yy;      //указатель текстовой  переменной
      main()
        {      for (i=0; i<3; i++)  x=x*10+(*y++&0x0f);
                 for (i=7;i>=0; i--)
                        {  *yy++= (x&0x01) ? '1' : '0';
                           x=x>>1;
                        }
                while(1);  //динамический останов
        }
```

Задания

Разработать алгоритмическую схему преобразования символических строк и микропрограммы в С++, С51 с прямым доступом и указателем. Измерить время исполнения и привести объем требуемой памяти .

1. Упорядочить текст лексикографически, в порядке возрастания ASCII-кода
   "This programmator" → " aaghimmootTrrs"

2. Вставить пробелы после символа "r"
   "This programmator" → "r" → "This pr ogr ammator"

3. Заменить прописную букву "x" на заглавную в тексте
   "This programmator" →"a" → "This progrAmmAtor"

4. Символьное (в ASCII) преобразование двоичного числа в шестнадцатеричное

“01001001110” → “0x24e”

5.Преобразовать число с естественной запятой в полулогарифмическую форму в десятичной системе с учетом знака порядка и знака мантиссы

“-25,023” → “e+2 - 0.25023”

6.Символьное (в ASCII) преобразование десятичного числа в шестнадцатеричное

“ 590 ” → “0x24e”

7. Десятичное сложение (вычитание) в неупакованных форматах, положение запятой фиксировано

“256,54” +” 75, 56” =” 332,10”

8. Сформировать сдачу минимальным количеством монет достоинством **50, 10, 5, 1** копеек и проверить обратным преобразованием

“132” → “2 ,3,0, 2”

9. Преобразовать символьный двоичный код в символьный Манчестерский код и восстановить исходный двоичный

“01011010” → 00 11 00 11 11 00 11 00 (+)
10 10 10 10 10 10 10 10 синхросигнал
→ “10 01 10 01 01 10 01 10 “ Манчестерский код

Восстановление символьного двоичного кода из Манчестерского

“1001100101100110” Манчестерский код
→ “ 0 1 0 1 1 0 1 0” двоичный код

10. Шифрование и дешифрование Гронсфельда

таблица символов {a,b,c,d,e,f, …}
нумерация 0 1 2 3 4 5 6
ключ {3,1,2,0,6, …}
“cadda” ←→ “ cdaad”

11. Преобразование двоичной импульсной последовательности в 3-значный код, перепад 0/1 обозначается 1, 1/0 обозначается 2, отсутствие перепада – 0 и обратно

“0 1 0 0 0 1 0 1 1“ ←→ “2 1 0 0 2 1 2 0”

12. Байты данных разбиваются на 2 тетрады, каждая тетрада заменяется HEX-цифрой и преобразуется в ASCII-код, подсчет контрольной суммы байтов по модулю 0x100 в конце строки HEX-кода

A0, B1, 0C, 1D → HEX-код строки “ A 0 B 1 0 C 1 D 8 A”

13.Обратное преобразование HEX-кода в строку байтов данных и проверить контрольную сумму - последний байт в строке

A0, B1, 0C, 1D → “ A 0 B 1 0 C 1 D 8 A” HEX-код строки

14. Регистр граничного сканирования n-контактов в JTAG-интерфейсе имеет длину 3n бит. Выбрать 3-хбитную j-ую ячейку в регистре. Нумерация битов регистра справа налево 3n,…2,1,0

“1 0 1 1 0 0 1 1 0 1 0 1 1 1 1 0 1 0 1” → “110”

o c i i c o

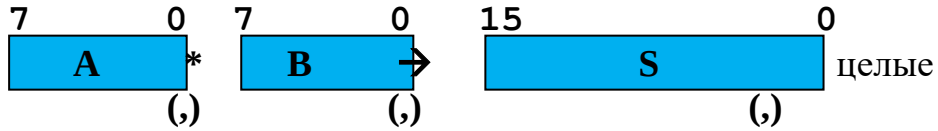
3.3. Машинная арифметика – умножение и деление.

В разделе рассматриваются **алгоритмизация выполнения рекурсивных** алгоритмов арифметических операций с числами в форматах с фиксированной запятой**.**

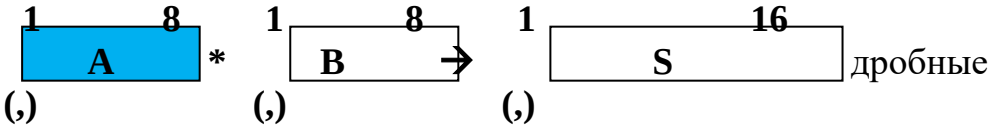
Исполнитель представлен **функциональной схемой**, которая определяет форматы используемых **регистров, организацию доступа к памяти и управление (микрооперации) функциональными преобразованиями .**

3.3.1. **Умножение**

Формулы вычисления произведения в форматах с фиксированной точкой без знака



$$S=A*B=A(\ b_{n=1}2^7 + b_{n-2}2^6 \ldots +b_1 2^1+b_0 2^0)$$



$$S=A*B= A*(b_1 2^{-1} + b_2 2^{-2} \ldots +b_{n-1}2^{-7}+b_n 2^{-8}\ )$$

Алгоритмы пересчета можно представить в логике исполнения рекурсивных функций

А) **С общим членом ряда**

Возможны четыре варианта

1) целые числа (суммирование со стороны младших разрядов)
   $S_{i+1}= S_i +Ab_i 2^i = S_i +(A2^i)\,bi = S_i +(R_i)\,bi\ , R_i=R_{i-1}*2,\ R_0=A,\ S0=0,\ i=0,..7$
2) целые числа (суммирование со стороны старших разрядов)
   $S_{i+1}= S_i +Ab_i 2^i = S_i +(A2^i)\,bi = S_i +(R_i)\,bi\ , R_i=R_{i-1}/2,\ R_0=A2^7,\ S0=0,\ i=7,..0$
3) дробные числа(суммирование со стороны старших разрядов)
   $S_{i+1}= S_i +Ab_i 2^{-i} = S_i +(A2^{-i})\ bi = S_i +(R_i)\ bi\ ,\ R_{i+1}=R_i*2^{-1}, R_1=A,\ S0=0,\ i=1,..8$

4) дробные числа**(**суммирование со стороны младших разрядов)

$S_{i+1}= S_i + Ab_i2^{-i} = S_i + (A2^{-i}) bi = S_i + (R_i) bi$, $R_{i+1}=R_i*2$, $R_1=A^{-7}$, $S0=0$, $i=8,..1$

B) Схема Горнера

Возможны два варианта

5) целые числа $S_{i+1}= 2S_i + Ab_i$ $S0=0, i=7,..0$

6) дробные числа $S_{i+1}=2^{-1}(S_i + Ab_i)$ $S0=0, i=1,..8$

Формулы применимы к дробным или целым, если использовать масштабирование множителя.

Умножение двоичных дробных чисел в С51 (вариант 2)

Вычисление с общим членом ряда со стороны старших разрядов множителя

$$S=A*B=A*(B=0.b_1b_2..b_7) = A*(b_12^{-1} + b_22^{-2} .. + b_62^{-6}+b_n2^{-7})=$$

$$= Ab_12^{-1} + Ab_22^{-2}.. + Ab_62^{-6} + Ab_72^{-7} = \sum Ab_i2^{-i} = \sum R_ib_i \rightarrow$$

$$S_{i+1}= S_i + (R_i2^{-1}) , R_i = R_{i-1}*2^{-1} \quad i=1..8; \quad R_0=A$$

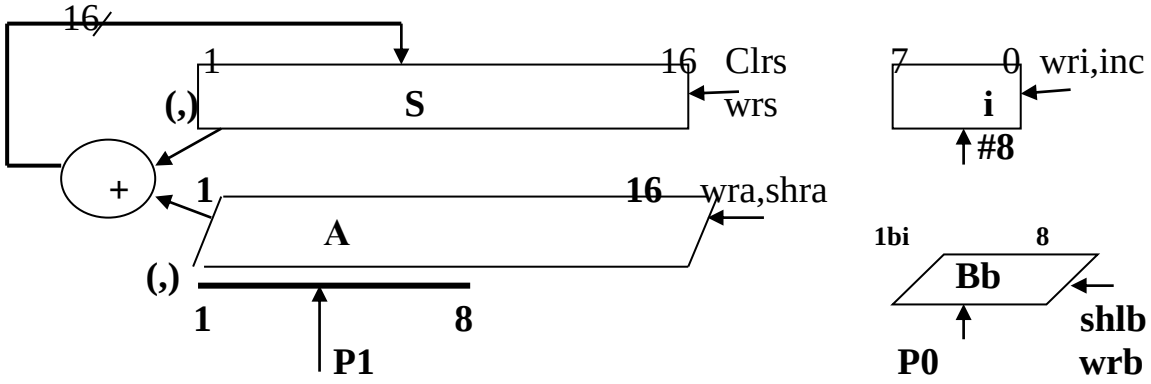


Рис.3.2.Функциональная схема умножения с общим членом ряда для дробных чисел

Требуемые ресурсы памяти – 16-разрядные регистры S и A. Регистр A со сдвигом вправо, 16-разрядный сумматор, 8-разрядный регистр B со сдвигом влево, счетчик циклов.

Если **bi=1**, то **S=S+ Ai*bi** - разрешение записи (**wrs**). Регистр A=**Ai>>1** сдвигается вправо (управление **shra)**

В регистре **B** множитель сдвигается влево (**shlb**), в старшем разряде контролируется текущее значение бита **bi**.

Функциональная микропрограмма в С++

```
 unsigned int  S,A;
 unsigned char  i,B;
main()
 {
m1:  { A=P1<<8; B=P0; S=0; i=8;  }//ввод
```


2

```
m2:  {if(B&0x80)  S=(S+(A>>1));  }
m3:  {R>>=1; B<<=1;  i++; }
m4:  { if (i<8) goto m2;}
     while(1);  //динамический останов
}
```

t=0,21 мс – время выполнения операции. Объем программы Code=78 байт.

1) **Умножение целых чисел по схеме Горнера** со стороны старших разрядов множителя **(вариант 5)**

$$S=A*B=A(\ b_{n=1}2^7 + b_{n-2}2^6 .. + b_1 2^1 + b_0 2^0)=$$

$$= A\ b_{n=1}2^7 + Ab_{n-2}2^6 .. + Ab_1 2^1 + Ab_0 2^0=$$

$$=((\ ..(\ 0+Ab_{n=1})2 + A\ b_{n-2})2 + .. + Ab_1)2 + Ab_0 \rightarrow$$

$$\rightarrow\ S_{i+1}= 2S_i + Ab_i \qquad S_0=0,\ i=7,..0$$

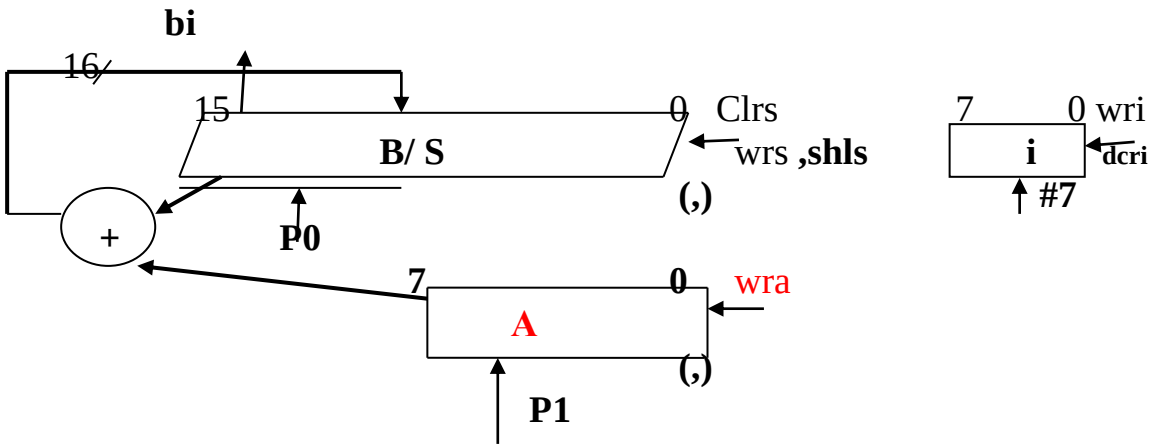


Рис. 3.3. Схема умножения дробных чисел по схеме Горнера в С51

Множитель **B** размещается в регистре **B**, при сдвиге регистра влево в старшем разряде контролируется текущее значение бита **bi**.

Суммирование выполняется в младших восьми разрядах регистра S частичных произведений . Микропрограмма умножения в С51, ввод и вывод выполняются через порты P0,P1,P2,P3.

```
#include <reg51.h>
char  i;
unsigned  int S;
unsigned char Bb,A;
main(){
m1: { A=P1;  S=P0<<8; i=-8; }
 m2: { if(~(S&0x8000==0)) {S=(S<<1); i++; goto m5;}  }
 m3:  { i++; S<<=1; }
 m4:  { i++; S+=A; }
 m5:  if (i!=0)  goto m2;  }
```

2.3.2. Деление.

Если рассматривать деление чисел B=S/A как обратную операцию для умножения S=A*B, то для рассмотренных итерационных формул умножения,

где произведение формируется со стороны младших разрядов, в делении цифры частного определяются всегда со старших разрядов.

Причем для операции деления дробных чисел, очевидно, делимое меньше делителя. Если условие не выполняется, то формируется признак переполнения и деление завершается. Также можно контролировать на очередном шаге равный нулю остаток и завершать деление простыми сдвигами.

Практически приемлемо только использование шестой формулы умножения при алгоритмизации деления.

Для **схемы Горнера (6)**

$S_{i+1}=2^{-1}(Si + Ab_i)$ **i=1,..8,** B-дробное

Если **$S_i= (2S_{i+1}-A) >=0$, то $b_i=1$.**

Если $S=S_i= (2S_{i+1}-A) <0$, то **$b_i=0$** и на следующем шаге предполагается восстановление положительного остатка и затем вычитание для получения следующего остатка $2(S+A)-A=2S+2A-A=2S+A$

Следовательно, при отрицательном остатке суммирование множимого и отрицательного остатка на следующем шаге эквивалентно восстановлению остатка и повторному вычитанию .

Если произведение вычислено с общим членом ряда (формула 1),

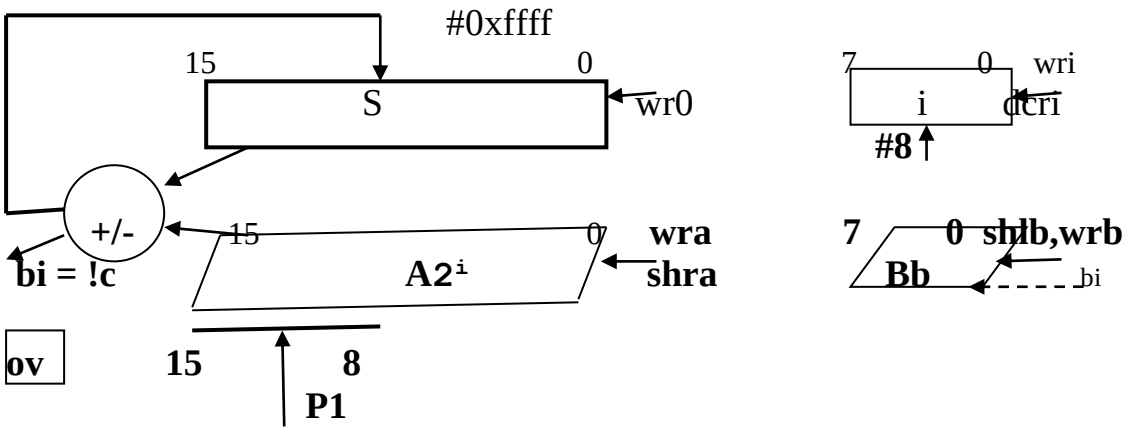
$S_{i+1}= Si + Ab_i2^i$, i=7,..0, B-целое

то обратная операция деления выполняется по формулам

Если **$S_i= (S_{i+1} - A2^7)>=0$, то $b_i=1$.**

Если $S=S_i= (S_{i+1} - A2^7) <0$, то **$b_i=0$** и на следующем шаге предполагается восстановление положительного остатка и затем вычитание для получения следующего остатка $(S+A2^7)-A2^6=S + A2^6$

Проблема переполнения – стандартное решение сводится к заверешению преобразования прерыванием и формированием признака неопределенного результата. Предлагается ограничиться признаком и продолжать вычисления, полагая результат равным максимальному значению 2^8-1 в формате unsigned char. В частности, при делении на нуль это значение формируется естественно при вычислении 8-разрядного частного.



3

Рис.3.4. Схема деления целых.

```
#include <reg51.h>
int S;
unsigned int A=P1<<8;
unsigned char Bb;
unsigned char i=0x08;
main( ){
  m1: { Bb=0;i=8; ov=0;}
  m2: {if(S>=A) {ov=1; S=0xFFFF;}
  m3: {if  (S>=0)  {S=S-A; Bb=(Bb<<1)|1; }}
  m4:  {if  (S<0)  S=S+A;  }
  m5:  {i=i-1; A>>=1; goto m2;}
  wihile(1); }
```

Задания к разделу.

Построить рекурсивные формулы и структурные схемы вычисления. Для схем разработать совмещенные программы и микропрограммы умножения и деления , ввод при тестировании с портов в С51

варианты	умножение
1	1
2	2
3	3
4	4
5	5
6	6
7	1
8	2
9	3
10	4
11	5
12	6

3.5. Вычисления функций

3.5.1. Числа с плавающей точкой

FP- машинный формат позволяет использовать полулогарифмическую запись числа при вычислениях и , следовательно, возможность обработки чисел на ЭВМ в широком диапазоне с автоматическим изменением масштаба, с постоянной относительной погрешностью .

В языке **Си** приняты форматы

float x= 12345,67 число с плавающей точкой в стандарте IEEE 754 с 24-разрядной мантиссой и 32 –разрядным форматом.

double x= 12345,6789 число с плавающей точкой в 64-разрядном формате с 48-разрядной мантиссой.

К числам с плавающей точкой в Си применимы операции **(+,-,*,/)** и функции стандартной библиотеки **math.h**

Формат с плавающей точкой и соответствующую арифметику называют научной, здесь диапазон практически не ограничен и имеет фиксированную относительную погрешность.

1)Используя функцию из библиотеки **math.h** языка C51, вычислить значения sin(x) в диапазоне аргумента 0-360º (2π радиан). При компиляции в Кейл записать параметры программы – объем требуемой памяти данных и объем программы.

2)В Логическом Анализаторе измерить среднее время вычисления функции.

Схема вывода значений функции цифро-аналоговое графическое преобразование выполняет Анализатор. В окне Анализатора как на экране цифрового осциллографа могут быть измерены временные параметры графика функции и абсолютные значения в масштабе. Для чисел в формате FP устанавливается по умолчанию масштаб 5.0 volt или выбирается в меню **Setup.** Целые значения отображаются в масштабе формата – для **char** выбирается **255.**

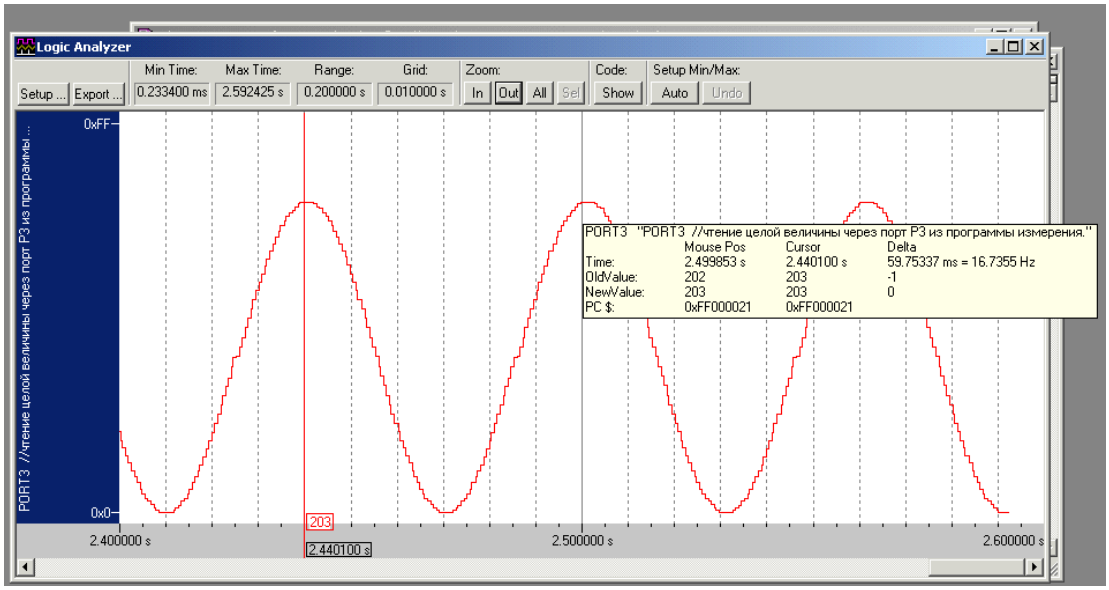
Виртуальное время вычислений контролируется Симулятором и определяется заданной при настройке частотой работы компьютера.

В опциях **Project.options.Target** частоту MCU выбираем **12.0 Мгц**

Программа вычисления функции **sinx в C51**

```
#include <reg51.h>
#include <math.h>
float x,y;
main()
{  while(1)
      for(x=0; x<6.28 ;x+=0.0628)
                  y= sin(x) ;  }
```


3



3.5. График функции в окне Анализатора.

Объем программы – 1.7 Кбайт, среднее время вычисления одного значения 3.7 мс.

3.5.2. Вычисление функции в целых числах и выбор масштабов.

Вычисления с фиксированной точкой позволяют существенно сократить время вычислений и объем программ, если операнды имеют ограниченную область значений (например, только дробные), в PLC основной машинный формат целый .

Функции в задании представлены **разложением в ряд Тейлора**

(3.8) **$\sin x \sim x - x^3/3! + x^5/5! - x^7/7! +$ при всех $x<1$**

Вычисления рядов выполняются **по схеме Горнера[1]** или с **общим членом ряда**.

$$\sin x \sim x - x^3/3! + x^5/5! - x^7/7! = x(1-x^2/6(1-x^2/20(1-x^2/42)) \rightarrow xS_i$$

$$S_1 = 1-x^2/a_0 * S_0 \quad , \; a_0 = 42, \; S_0 = 1$$

$$\rightarrow S_2 = 1-x^2/a_1 * S_1 \rightarrow \quad , \; a_1 = 20$$

……..

$$\rightarrow S_{i+1} = 1-x^2/a_i * S_i \; , \; S_0=1; \; i=0,1,$$

Выберем аргумент в диапазоне дробных чисел **0- 0.99** радиан и преобразуем **в целые** с масштабом **m**.

$$S_{i+1} = 1-x^2/a_i * S_i \quad \rightarrow \quad S_{i+1} = (m - (((x^2/m))/a_i * S_i)/m)$$

.

1) Масштаб m=100, программа в C51

```
typedef unsigned char uchar;
  uchar x,y,S,m;
 uchar Si(uchar ai)
 { return S=m-((y/ai*S)/100) ;}
 main()
 { m=100;
```

```
  while(1)
 for(x=0;x<m;x++)
  { y=(x*x)/100;S=m;
          S=Si(42);
          S=Si(20);
         S=Si(6);
       P2=(S*x)/100;}}
```

2)Функциональная микропрограмма в С51 , вычислений с масштабом 100

```
#include <reg51.h>
typedef unsigned char uchar;
   uchar x,y,S,z;
 Si(char ai )
 { m9: {z=y/ai; }
   m10: {z=(z*S)/100; }
   m11: {S=100-z; }}

 main()
 {
 while(1)
    {m1:  {x=0; S=100; }
     m2:  {y=(x*x)/100; }
     m3:  { Si(42);}
     m4:  { Si(20);}
     m5:  { Si(6);}
     m6:  {S=(S*x)/100; }
     m7:   {  x++;  P2=S; }
    m8:     {if(x<100) goto m2;
           } }
```

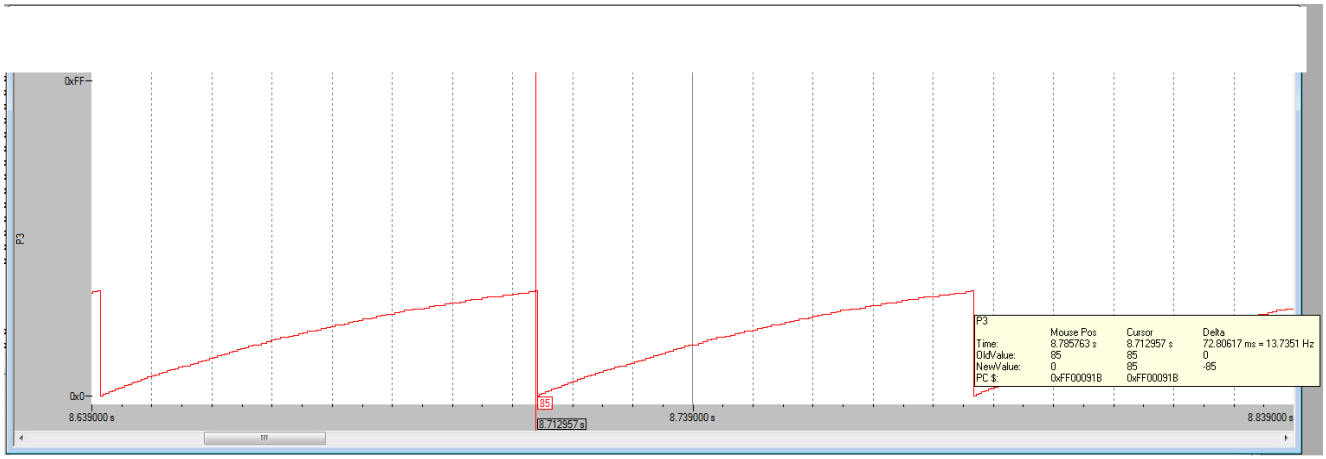


Рис.3.6. График значений sin(x) с приближением в масштабе m=100 в С51
x – значение аргумента в диапазоне [0 – 100], от 0 до 1.0 радиан.
Объем программы 320 байт, среднее время 0.72мс

3) Вычисления с масштабом $m=2^8$

Программа в С51.

```
#include <reg51.h>
typedef unsigned char uchar;
   uchar x,y,S;
 uchar Si(uchar ai)
 { return S=0xFF-((y/ai*S)>>8) ;}
 main()
 {
 while(1)

  for(x=0;x<m;x++)
   { y=(x*x)>>8;
            S=Si(42);
            S=Si(20);
           S=Si(6);
        P2=(S*x)>>8;}}
```

Задания.

Выполнить в С51 вычисления с плавающей точкой и с фиксированной точкой по схеме Горнера и с общим членом ряда с масштабами m=100 и $m=2^8$ в С51. Привести графики значений и погрешности, параметры программ.

1. $(1+x)/((1-x)2) \sim 1/2 + x + x^2 + x^3 +$

2. $1/(1+x) \sim 1 - x + x^2 - x^3 +$

3. $x^{0.5} \sim x/2 - x^2/(2*4) + 1*3*x^3/(2*4*6) - 1*3*5*x^4/(2*4*6*9)$

4. $a^x \sim 1 + (\ln a)*x + (\ln a)^2 x^2/2! + (\ln a)^3 x^3/3! +$
$a=1/2$

5. $\cos(x) \sim 1 - x^2/2! + x^4/4! - x^6/6! +$

6. $tgx \sim x + x^3/3 + 2x^5/15 + 17x^7/315 + 62x^9/2835$

7. $ctgx \sim 1/x - (x/3 + x^3/45 + 2x^5/945 + 2x^7/4725 + \ldots)$

8. $\ln(1+x) \sim x - x^2/2 + x^3/3 - x^4/4 + x^5/5 +$

9. $\arcsin(x) \sim x + x^3/(2*3) + 1*3*x^5/(2*4*5) + 1*3*5x^7/(2*4*6*7) +$

10. $arctg(x) \sim x - x^3/3 + x^5/5 - x^7/7 +$

11. $(1-x)^{0.5} \sim 1 - x/2 - x^2/(2*4) - 1*3*x^3/(2*4*6) - 1*3*5*x^4/(2*4*6*9)$

12. $(1+x)^{1/3} \sim 1 + x/3 - 2x^2/(3*6) + 2*5*x^3/(3*6*9)$

13. $(1+x)^{3/2} \sim 1 + 3x/2 + 3x^2/(2*4) - 3x^3/(2*4*6) + 9x^4/(2*4*6*8)$

15. $arsh(x) \sim x - x^3/(2*3) + 1*3*x^5/(2*4*5) - 1*3*5x^7/(2*4*6*7)$

16. $ch(x) \sim 1 + x^2/2! + x^4/4! + x^6/6! +$

17. $sh(x) \sim x/1 + x^3/3! + x^5/5! + x^7/7! +$

18. $Si(x) \sim x - x^3/(3*3!) + x^5/(5*5!) - x^7/(7*7!) +$

19. $Ci(x) \sim 1 - x^2/(2*2!) + x^4/(4*4!) - x^6/(6*6!) +$

3.6. Вероятностная логика

В работе [9] предлагается арифметика вычисления вероятности, эквивалентная по смыслу логике, представленной логическими формулами. Значение истинности интерпретируется вероятностью в интервале [0..1]

Вероятностная ВФ функция P(f(x1,..xn)=1), обозначает вероятность истинности логической формулы.

Формула перехода ФПЗ к замещению допускает переход к ВФ замещением переменных на вероятности.

Интерпретация операций логики высказываний в вероятностной логике необходимо ограничены и должны совпадать по смыслу в дискретных состояниях переменных {0,1}, соответственно с вероятностями P(¬x=1) или P(x=1).

Эти состояния обозначают абсолютную “истинность” или “ложь”.

(1) Если **P(xi=1)=Ri** –вероятность прямого значения переменной xi и **P(¬xi=1)=(1-Ri)=Qi** –вероятность инверсного значения переменной ¬xi,

(2) **P{(xi&xj)=1}=Ri*Rj**

(3) P{(¬xi&¬xj)=1}=(1-Ri)(1-Rj)=QiQj

(4) P((xi v xj)=1)=P(¬(¬xi&¬xj)=1)= 1- QiQj

В этих определениях сохраняется смысл логических операций для дискретных значений **Ri,Rj є{0, 1}** и сохраняются таблицы истинности соответствующих логических операций

Очевидно, выполняются законы коммутативности и ассоциативности относительно конъюнкции.

Интерпретация некоторых законов алгебры логики

(5) P(¬¬xi=1)=1-(1-Ri)=Ri =1-Qi, двойное отрицание

3

(6) правила де Моргана

P(¬(xi v xj)=1)=1-(1- QiQj)=QiQj=P{(¬xi&¬xj)=1}

P{¬(xi & xj)=1}=1-P{(xi & xj)=1}=1- Ri*Rj=P((¬xi v ¬xj)=1)

Эти законы позволяют преобразовать любую логическую формулу в ВФ

- последовательно с использованием инверсий преобразуются дизъюнкции в конъюнкции с использованием законов де Моргана по правилам (1-6), затем последовательной заменой логических операций вероятностными формулами и упрощающими алгебраическими операциями

Булева функция может быть использована для описания структуры соединений блоков технической системы, технологического процесса, контактной схемы и др.

Пример

F= $x_1(x_2 \vee x_3 \vee \neg x_4) \vee x_5(x_6 \vee x_7 \neg x_8)$

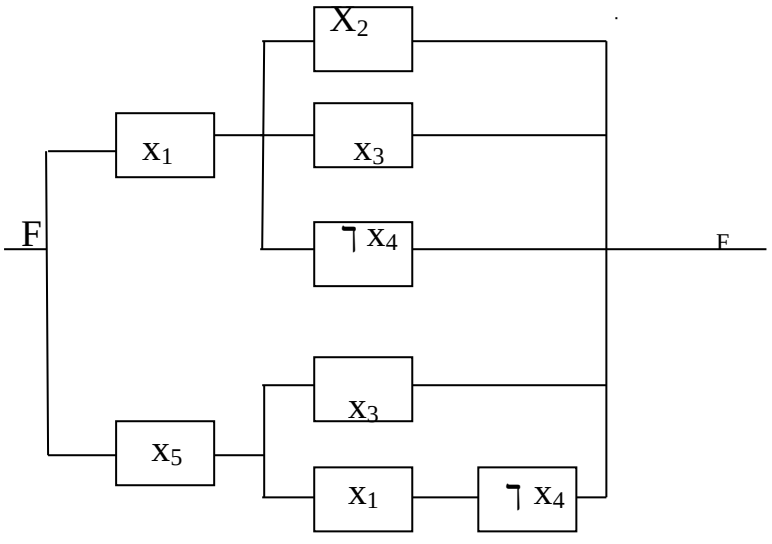


Рис.3.7. Структура объекта, определяемая логической функцией.

Переменные x_1 - x_5 интерпретируются как вероятности исправности блоков Ri, а инверсии как неисправности (1-Ri)=Qi,

Вероятность исправной работы системы вычисляется в области вероятностей с использованием вероятностной функции ФВ, которая может быть получена преобразованием логической формулы с использованием тождеств 1-6.

Пример преобразования логической функции

F= $x_1(x_2 \vee x_3 \vee \neg x_4) \vee x_5(x_6 \vee x_7 \neg x_8) = \neg\neg [x_1(x_2 \vee x_3 \vee \neg x_4) \vee x_5(x_6 \vee (x_7 \neg x_8))]$=

Последовательное исключение дизъюнкций с перемещением инверсий вниз по правилу де Моргана с заменой дизъюнкции на конъюнкцию

= $\neg [\neg(x_1(x_2 \vee x_3 \vee \neg x_4)) \& \neg (x_5(x_6 \vee (x_7 \neg x_8)))]$=

= $\neg [\neg(x_1 \neg\neg (x_2 \vee x_3 \vee \neg x_4)) \& \neg (x_5 \neg\neg (x_6 \vee (x_7 \neg x_8)))]$=

= $\neg [\neg(x_1 \neg (\neg x_2 \neg x_3 x_4)) \& \neg (x_5 \neg (\neg x_6 \neg (x_7 \neg x_8)))]$= формула ОПЗФ содержит только инверсии и конъюнкции

С учетом скобок замена переменных на вероятности и исключение инверсий получим ФВ-формулу

$S=1-[(1-R_1(1-Q_2Q_3R_4))\ (1-R_5(1-Q_6(1-R_7Q_8)))]$

Задание. 1)Для заданного варианта логической функции построить эквивалентную логику в виде структурной схемы

Построить таблицу истинности в С51 по формуле в портах.,

2)Выполнить преобразование булевой функции $z(x_1x_2x_3x_4)$ в формулу вероятностей $S(Q_1Q_2R_3R_4)$.

Значения Q,R=False(0) и Q,R=True(1) кодируются при вычислениях в байтах и обозначают вероятности исправности или неисправности соответствующих блоков структуры. Значение S, равное 1 обозначает, что схема работоспособна, 0-схема неисправна. Тестировать формулу в С и С51 на наборах значений истинности .

Задания

1. $z=(y_1/x_1 \vee y_2x_2)(/y_1 \vee x_2)$
2. $z= (y_1 \vee /x_1)(y_2x_2 \vee x_1)$
3. $z=/x1(x2 \vee /x3) \vee x1x4$
4. $z=(x_1 \vee /x_2x_3)(/x_2 \vee x_4)$
5. $z= /y_1 \vee /y_2(y_1x_1 \vee /x_2)$
6. $z=(x_1 \vee /x_3x_4)(/x_1 \vee x_2)$
7. $z=/y_1x_2 \vee y_2(/x_1 \vee /x_2)$
8. $z=(/x_1 \vee x_2)(x_1x_3 \vee /x_4)$
9. $z=(x_1y_1 \vee /x_2y_2)(/x_2 \vee y_1)$
10. $z=(/x_1 \vee y_1)\ (x_2y_2 \vee /y_1)$
11. $z=y_1(/y_2 \vee /y_3) \vee /y_1y_4$
12. $z=(y_1 \vee /y_2y_3)(/y_2 \vee y_4)$

3

13. $z=/y_1\ y_2\ v\ /x_1x_2(y_1\ v\ /y_2)$
14. $z=x_1y_1\ v\ /x_2(/y_2\ v\ /x_1)$
15. $z=(x_1\ v\ /x_2\ v\ /x_3x_4)(/x_1\ v\ /x_4)$

3.7.Работа с таблицам данных.

Во многих задачах численные данные представлены при вводе и отражают естественно исходные знания в памяти ЭВМ, например, в вид таблиц и численных массивов.

Логика в алгоритме решения задачи требует интуитивных размышлений и не очевидна. Таким образом, в алгоритмизации приемлема только текстуальная форма записи с последующим уточнением в формальной записи в виде блок-схемы и алгоритмического языка.

Пример такой задачи с расписанием поездов для ограниченного транспортного узла.

Требуется выбрать минимальный по времени маршрут между двумя поселениями Сыров(2) и Лесное(3), соседних областей. Известно расписание движения для шести поселков Ложев(1), Сыров(2), Лесное(3), Березов(4), Ольсов(5), Опуг(6).

Отправление(о)	Прибытие (п)	Toi (час)	Tjп (час)	Tji
Ложев(о1),	Сыров(2п),	8.20	10.05	T12=1.45
Ложев(о1),	Лесное(3п)	9.10	12.35	T13= 3.25
Сыров(о2),	Березов(4п),	6.05	07.05	T24= 1.0
Сыров(о2),	Ложев(1п),	6.25	08.15	T21= 1.50
Березов(о4),	Ольсов(5п),	7.15	08.40	T45= 1.25
Ольсов(о5),	Лесное(3п)	8.25	09.45	T53= 1.20
Сыров(о2),	Лесное(3п)	5.30	09.05	T23=3.35
Опуг(о6).	Лесное(3п)	7.0	9.20	T63=2.20
Сыров(о2),	Ольсов(5п),	7.0	8.20	T25=1.20
Сыров(о2),	Опуг(6п).	6.10	7.10	T26= 1.0

Формализация схемы путей между городами в виде ориентированного графа Рис.

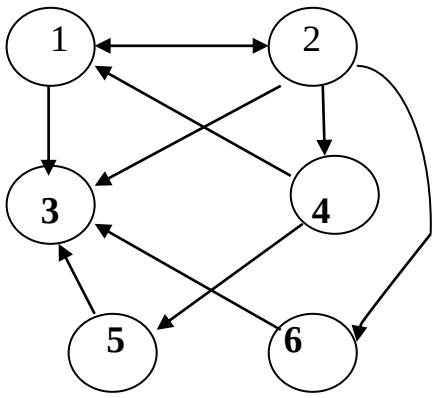


Рис.3.8. Граф сети марщрутов

Временные интервалы определены задержками То(время отправления), Тж(время ожидания), Тп(время прибытия).
Пусть время прибытия в исходный пункт Тн2- предполагается равно То2.
Время ожидания Тж2=То2 -Тн2 =0
Выбор направления от Сырова(о2) на Лесное(3п) – время в пути

2→3: T23= 3.35

Найти все доступные пути

2→1→3: T21 + T13 = 1.50 + 3.25 = 5.15

2→4→5→3: T23= T24 + T45 + T53 = 3.35 + 1.25 + 1.20 = 6.20

2→6→3:T23=T26+T63=1.0+2.20=3.20

Описание шагов и логики алгоритма в содержательной вербальной записи:

1. Описание знаний таблицами.
   Необходимая информация может быть представлена в более доступной в алгоритмическом мышлении в форме двумерного массива S[j,i]

j \ i	1	2	3	4	5	6
1		T12=1.45	T13= 3.25			
2	T21= 1.50		T23=3.35	T24= 1.0		T26= 1.0
3	T31= 3.25					
4	T41=2.0				T45= 1.25	
5			T53= 1.20			
6			T63=2.20	T64=1.7		

При вводе должны быть заданы переменные :
номер входного пунка j=2 и выходного i=3, продолжительность T=0 движения по трассе j→i, одномерный массив пунктов пути Q[0]=i,
2.Найти в столбце i=3 таблицы S[j,3] не равный нулю Tj3 , j=1,2,..6
Может завершиться поиск, если соответствующий элемент не найден.
Если (T13 ≠ 0), то (T13= 3.25, j=1) , время в пути T=T+ T13, сохранить номер j в Q[ ]={3,1},
3.Найти в столбце j= i=1 для j=1,2,..6 ближайший не равный нулю элемент (T21= 1.50, j=2), длина пути T= = T+ T21 =T+1.50=4.75, Q={3,1,2}

4

Если j ≠2 , то время в пути T= T+ T21, сохраняем найденный путь в массиве Q[i]={1,i=j} и переход к шагу 3- поиск продолжения пути (i=j,j=1) Если j =2 заданный пункт отправления, то время в пути T= T+ T21 и сохраняем найденный пункт в массиве Q[i]={3,1,2} и переходим к шагу 2- поиск нового пути (i=3,j=?). Здесь повторение поиска пути при (T13 ≠ 0). Предлагается сбрасывать его на шаге 2.

Представленную текстом интуитивно логику алгоритма выбора одного пути обозначим блок-схемой, в интуитивной формализации алгоритм уточняется и доопределяется поиском возможных других путей. Предлагается в последующей формализации на языке С++ уточнить логику поиска других путей и их оформление в массивах для выбора минимального.

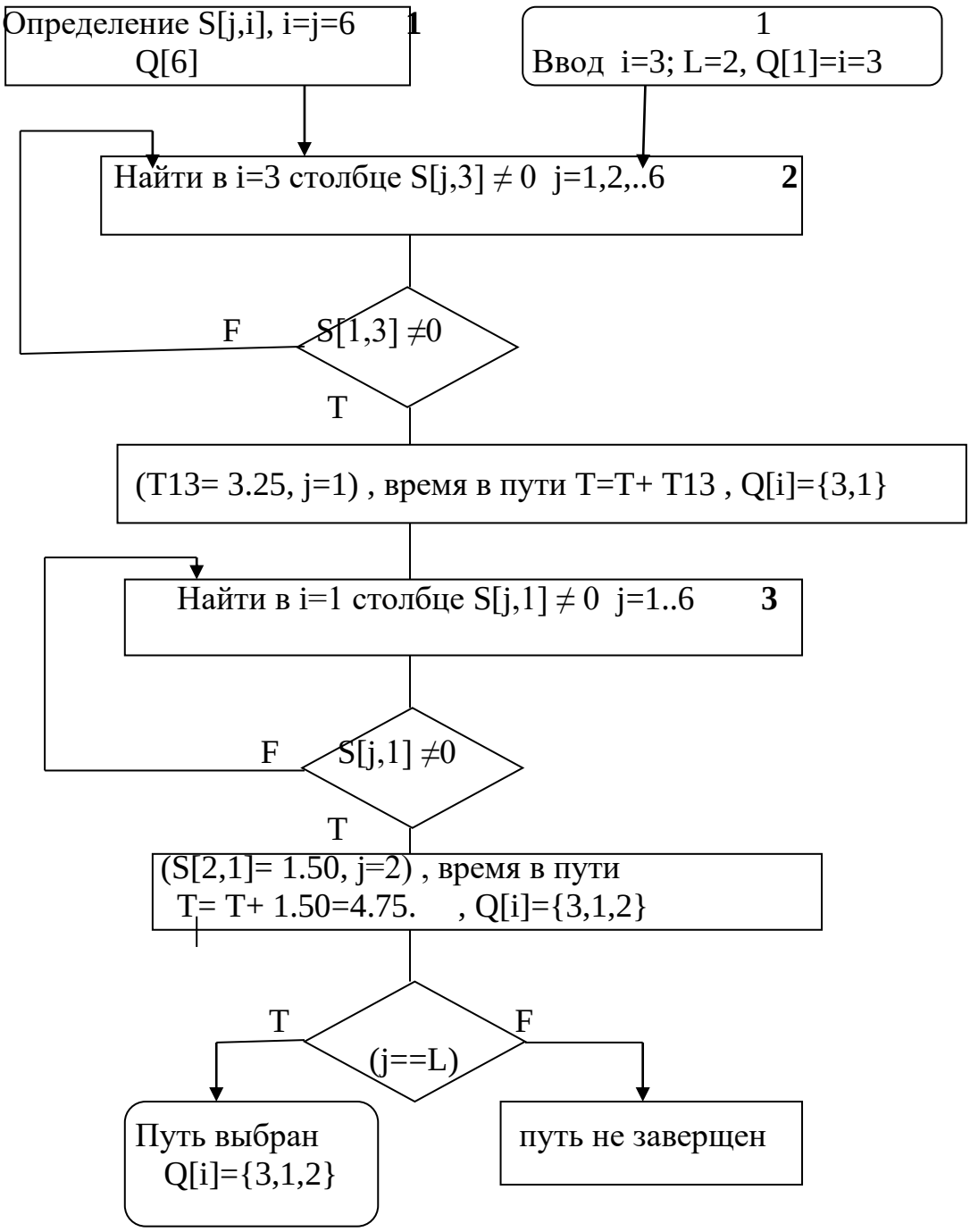


Рис. 3.9. Блок-схема выбора маршрута между пунктами

```
        #include <stdio.h>
 char Q[12];
 int R[6];
 xdata int S[6][6]={{0,145,325,0,0,0},     //j=1.., i=3
        {150,0,350,1.0,0,100},             //j=2,
                 {0,0,0,0,0,0},            //j=3
                 {0,0,0,0,125,0},          //j=4
                {0,0,120,0,0,0},           // j=5
                {0,0,220,0,0,0}};          //j=6

unsigned char j,i,k,L,F,N,p;
  int T,M;
  main(){
     k=0; j=1;  F=i=3; L=2;
m2:  M=S[j-1][i-1];
       if(M)   //M>0
          {T=M+T; Q[k++]=i;
                        if(i==F) S[j-1][i-1]=0;   //
              if(j==L) { Q[k++]=j;
                       S[j-1][i-1]=0;   goto m5;}
                                 else  { i=j;j=1;}
                                       }
                      else if(j==6)goto m6;
                           j++;    goto m2;

 m5:  {i=F;j=1;R[N]=T;N++;  goto m2;}

   m6: while(1);
       }
```

Предлагается решить задачи с использованием модели расписания в виде графа

4

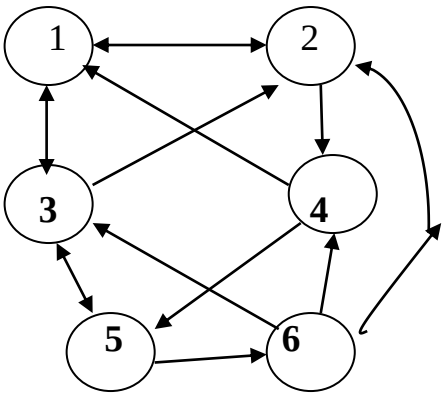


Рис. 3.10. Граф сети марщрутов

Варианты j\i	начало	конец
1	**1**	**2**
2	**2**	**1**
3	**2**	**3**
4	**3**	**2**
5	**1**	**4**
6	**4**	**1**
7	**2**	**5**
8	**5**	**2**
9	**2**	**6**
10	**6**	**1**
11	**1**	**6**

3.8. Управление вводом и выводом в ПЛК

ПЛК как система на кристалле интегрирует схемы микропроцессора, память, средства управления внешними интерфейсами для управления вводом/выводом, схемы контроля и управления системой в реальном времени.

К последним относятся:

- **таймеры,** которые позволяют организовать измерение временных интервалов и управление совмещенными алгоритмическими процессами в реальном времени;

- **подсистема прерываний** обеспечивает контроль асинхронных системных событий в реальном времени.

Интерфейсы ввода/вывода классифицируются как **параллельные** (обмен данными в форматах байта за одно обращение) и **последовательные** (побитовый обмен данными).

Выполнение алгоритма начинается с ввода данных, сохранением их в доступной памяти. Решение задачи завершается чтением результата из памяти

и выводом. Форматы входных данных могут отличаются от стандартных форматов, используемых в ЭВМ для хранения и преобразования. Таким образом, при вводе необходимо не только идентифицировать двоичный код в цифровых портах ввода, но и выполнить необходимые преобразования в общепринятые форматы данных для сохранения в памяти.

4.1. Алгоритмическое управление прерываниями

Для знакомства с организацией прерываний и алгоритмизацией управления рассматривается конкретная схема прерываний mcs51.

Схема регистрирует предполагаемые внутренние и внешние сигналы (запросы) прерываний, случайные по времени возникновения относительно исполнения “основной” программы.

Схема многоуровневой организации работы системы прерывания приведена на рис. 4.1.

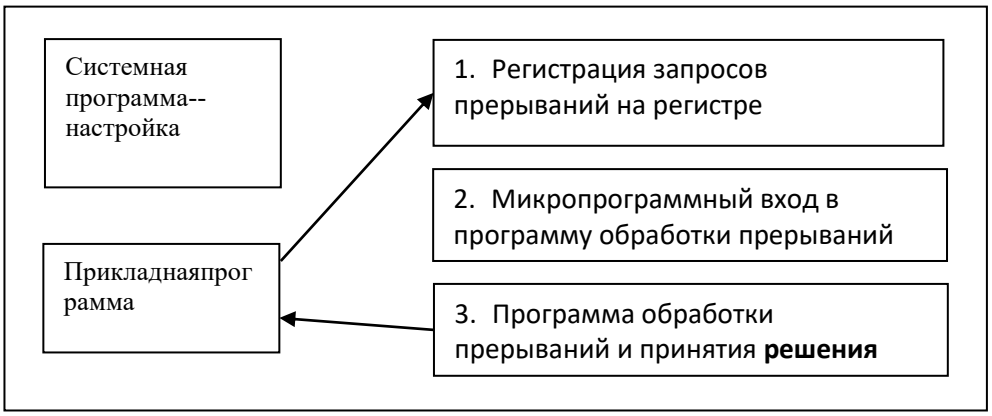


Рис.4.1.. Многоуровневая организация работы системы прерывания

Если разрешены конкретные запросы, на микропрограммном уровне, то выбирается наиболее приоритетный зарегистрированный запрос. Формируется **адрес-вектор** перехода к подпрограмме принятия решения с сохранением адреса возврата.

В подпрограмме обработки прерывания сохраняются состояния регистров **(контекст),** которые могут быть при этом изменены. Выполняется программа принятия решения, восстанавливается контекст и происходит возврат в прерванную программу.

1) Внутренние сигналы прерывания.

Сигналы, формируемые таймерами при завершении контрольных временных интервалов, сигналы завершения приема и/или передачи байта данных в последовательном интерфейсе. Признаки внутренних прерываний приведены в Таблице 2.1.

4

Таблица 2.1.

a	b	c	d
Tm0	TFO	ETO	1
Usart	TI ∨ RI	ES	4
Tm1	TF1	ET1	3

В столбцах таблицы представлены:

a) Источники запросов прерываний - таймеры **Tm0**, **Tm1**, последовательный интерфейс **Usart**.

b) Биты запросов прерываний таймеров **TF0, TF1**. Биты сбрасываются автоматически при входе в прерывание.

В модуле управления **Usart** формируются:

TI – запрос прерывания при завершении передачи байта;

RI – запрос прерывания при завершении приема байта.

c) Маски разрешения прерывания от таймеров **(ET0 = 1** разрешено).

ES = (TI ∨ RI) маска разрешения прерывания от **Usart**.

d) Номер (приоритет) прерывания – максимальный приоритет первый.

2) Внешние прерывания.

В mcs51 формируются сигналами, связанными с асинхронными процессами в периферии на **входных** контактах цифровых портов.

a) При вводе с клавиатуры – сигналы нажатия клавиш.

b) Контрольные точки временных диаграмм внешних сигналов.

c) Программные прерывания могут быть сформированы на входных контактах для контроля программно доступных событий, представленных временными диаграммами.

Сигналы внешних прерываний подключаются к входам порта P3. Признаки внешних прерываний приведены в Таблице.

Таблица 2.2.

a	b	c	d	e
p3.2 /**INT0**	IT0	IE0	EX0	0
p3.3 /**INT1**	IT1	IE1	EX1	2

В столбцах таблицы представлены:

a) Входные сигналы прерываний на контактах порта P3.2=INT0 и P3.3=INT1.
b) Тип прерываний IT0=1 – выбирается H/L фронт входного сигнала. При IT0=0 сигналом прерывания является низкий уровень.
c) Бит регистра запросов прерываний (IE0=1 запрос). Бит сбрасывается при входе в прерывание.
d) Маски разрешения прерывания (EX0=1 разрешено).
e) Номер (приоритет) прерывания.

Управление прерываниями:

1) Установить маску прерывания и бит разрешения прерываний **EA**=1.

2) Оформление функции принятия решения по прерыванию **INT0**

void klav(void) interrupt 0

{ }

Klav – имя подпрограммы обработки прерывания.

interrupt – служебное слово и **0** – номер(приоритет) прерывания **INT0** в таблице 2.2.

Компилятор в С51 формирует Стек для сохранения и восстановления контекста.

4.2. Алгоритмическое управление таймерами

Таймеры – счетчики реального времени. Единицу измерения времени определяет частота генератора синхросигналов ПЛК. В Keil для mcs51 частота F0 может быть выбрана в широком диапазоне, в частности, F0=12 Мгц.

В mcs51 и большинстве клонов **основной цикл** выполнения команды 12 тактов частоты F0. Команды выполняются за 1–3 цикла, что согласуется с форматами команд 1–3 байта и их последовательной выборкой из программной памяти. Один цикл является единицей времени, отсчитываемой таймером. Если установить частоту генератора 12 Мгц, то частота отсчета 1 Мгц и единица измерения времени 1 мкс.

Упрощенная схема таймера tmi (i = 0, 1) приведена на рис. 2.7.

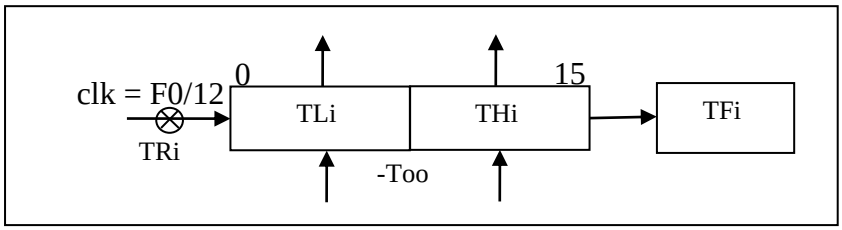


Рис.4.2. Схема таймера tmi

THi.TLi инкрементный 16-разрядный счетчик **частоты F0/12.**

Интервал измерения 16-битного таймера 2^{16} ~ 65000 мкс = 0.065 с.

TFi – бит переноса из старшего разряда таймера используется как сигнал запроса прерывания.

TRi – бит разрешения счета

Для **расширения диапазона измерения времени** используется программный счетчик, который инкрементируется по прерыванию TFi.

Для инкрементного счетчика при записи константы**(Too)** задается интервал задержки.

1)Измерение реального времени.

4

в интервале нескольких минут с точностью 1 мкс

Представляем интервал последовательным счетчиком из нескольких регистров

→Tm0→count→sec→min

Счетчик Tm0 гарантирует максимально доступную точность отсчета, реализуемую схемой.

Too = 50000; //1сек = 10^6мкс, count = $10^6/50000 \leq 20$.

Следующие счетчики – переменные инкрементируются программой по прерыванию при переполнении таймера

(count = count + 1), (sec = sec + 1, если (count = 20)),

(min = min + 1, если (sec = 60)).

```
#include <reg51.h>


int Too,Tm0;
char count,sec,min;
intt0() interrupt 1
{count++;
   if(count = = 20) {sec++; count = 0;}
   if(sec = = 60 ) {min++; sec = 0;}
  Tm0 = Too + TL0;       //переменная программная задержка
  TR0 = 0;                //останов таймера
  TH0= (Tm0 >>8); TL0= Tm0; //коррекция временной константы
  TR0=1;                  //разрешение таймера
  }
main()
{ TMOD = 1;             //режим 16-битового таймера Tm0
  ET0 = 1;              //маска прерывания при переполнении Tm0

  TR0 = 1;              //разрешение счетчика Tm0
  EA = 1;               //разрешение прерываний
-Too = -50000;
while(1);               //ожидание прерываний
```

График состояния счетчика **sec** реального времени приведен на рис. 2.8.

4

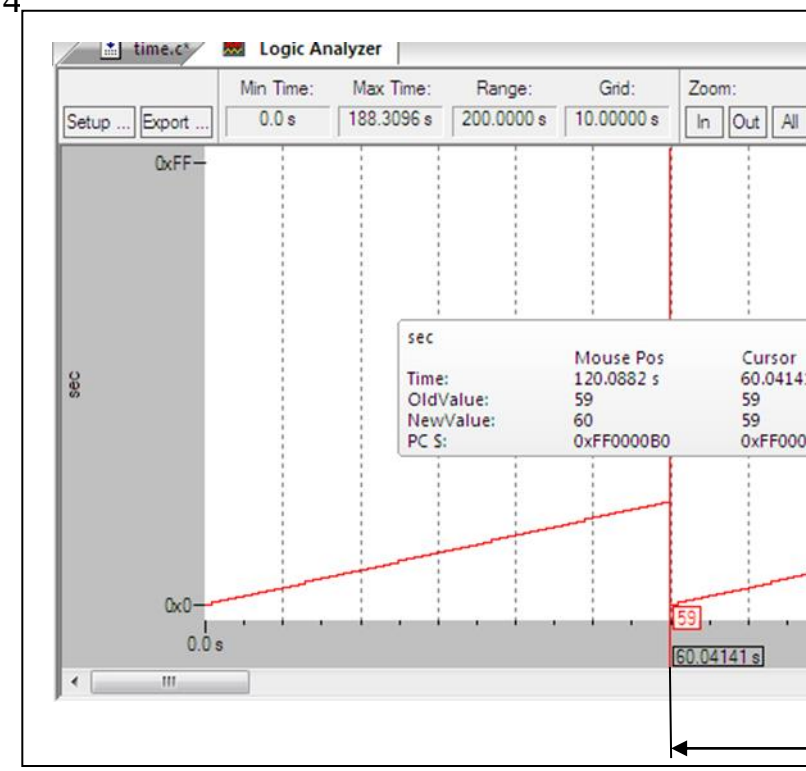


Рис.4.3. График состояния счетчика **sec** реального времени в Анализаторе

Длительность интервала одна минута, счетчик секунд sec = 60`, погрешность “реального” времени за минуту 0.05с.

Виртуальное время в симуляторе интерпретируется с частотой собственного генератора частоты ПЭВМ.

Предлагается измерить отношение реального времени к виртуальному и сократить погрешность измерения в “реальном” времени.

Эксперимент с ручным и программным контролем реального времени –

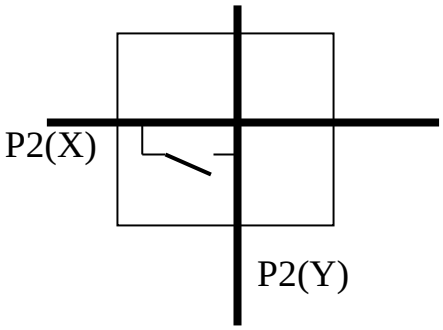
-

8. Ввод данных с клавиатуры.

Ручное управление , ввод и контроль состояния ЭВМ и объекта управления осуществляется с **клавиатуры**.

Принципы переключения – механические, сенсорные. В свою очередь сенсорные – конденсаторные и резистивные.

N переключателей для сокращения числа используемых цифровых линий при подключении к MCU объединяются в матрицу x*y.



4

Рис.4.5.Схема сканирования клавиатурыю

Линии X используются для последовательного сканирования строк матрицы при подключении к выходному порту P2[7.4], линии Y считываются при вводе с входного порта P2[3.0].

Сканирование в MCS51 выполняется сигналами низкого уровня построчно (унитарные инверсные коды сканирования Cx=P2[7..4]). Исходное состояние линий портов P1,P2,P3 в mcs51c опорным резистором по схеме с открытым коллектором высокое H. (рис.4.5.). При ручном несоглассованном вводе запрещено использовать активный положительный сигнал, если бит регистра порта установлен в 0 и контакт закорочен на землю.

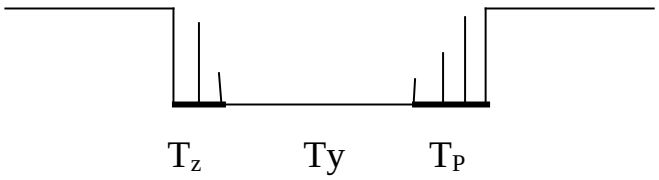
В данном случае при вводе соответствующие разряды порта P2 , по умолчанию, во время сканирования находятся в состоянии H.

При замыкании переключателя считывается инверсный унитарный код Cy=P2[3..0].

Если используются m=X+Y выводов для сканирования и распознавания замкнутых переключателей, то матрица позволяет коммутировать N=X*Y=X*(m-X) переключателей рис.4.6..

Клавишам присваиваются в определенном порядке символы алфавита, цифры и специальные функциональные обозначения их размещения (**раскладка**)

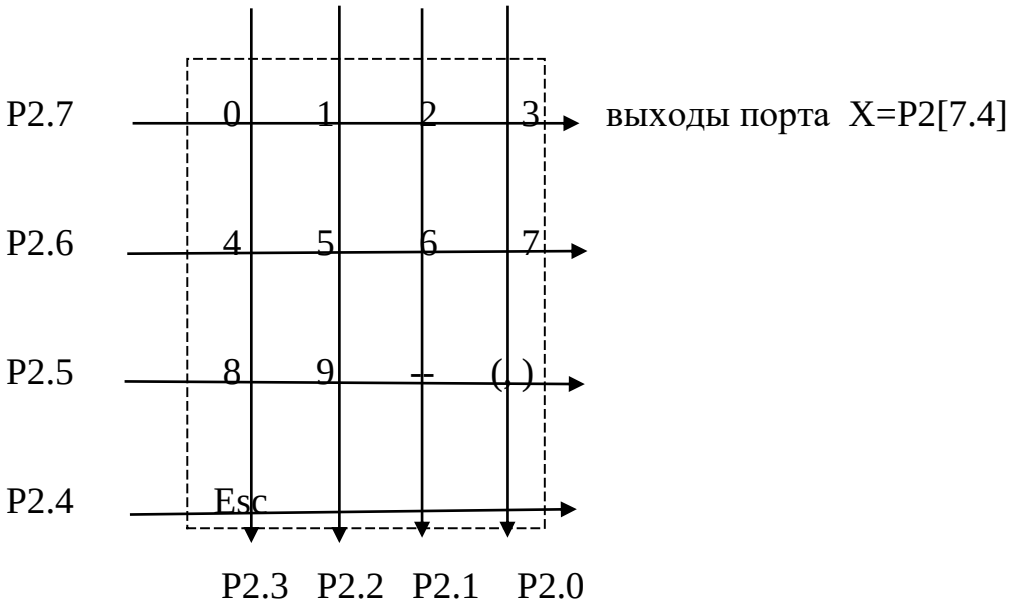
Временная диаграмма переключения



Tz~ Tp – время замыкания и размыкания с дребезгом около 0.04 с

Ty – время удержания в замкнутом состоянии 0.3 с

Раскладка клавиатуры и схема включения с портом P2 mcs51.



Входы порта Y=P2[3..0]

Рис. 4.6. Раскладка клавиатуры

Функциональные клавиши (-), (,), (esc)- конец ввода
Если замкнут контакт одной из клавиш, L-уровень поступает на входные контакты порта P2[3.0]. Считываемый код с порта P2[7.0] идентифицирует нажатую клавишу.

Например, при сканировании строки P2.7=0 и нажатой клавише ‘1’ считывается состояние порта P2 - **Код сканирования**.

```
        0 1 1 1  1 0 1 1
P2 [ 7       4   3       0 ]
   вывод  Cx   ввод  Cy
```

Алгоритм сканирования

1. Формирование кода сканирования Cx
2. Ввод и контроль Cy - контроль нажатия клавиши
3. При нажатии клавиши идентифицировать замкнутый переключатель и выполнить соответствующую функцию
4. Задержка дребезга Tz
5. Ожидание размыкания Tp – считыванием и контролем кода Cy

Числа поразрядно вводятся с естественной запятой и преобразуются в заданном машинном формате (целое или с плавающей точкой).
Можно представить, что кроме дребезга, возможно неуверенное считывание кода сканирования в конце или в начале импульса сканирования. По этой причине предпочтительно синхронизировать считывание по прерыванию.

В системе Кейл нет адекватной модели клавиатуры. Предлагается заменить сканирование клавиатуры массивом кодов сканирования, использовать вручную программное прерывание INT0/INT1 - входы порта P3.2, P3.3 для имитации нажатия.

Программа ввода числа с естественной запятой с клавиатуры и преобразование с плавающей запятой

```
#include <reg51.h>
 char w,x,i,digit,mas[7],min;
 float numb,m;          //массив кодов нажатия клавиш
char code scancode[]={ 0xdd,0x7b,0xde,0x7d,0x7e,0xb7,0xe7 };     //-1,234e
void Delay(int t)
```


5

```
  { while(t--); }
char what(void);
scan( ) interrupt  2    //прерывание INT1
{   Delay(100);
 w=scancode[i];
  what();
     Delay(100);
     }

main ()
{     x=0xFE;
       numb=0;
       i=0;
      m=0;
      IT1= EX1=EA=1;   //маски  и тип прерывания
      while (digit!='e')  ;   //прзнак завершения строки
     while(1);              //динамический останов
 }
char what(void)
{   switch (w)
              {  case 0x77: digit='0'; break;
                  case 0x7b: digit='1'; break;
                  case 0x7d: digit='2'; break;
                 case 0x7e: digit='3'; break;
                 case 0xb7: digit='4'; break;
                 case 0xbb: digit='5'; break;
                 case 0xbd: digit='6'; break;
                 case 0xbe: digit='7'; break;
                case 0xd7: digit='8'; break;
                 case 0xdb: digit='9'; break;
                 case 0xdd: digit='-'; break;
                 case 0xde: digit=','; break;
                case 0xe7: digit='e'; break;
                 default: digit=0xff;
              }
  if(digit==',')
        { m=1;mas[i++]=digit; goto exit;}
    if(digit=='-') {min=1;mas[i++]=digit;goto exit;}
       if(digit=='e')  { if(m)  numb=numb/m;   i=0; m=0;
                    if (min) numb=-numb; goto exit; }
        else {mas[i++]=digit;
                numb=numb*10 + (digit&0xf);
                 m*=10; }
    exit: return digit; }
```

Задание 3.

Организовать в С51 последовательный ввод по прерыванию.

Использовать индивидуальный вариант раскладки клавиатуры

Интерпретировать прерывание чтением кода из массива кодов сканирования. Числа при вводе в формате с естественной запятой преобразуются в формат с плавающей. Комментировать в программе алгоритм ввода.

1)
```
1 2 3 4
5 6 7 8
9 0 (,) .
(-) . . e
```

2)
```
0 1 2 3 4
5 6 7 8 9
(,)(-). . e
```

3)
```
1 2 3 (-)
4 5 6 .
7 8 9 .
0 (,) e .
```

4)
```
8 9 . .
4 5 6 7
0 1 2 3
(-)(,) . e
```

5)
```
1 2 3 . .
4 5 6 .
7 8 9 e
0 (,) (-) .
```

6)
```
0 4 8 .
1 5 9 e
2 6 . (-)
3 7 . (,)
```

7)
```
3 4 . .
2 5 . e
1 6 9 (-)
0 7 8 (,)
```

9. Аналого-цифровое преобразование

Сенсоры (датчики) в составе измерительных, управляющих и медицинских приборов преобразуют параметры состояния внешней среды в **данные**, которые содержат информацию об их значениях в конкретный момент. Преобладает электрический входной сигнал в виде напряжения U~C*X, пропорционального измеряемым физическим величинам X: температура, скорость, давление, освещенность и др. в соответствующих единицах.

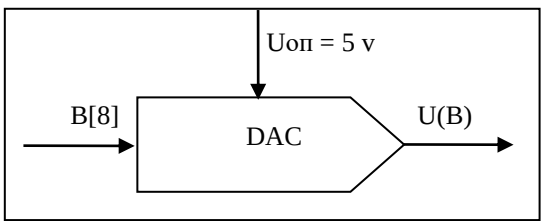
Входные данные в измерительных и управляющих системах, в медицинских приборах, в системах искусственного климата формируются **дискретным** преобразованием **аналоговых электронных сигналов** в **цифровые** форматы дискретных данных (**ADC-преобразование**).

В ПЛК выходные дискретные цифровые данные преобразуются в аналоговые сигналы управления скоростью, температурой, давлением (**DAC-преобразование).** Прямое преобразование сохраняет дискретность в виде ступенчатого аналогового сигнала. Переход к аналоговому сигналу осуществляется интегрированием и фильтрацией.

ADC и DAC преобразования могут быть представлены как внешними, так и встроенными модулями в ПЛК и являются распространенными внешними интерфейсами.

Базовым элементом схемы преобразования является модуль DAC – электронная цифро-аналоговая схема взвешивания и суммирования уровней напряжения для битов двоичного кода.

Обозначение цифро-аналогового функционального



5

преобразователя DAC

B[8] – 8-разрядный двоичный код,
U(B) –выходное напряжение в диапазоне 0 – Uоп,
Uоп = 5 v – опорное напряжение.

Схема ADC преобразования последовательного приближения приведена на рис. 2.10.

Интуитивно схема исполнения и управляющая программа могут быть определены исполнением следующего алгоритма.

Регистр j = 0x80 устанавливается в начальном состоянии и может циклически (**/ror**) сдвигаться вправо, единица последовательно сдвигается по всем разрядам регистра направо.

1) Схема суммирования по модулю два (⊕) формирует двоичный код преобразования аналогового входного сигнала Ux.

Формируемый код D = j (⊕) D записывается в регистр D управляющим сигналом **/wr** и поступает на вход DAC – предполагается значение текущего бита кода равно единице.

2) Если при этом Ux<Ud, то формируется инверсный сигнал **/wrd**, который сбрасывает установленный в D бит повторным суммированием
D = j (⊕) D.

3)Единица в регистре j сдвигается на один разряд вправо управляющим сигналом **/ro**r.

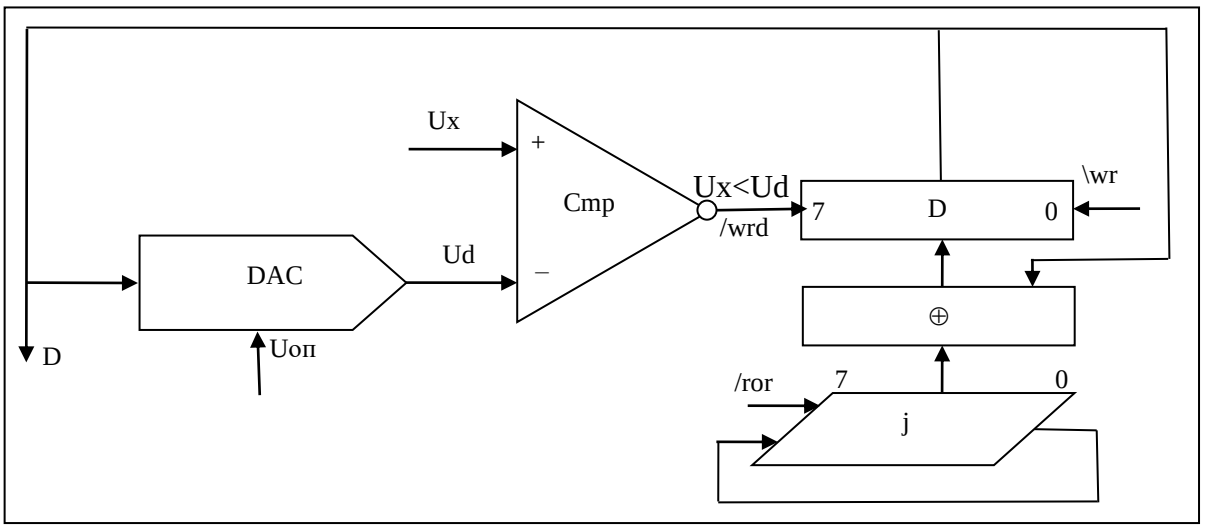


Рис.4.7. Схема ADC преобразования последовательного приближения

Программа моделирования преобразования в C51:

//двоичные коды

```
                                    unsigned char j;  //значение текущего контролируемого бита кода
      unsigned char D;      //8-разрядный двоичный код преобразования
          //аналоговые значения уровней напряжения в вольтах
       float Ud;                    //выход DAC
       float Ub = 5.0/256;          //цена бита в вольтах Uоп/2^8
       float Ux = 4.0;              //аналоговый сигнал на входе в вольтах
       main(){
         D=0;Ud = 0;
         j = 0x80;    //предполагаемый старший бит кода
      do{ D = j ^ D; // D = j + D
      Ud=Ud+(Ub * j);   //DAC преобразование кода D в аналоговый сигнал
         if(Ux<Ud)  //сравнение значений компаратором Cmp
          {D=j^D;                   //сброс бита

                    Ud = Ud – Ub*j;}
            j>>=1;}
            while(j);
            while(1);
          }
```

Временная диаграмма выполнения одного шага алгоритма преобразования (рис.4.7), clk – синхросигнал. Требуются две микрокоманды и два такта синхронизации

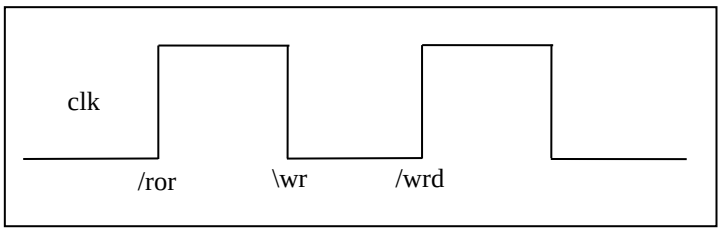


Рис.4.8.. Временная диаграмма выполнения одного шага алгоритма

/ror – уравляющий сигнал сдвига вправо, формируемый положительным фронтом синхросигнала /clk;

\wr – уравляющий сигнал записи, формируемый отрицательным фронтом синхросигнала \clk;

/wrd - уравляющий сигнал сдвига вправо, формируемый положительным фронтом синхросигнала clk.

m1: /ror(j>>=1), \wr(D = D^j); Ud = Ud + Ub*j;
m2: /wrd(if(Ux<Ud); {D = j^D; Ud = Ud-Ub*j;}

Время преобразования 16 тактов.

5

Для демонстрации работы встроенного ADC-преобразования в системе Keil доступен контроллер **SAB515/535** фирмы Infenion (Siemens) с ядром mcs51.

На рис. 4.9. приведена упрощенная схема встроенного модуля ввода двоичного кода значения аналогового сигнала (ADC) в SAB515 (полные настройки не представлены, но типовой программируемый режим доступен, Uоп=5v, регистр результата Addat~D, выделен один из входов Ain0, BSY – сигнал завершения преобразования.

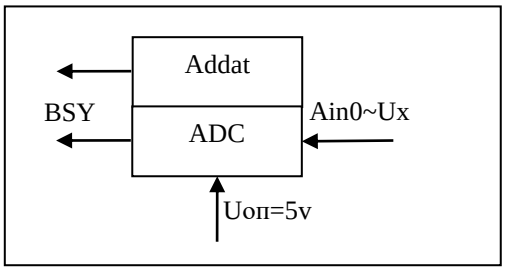


Рис.4.9.. Схема встроенного ADC контроллера SAB515 фирмы Siemens

Значение аналогового сигнала **Ux=(5v/2^8) *Addat**.

В симуляторе Keil сигнальной функцией формируется аналоговый сигнал **sin(x).** Аналоговое значение типа **float** в диапазоне **0** – **1.0** интерпретируется в графике в диапазоне **0** – **5.0 v**. Аналоговый сигнал при вводе через виртуальный порт **Ain0 = sin(x)*2 + 2.0** масштабируется и смещается в положительной области на 2 вольта.

Сигнальная функция (файл **adc.inc**) в интерпретаторе Keil

```
SIGNAL void Signa (time) {
float x;
char i;
while (1) {
   for(i=0; i<100; i++){
  AIN0 = sin(x)*2 + 2.0;  //виртуальный аналоговый ввод
   twatch (time);    // интервал формирования (дискретность)значения в мкс
   x = x + 0.062;
  }}}
   Signa(time)      //запуск функции преобразования
   La Addat         //вывод значения на экране Логического анализатора
   LA AIN0          //график аналогового сигнала в окне Анализатора
```

Период аналогового синусоидального сигнала

100*time*0.062 = 2π *time (мкс)

Порядок выполнения программы измерений:

1) выполняется в цикле библиотечная функция **ADC()** в С51 с ожиданием ее завершения по сигналу **BSY = 0**

```
 adc(void)  //adc-преобразование
 {
   DAPR=0xA0; //запуск преобразования с опорным напряжением 5v/16*A,
while(BSY);          //ожидание завершения преобразования
```

}

2) в командном режиме

#include adc.inc //загружается файл сигнальной функции

3) Запуск **Run** программы измерений ADC.C и сигнальной функции. В окне анализатора формируются графики AIN0 и Addat.

Задача измерения параметров синусоидального сигнала (рис. 2.13). Основные параметры **– Umax, Umin, Uсм, T.**

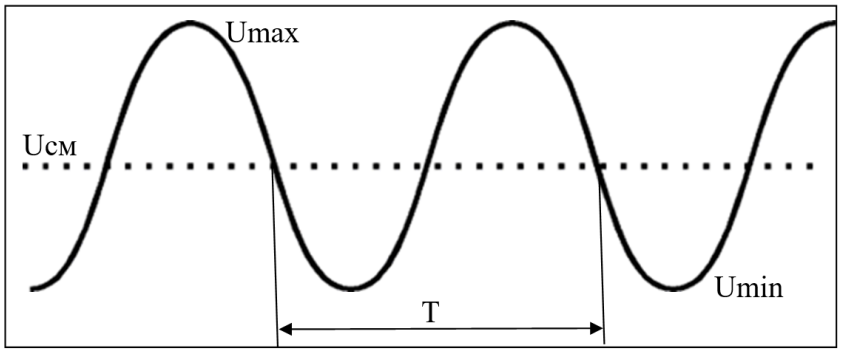


Рис. 4.10. Исследуемый синусоидальный сигнал

Реальные измерения выполняются в условиях помех и случайных воздействий внешней среды. При этом значение **min** и **max** сигнала измеряются с погрешностью и практически при многократных измерениях не совпадают. В динамике выбираем метод измерения периода между двумя переходами через среднюю линию Uсм.

```
#include <reg515.h>  // заголовочный файл из библиотеки Keil
int mm;
unsigned int T;          //период
unsigned char max,min;
float fmax;
adc(void)          //интерпретация adc преобразования
  { DAPR=0;

     while(BSY); }
main()
  { max = 0; min = 0x70;
while(INT0)     //бит доступен для ручного контроля с пульта, по умолчанию равен 1
          {   adc();
               if (Addat>max) max = Addat;
          if (Addat<min) min = Addat; }
                mm = (max + min)/2;   //среднее
```


5

```
                TMOD=1;      //разрешение таймера 0
                TH0=TL0=0; //сброс таймеров
                TR0=0;          //запрет счета таймера 0
                while(adc()>=mm);    //ожидание пересечения среднего с
нарастанием
                    while(adc()<=mm);       //ожидание +перехода к убыванию
              TR0=1;    //разрешение работы таймера, единица отсчета 1 мкс
          while(adc()>=mm);     // ожидание среднего-перехода к возрастанию
                TR0 = 0;                          //конец измерения
                T = ((TH0<<8)+TL0)<<1;    //период
                Fmax = max*5.0/0x100;
                    while(1);
     }
```

Задания в Keil

В С51 выполнить программу измерений амплитуды и среднего значения в вольтах. Измерить период сигнала.

N	Частота (кГц)	Амплитуда (В)
1.	1	2.5
2.	2	3
3.	3	4
4.	4	5
5.	5	2.5
6.	1	3
7.	2	4
8.	3	5
9.	4	2.5
10.	5	3

Литература.

1. Фестингер Л. Теория когнитивного диссонанса / Пер. с англ. А. Анистратенко, И. Знаешева. — СПб.: Ювента, 1999
2. Найссер У. Познание и реальность. — М.: Прогресс, 1981.
3. Шереметьев К. Бешенный креатив.

4.Проектирование цифровых вычислительных машин. / С.А. Майоров, Г.И. Новиков, О.Ф. Немолочнов и др. Под ред. С.А. Майорова. М.: Высш.шк., 1972. 344 с.

5. Таненбаум

6.Сташин В.В. Урусов А.В. Мологонцева О.Ф. Проектирование цифровых устройств на однокристальных микроконтроллерах, М: Энергоатомиздат, 1990.
7.Злобин В.К. Григорьев В.Л. Программирование арифметических операций в микропроцессорах, М:ВШ, 1991 г-303 с
8. Гисин, *В. Б.* Дискретная математика: учебник и практикум для академического бакалавриата. / В. Б. Гисин. – М.: Изд. Юрайт, 2016.
9.*Магда Ю.С.* Микроконтроллеры серии 8051: практический подход – М.: ДМК Пресс, 2008.
10.Keil µVision MDK-ARM 5.20.
11. Павлов А.В. Архитектура вычислительных систем – СПб:ГУ ИТМО,2016
12. Скорубский В.И. Поляков В.И. Зыков А.Г. Математическая логика, учебник, М:Юрайт=2016
13.Введение в психологию
Р.Л. Аткинсон, Р.С. Аткинсон, Э.Е. Смит, Д.Дж. Бем, С. Нолен-
14.В.Н.Крупский , Е Плиско. Математическая логика и теория алгоритмов ДРОФА, 2013г 416с
15.Демидович Е.М. Основы алгоритмизации и программирования.Язык СИ. Минск-Бестпринт-2003.

5

С и с т е м а к о м а н д MCS51 - мнемокоды

Приложение 1

Арифметика и логика

ь ----------

```
add  a,{ri,@rj,#d,ad}  a←a+{...}, призн c,v,p
addc a,{ .....}        a←a+{...}+c,    ....
subb a,{.....}         a←a-{...}-c, .   ...
inc  {ri,@rj,ad,dptr,a}  {....}+1, inc a →p
dec  {ri,@rj,ad,a}       dec a →p
mul  ab     b.a←a*b    v=(a*b>255)  0→c,p
div  ab   a←a/b, b←a%b  (b==0)→ov, 0→c

anl  a,{ri,@rj,#d,ad}  a&{..}→a      0→c,p
anl  ad,{#d,a}

orl  a,{ri,@rj,#d,ad}    a v {…}→ a
  ad,  {#d,a}          a  v  {…}→  Data[ad]
xrl  a,{ri,@rj,#d,ad}    a v {…}→ a
xrl  ad,{#d,a}
clr  a                                 призн p
cpl  a       не(a)
rl   a       rol(a)                    ….   p
rlc  a       rolc(a,c)                 ….  c,p
rr   a       ror(a)                    ….   p
rrc  a       rorc(c.a)                 …. c,p
da   a       коррекция  (+,-)2
```

Пересылки

```
mov a,{ri,@rj,#d,ad}     a←{......}, призн p
mov {ri,@rj},a           {.....}← a
mov {ri,@rj},ad          {.....}← ad
mov ad,{ri,@rj,#d,ad,a}  ad←{.....}
mov {ri,@rj},#d
mov dptr,#d16
movc a,@a+dptr       a←Code(dptr+a), призн p
movc a,@a+pc         a←Code(pc+a)     ….
movx a,{@rj,@dptr}   a←xram{..}       ….
movx {@rj,@dptr},a   xram{..}←a
push ad              Data(+sp)←Data(ad)
pop  ad              Data(sp-)←Data(ad)
     xch  a,{ri,@rj,ad}          a←→{.....}      призн  p
    xchd a,@rj          a(3-0)←→@rj(3-0) …. p
swap a                  a(3-0)←→a(7-4)
```

-

команды булевского процессора

```
mov  bit,c                  mov c,bit
clr  {c,bit}                anl c,{bit,/bit}
cpl  c                      orl c,{.....}
setb {c,bit}                jbc bit,rel
jc rel    jnc rel    jb bit,rel    jnb bit,rel
```

Управление программой и ветвления

```
ljmp a16            PC←a16
ajmp a11            PC(10.0)←a11[10.0]
sjmp rel            PC+2+ rel[7.0]
jmp  @a+dptr        PC←a+dptr
jz   rel            PC+2+rel[7.0],если (a=0)

jnz  rel      ....  ,если (a<>0)
jc   rel      ....  ,если C
jnc  rel      ....  ,если неC
jb   bit,rel        PC+3+rel,если bit=1
jnb  bit,rel  ....  ,если bit=0
jbc  bit,rel  ...   ,если bit=1,bit←0
djnz {ri,ad},rel  {}-1;
                  PC+1/2+rel[7.0],если {}<>0
cjne {ri,@rj},#d,rel ……,если {}<>#d
```

обозначения битов SFR

	7	6	5	4	3	2	1	0	адрес
acc	.	.	.	.	.	.	.	.	e0i
b	.	.	.	.	.	.	.	.	F0i
psw	c	ac	f0	rs1	rs0	ov	.	p	d0i
sp									81
dph									83
dpl									82
ie	ea	.	.	es	et1	ex1	et0	ex0	a8i
p0	.	.	.	.	.	.	.	.	80i
p1	.	.	.	.	.	.	.	.	90i
p2	.	.	.	.	.	.	.	.	a0i
p3	rd	wr	t1	t0	int1	int0	txd	rxd	b0i
ip	.	.	.	ps	pt1	px1	pt0	px0	b8i
tmod	gate1	c/t1	m1	m0	gate0	c/t0	m1	m0	89
tcon	tf1	tr1	tf0	tr0	te1	it1	ie0	it0	88i

		lcall a16	**Data[+sp]← PC+3, PC←a16**	**th0 8c**
		tl0		**8a**
acall a11	**…**	**..., PC(10-0)←a11[10.0]**	**sbuf**	**99**
ret		**PC←Data[sp-]**	**th1**	**8d**
			tl1	**8b**
reti		**PC←стек0□,tf**	**pcon smod . . . gf1 gf0 pd idl**	**87**
nop		**пропуск**	**scon sm0 sm1 sm2 ren tb8 rb8 ti ri**	**98i**

ri= {r0,r1,...,r7}, rj={r0,r1}

psw=(c,ac,f0,rs1,rs0,v,-,p)

p - нечетное число единиц в аккумуляторе

f0- признак пользователя, rs1.rs0 - банк регистров

@r0,@r1 - косвенная адресация к внутренней RAM Data,

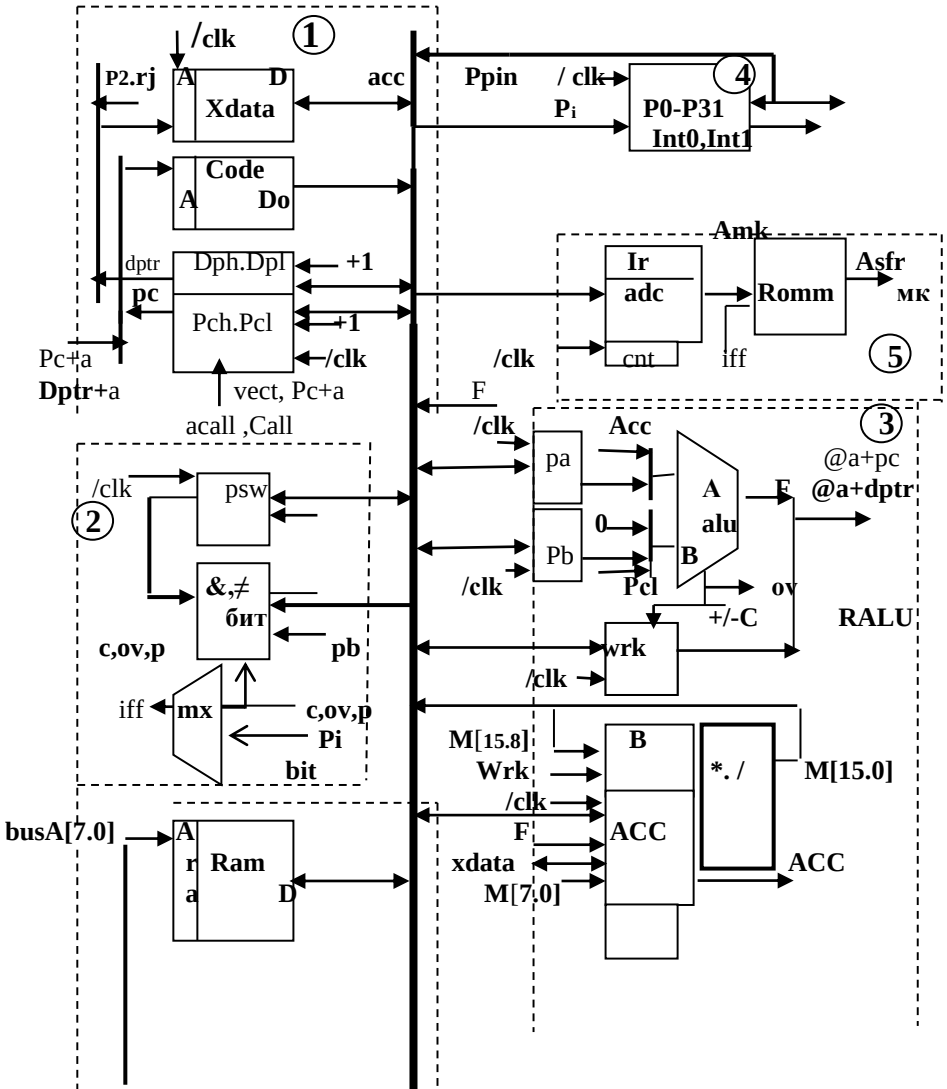
ad - адрес Data, имя регистра SFR

bit - адрес бита в поле битов 00-7f или в специальном регистре- 80-ff ,адрес образуется из собственного адреса регистра, к которому добавляется номер бита (разряд регистра acc.5, psw.0, ... , 80i - адеса битов 80,...87 регистра 80),

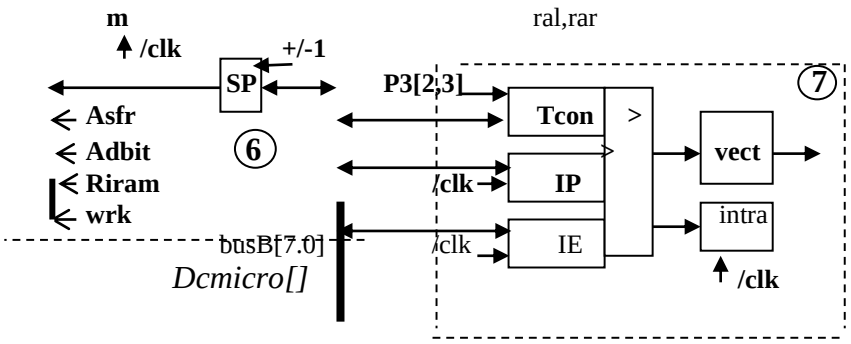
обозначение бита smod,sm0,..., /bit - инверсия бита

rel - <метка>=смещение PC в дополнительном коде

Приложение 2. Структурная схема mcs51



6



Для проектирования HDL-схемы выполняется разбиение структурной схемы на функциональные блоки .

Блок основной памяти (1-**16mem**) включает:

1) Постоянную программную память **Code** , регистр команд **IR**
2) Память данных **Xdata.**
3) 16-битовые адресные регистры-счетчики с прямым доступом (**DPTR, PC),**
4) используются раздельные адресные 16-битовые шины , слово памяти – байт**.**

Блок выборки битов и выполнения битовых операций (2-U**bit)**.
Блок подключается к рабочим регистрам Wrk, PB и содержит регистр признаков **PSW.** Формируется адрес доступа к битам **Adbit** и выбирается значение бита из памяти Ram для условных микрокоманд . Выбораются и формируются биты признаков в регистре PSW.

Регистровое арифметико-логическое устройство (3-RALU) включает
1) схему 8-разрядого универсального программируемого арифметико-логического устройства (**ALU**), рабочие регистры временного хранения операндов **PA, PB**, **Wrk** .
2) 8-битовые регистры с прямым доступом **(ACC, B),** используемые по умолчанию в арифметических и логических операциях, регистры имеют теневое отображение в SFR.
3**)** функциональные элементы параллельного целого **умножения (mul 8*8→16) и целого деления (div 8/8→8,8)**

Блок **(4-Ports)** - Четыре 8-разрядных универсальных порта, подключаемые к контактам **Ppin** корпуса микросхемы, включают регистры и логику управления записью и чтением.

Двунаправленный порт P0 и квазидвунаправленные порты ввода-вывода (P1,P2,P3), связанные с внешними контактами микросхемы, содержат одноименные регистры с прямым доступом и с **теневым** отображением в SFR. Раздельные каналы ввода с контактов Pin, чтения и записи в регистры Pi
Устройство управления (5-Control) содержит схемы декодирования команд ADC, память микропрограмм Rom, схему адресации микрокоманд

Блок внутренней быстрой памяти (6-bus8) включает 8-разрядную память данных **RAM(256 байт),** которая объединяет **Data и SFR** с локальной 8-битовой шиной адреса **BasA[7..0]** и адресным регистром**,** регистр- указатель стека **Sp с прямым и адресным доступом в SFR.**

Блок прерываний (7-interrupt) содержит входной регистр запросов , регистры управления прерываниями **Tcon, IE, IP**, схемы считывания запросов прерываний, выбора наиболее приоритетного запроса, формирование адрес-векторов прерываний.

Шинная организация соединений определяет основные информационные связи между блоками и в дальнейшем представляет
средства упорядоченного обмена данными между функциональными элементами.

Предполагается одна 8-разрядная общая шина данных **busb[8]** и шина управления **,** в которой формируется общий код микрокоманды ***Dcmicro[]*** для передачи для декодирования управляющих сигналов в каждом блоке..

Приложение 3.

Интегрированная система программирования и отладки Keil.

Назначение Интегрированной среды **IDE:**

1)Программирование (редактировать) задачу на языке Ассемблера MCS51, C51. **(new file)** и сохранить в своем каталоге

2)Создание проекта для работы с программой
в своем каталоге).

New project
→ имя.uprov
→Intel
→ выбрать Device 80C51BH
→ отказать Startup
→project
→Manage компонент
→Add Files
→имя.c (включить файл в проект)

Manage component →включить файл в проект)

3)Синтаксический контроль.(**Compile**)

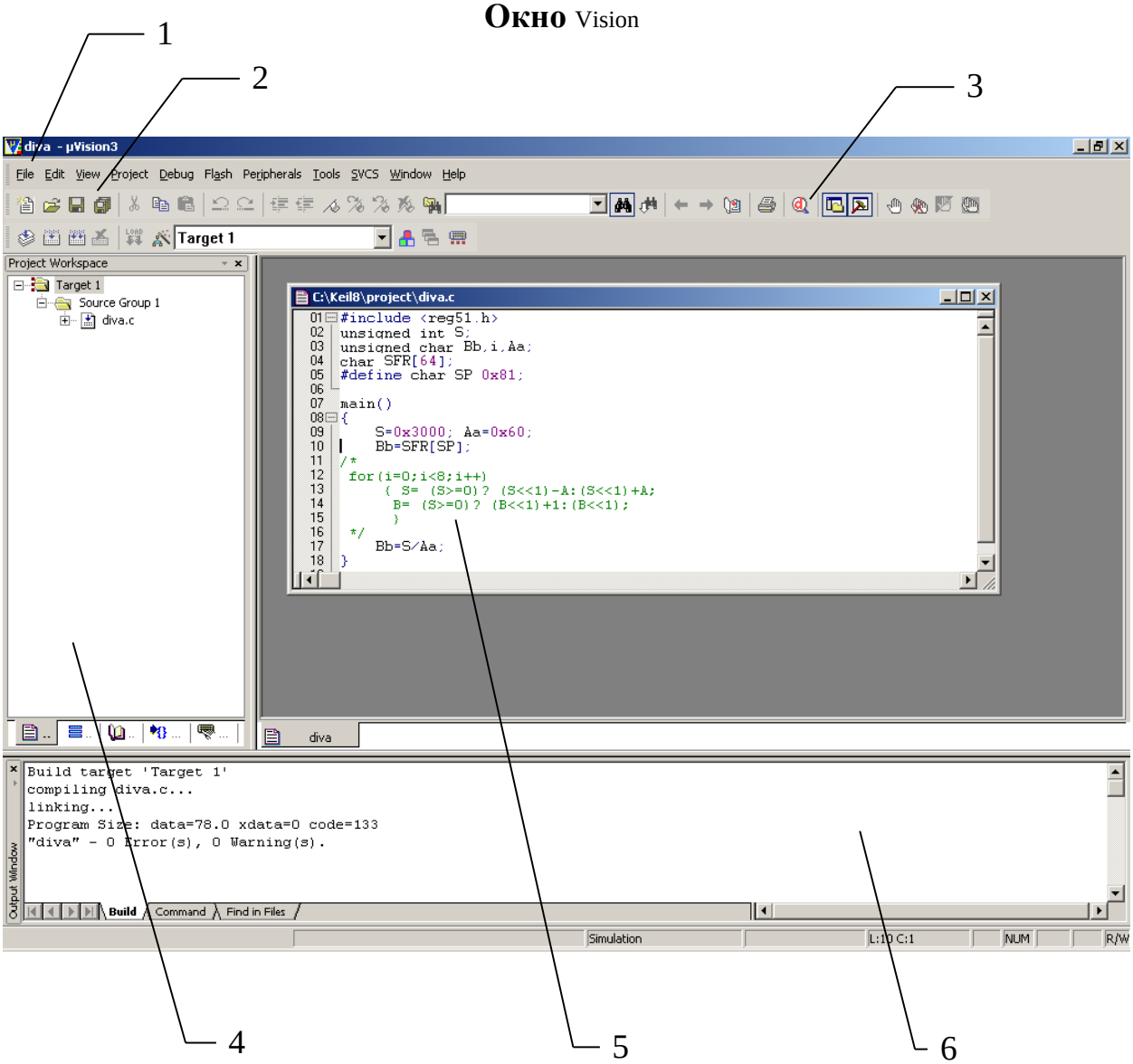
4) Компиляция программы в объектный код (HEX-файл и LIST-листинг)
(**Build)**.

5)Загрузка и симуляция выполнения программы с контролем состояния памяти и периферии.(**Debug**)

6

Система содержит полную библиотеку элементов с ядром MCS51, выпускаемых различными фирмами. Библиотека дополняется новыми элементами в последних версиях, которые можно загрузить из Интернета. Система работает во всех версиях ОС Windows.

Окно Vision



1. Основное меню.
2. Кнопки – синтаксический разбор, компиляция и сборка.
3 Кнопка вызова загрузчика и симулятора.
4. Проект.
5. Окно редактирования исходного текста программы.
6. Окно сообщений компилятора.

Меню Vision

File Edit View Project Debug Flash Peritherial Tools SVCS Window Help

Standart Tools Menu загружается в **Tools** и **View** и содержит символы обращения к различным функциям, локализованным в других ссылках **Menu**

File

New - редактирование текстовых файлов

Open -
Close -
Save - все остальные имеют стандартное назначение

Edit - имеют стандартное назначение

Project

New → μVision project (создать проект)
Import
Open
Close

Manage → компоненты, окрестности (в проект включить файл)
Select Device → библиотека элементов
Options → настройки параметров компиляции и загрузки
Device – выбор модуля
Target - выбор частоты MCU
Out - вывод HEX-кода
List – вывод листинга .lst
C51 – размещение таблицы векторов
L51 – размещение программы Code
размещение данных в памяти Xdata

Build - синтаксический разбор и линкирование
Translate –синтаксический контроль

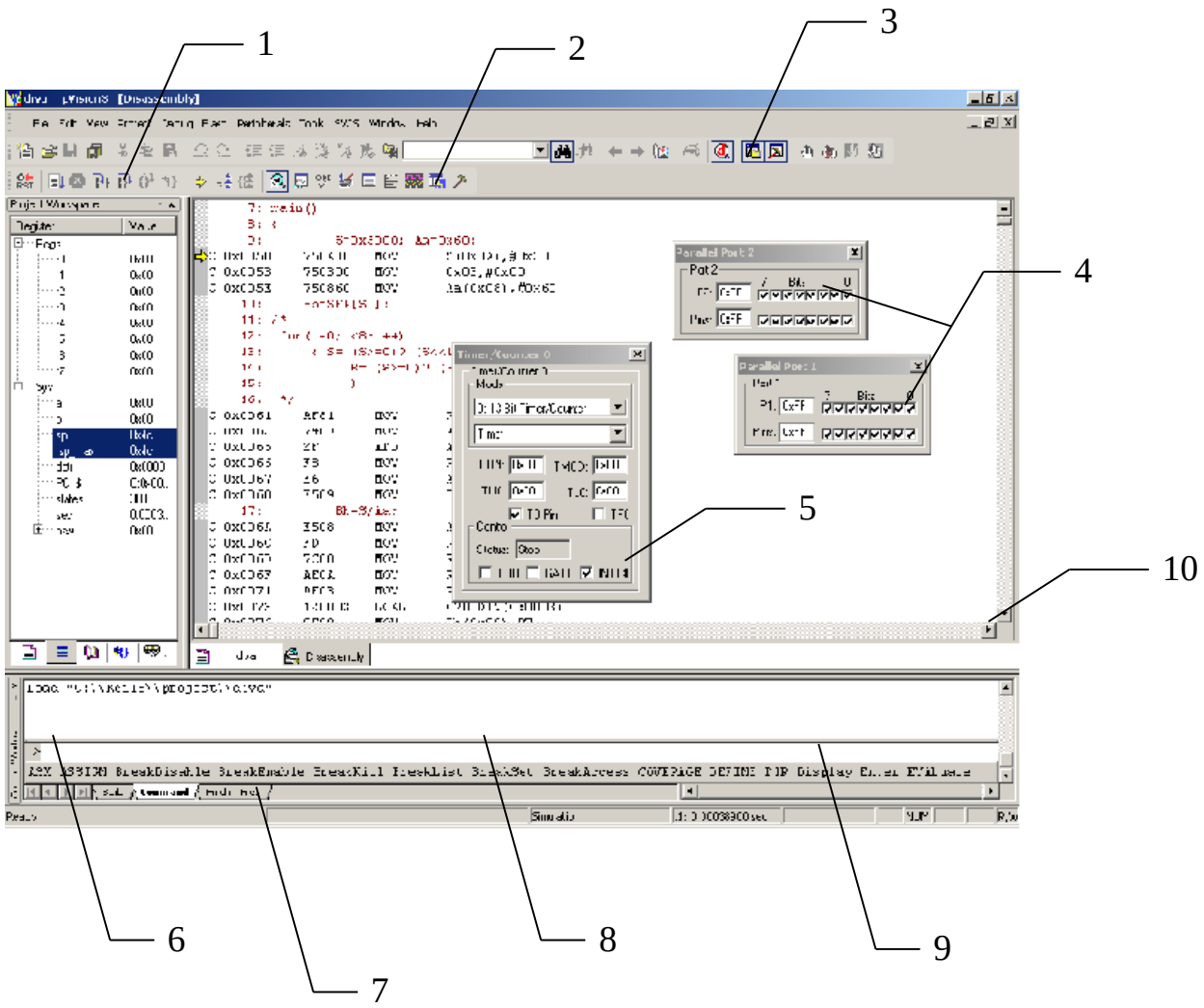
Стандартная функция инициализации проекта **Startup** выполняет
- сброс всех типов памяти Ram
- формирование стека реентранной функции
- формирование значения указателя стека SP с учетом структуры программы
- выполняет инициализацию глобальных переменных

Peritherial –активизируется после загрузки Project и

6

Содержит ссылки на периферию конкретной выбранной в проекте машины.

Окно Загрузчика (Debug)



1. Кнопки управления исполнением программы – автомат, шаг, ..до маркера.
2. Выбор окна Анализатора.
3. Выход из загрузчика.
4. Окна цифровых портов.
5. Окно таймера выбрано из Периферии.

6. Сообщения загрузчика.

7. Командная строка.

8, 9. Размещение окон Watch, Memory – выбираются в меню View.

10. Загруженный исполняемый файл в смешанной форме.

В меню Анализатора выбираем форму оценивания

Grafic **Perfomens Analizer**

В **Grafic** оценивание выполняется средствами, близкими к оцениванию осциллографом

В **Perfomens Analizer** формируются гистограммы оценивания производительности - командами **PA function** в командном режиме **Debug** определяются (максимальное, среднее, минимальное) время исполнения циклической функции **function.**

6