25.11.16

0,,,,,0,0,Министерство образования и науки Российской Федерации

Федеральное агентство по образованию

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

****

**В.И. Скорубский**

**Организация микро-ЭВМ**

**с архитектурой mcs51 и примеры программирования**

# Пособие к лабораторным работам

Часть I



**Санкт-Петербург**

**2015**

***Скорубский В.И. Организация микро-***ЭВМ пособие к лабораторным работам . – СПб: СПбГУ ИТМО, 2015г. – 56 с.

Пособие содержит описание и примеры выполнения лабораторных работ по курсу Организация ЭВМ. В качестве основной технологической базы используется Интегрированная системы проектирования (Integraited Development Environment -IDE) Keil одноименного подразделения фирмы ARM. Для демонстрации логики работы ЭВМ в конкретном конструктивном исполнении используется микроЭвм mcs51 фирмы Intel.

Работы выполняются на двух уровнях – алгоритмическом с использованием языка С51 и уровне Макроассемблера А51. Используются эффективные и наглядные средства отладки в системе Keil на всех уровнях, в частности, графика Логического Анализатора для вывода и имитатор внешних событий в виде Сигнальных функций при вводе.

Пособие используется в курсе «Организация ЭВМ» и может быть использовано при организации компьютерной практики по курсу “Дискретная математика” для специальностей 230100 «Информатика и вычислительная техника», 230101 «Вычислительные машины, комплексы, системы и сети», 210202 «Проектирование, программирование и эксплуатация ИВС», 230104 «Системы автоматизации проектирования».

Лабораторная база продолжает развиваться –по этой причине методические пособия претерпевают изменения, сохраняя основные положения из [1]

В качестве основы для изучения различных вопросов организации и работы компьютеров используется **программная модель** микрокомпьютера MCS51/52, которая является промышленным стандартом и полезна как широко используемая и доступная в приложениях.

Рекомендовано Советом факультета Компьютерных технологий и управления \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2012 г., протокол №\_\_\_\_\_\_\_



**Содержание**

стр.

**Введение**

**I. Программные модели mcs51**

**1.1. Программная модель в С51**

**1.2. Программная модель в А51**

**1.3. Размещение и адресация данных в памяти ЭВМ**

**1.4. Управление программой**

**II. Лабораторные работы**

**2.1. Иерархия памяти ЭВМ……………………………………………………**

**1) Программирование в С51 с прямым доступом**

**2) Программирование в А51**

**3) Программирование в С51 с указателем**

**Задания к лабораторным работам**

**2.2. Ввод-вывод численных данных**

**2.2.1. Преобразование целых при вводе и выводе численных данных.**

**2.2.2.Преобразование дробных при вводе и выводе численных данных.**

**2.3. Двоичная арифметика с фиксированной точкой**

**2.3.1. Умножение**

**1) Умножение положительных дробных чисел**

**2) Умножение целых**

**3) Умножение знаковое в библиотеке C51**

**2.3.2.Деление**

**1) Деление дробных двоичных чисел**

**2) Деление целых чисел со знаками в библиотеке С51**

**3) Совмещение деления с дробным умножением**

**2.3.3. Извлечение корня квадратного**

**2.3. Вычисление функций**

**2.3.1. Вычисление с плавающей точкой**

**2.3.2. Вычисление функций с фиксированной точкой**

**1) Вычисление функций с десятичным масштабом**

**2) Вычисление функций с двоичным масштабом**

**3) Исключение деления**

**Задания к лабораторным работам**

**2.4. Битовые данные**

**2.4.1. Доступ к битам в С51**

**2.4.2.Доступ к битам в A51**

**Задания к лабораторным работам**

**Литература**

**Приложение 1. Система команд MCS51 – мнемокоды**

**Приложение 2. Интегрированная система программирования**

**и отладки Keil**

**Приложение 3. Вопросы по курсу лабораторных работ**

**к зачету и экзамену**

**Введение**

МикроЭВМ **(MCU)** (mcs51, sab515, Pic16, Cortex..) в отличие от **микропроцессоров общего назначения** **( mcs80, mcs486, Pentium** ) ориентированы на применения во **встроенных и портативных** системах контроля, управления и измерений.

В MCU **интегрированы** и согласованы на уровне стандарта разнообразные интерфейсы и средства **прямого управления** периферией, принципы организации программного управления - традиционные , понятие процессора (CPU) не акцентируется.

Знакомство с архитектурой и принципами работы микрокомпьютера поддерживается лабораторными работами с использованием программирования.

Большое число конструктивных решений MCU не позволяет использовать какое-либо обобщенное представление об их архитектурах. В учебных курсах по направлению **Организация ЭВМ** приходится опираться на некоторую конкретную схему.

Для первого знакомства можно выбирать любой MCU, распространенный, документируемый и поддерживаемый средствами **Integrate Development Environment (IDE)**.

Одной из актуальных представляется **микроЭВМ MCS51** (фирма Intel 1980 )- промышленный стандарт в приложениях 1980-2000-х годов

Выбор микроконтроллера MCS51 [1] обусловлен

1) популярностью ,

2) относительно простой и доступной для изучения схемотехникой .

3) На уровне ПЛИС вся схемотехника размeщается как ядро системы на кристалле

4)многообразием расширений и модификаций, сохраняющих ядро MCS51/52.

5)наличием эффективных и доступных средств программирования и отладки.

6)доступным лабораторным оборудованием

Для записи алгоритмов в микроЭвм широко используется адаптированный к конструктивным особенностям mcs51 язык С51. Этот уровень представления MCU определяем как **Программную модель** **высокого уровня в С51**.

Детали построения и работы MCU представлены системой команд на уровне **Ассемблера А51**, под которой подразумевается **микроархитектура** ЭВМ.

Для ее описания может быть использован С51, включающий расширения, позволяющие представить функциональную схему ЭВМ.

Дальнейшая детализация может быть представлена микропрограммными моделями, реализуемыми управляемыми функциональными схемами. Этот уровень представления MCU определяем **Программной моделью в Ассемблере и структурной схемой**. Представление программ в Ассемблере А51 определяет операционную **семантику** алгоритмического языка С51.

Программные модели MCU являются спецификациями при конструировании новой машины в **технологии ASIC**.

Основным методом **тестирования** алгоритмов является исполнение скомпилированной программы, доступной также для анализа в виде листинга в Ассемблере.

Для понимания принципов исполнения алгоритмов, представленных в данной программной модели, требуются доказательства,

Некоторые из них доступны в общих разделах Дискретной математики, но в большинстве случаев конструктивны. В действительности, приходим к **тестированию** интуитивно разработанных программ**.**

Цикл лабораторных работ опирается на средства моделирования, представленные популярным программным комплексом **IDE Keil** [1]. (См. Приложение 2.)

В библиотеке KEIL представлены около сотни фирм и некоторые из них предлагают десятки различных типов MCU с ядром mcs51 - отличаются разнообразием внешних интерфейсов, ресурсами памяти, наличием специальных средств управления питанием, частотой, сбросом. Библиотека постоянно расширяется в новых редакциях Кейл, что свидетельствует о сохранении актуальности MCU в приложениях.

Предполагаются две части, соответствующие двум семестрам.

**В первой части** рассматриваются:

- организация памяти и типы данных в программных моделях С51 и А51

- декодирование двоично-кодированных форм записи данных при вводе и выводе

* арифметические операции, их применение к целым и дробным числам.

- алгоритмы вычислений стандартных функций

* логика вычислений с битовыми и булевскими типами данных;

**Во второй части**:

- система прерывания, измерение времени и ШИМ

* прямое программное управление вводом с клавиатуры,

- ввод данных с использованием аналого-цифрового преобразования, вывод с цифро-аналоговым преобразованием

* последовательный канал передачи данных USART.
* Синхронный интерфейс I2C,SPI

При этом используются MCU Sab515 (Siemens,/Infinion), Aduc8хx (ADC) с ядром MCS52 , которые позволяют эмулировать некоторые из распространенных внешних интерфейсов.

Большой опыт использования в промышленности и образовании отражается в разнообразной литературе по архитектуре mcs51**(**первое подробное описание на русском языке [1] 1990 г, в материалах к лабораторным работам предлагаются оригинальные фирменные тексты -описания mcs51, C51, A51)

1. **Программные модели mcs51**

Признанная схема организации любого компьютера

Внутренняя процессор

память

периферия

внешняя

память

В микроЭВМ интегрированы блоки внутренней памяти и процессор- конструктивно интегрированы в одной микросхеме. В более мощных компьютерных системах блоки интегрированы на одной печатной плате

**1.1. Программная модель в С51.**

Программная модель ЭВМпредставляет ресурсы памяти, интерфейсы ввода-вывода, нееобходимые и достаточные для программирования в Си. Применение стандартного языка Си (стандарт ANSI) к конкретным MCU имеет ограничения, поэтому используются модификации с расширениями в С51,

учитывающие свойства конкретной ЭВМ.

Язык Си получил широкое распространение во встроенных применениях, в первую очередь, благодаря присутствию в нем как общезначимых аппаратно-зависимых понятий, так и возможности создания на функциональном уровне

аппаратно-зависимых расширений. Такая модификация Си доступна для всех существующих моделей MCU (Avr, Pic, TMS, Intel, ...)

Для mcs51 фирмой Keil предлагается аппаратное расширение стандарта

С51 с ассемблером А51.

Диаграмма MCU(рис.1.1.) представляет доступную иерархию памяти и интерфейсы ввода-вывода в С51.

Все типы памяти отличаются объемом , способом доступа и временем доступа. В виде диаграммы приведена программная модель в С51

SFR Code

bit

Data

Xdata

Tmod Usart

Tcon P0 P1 P2 P3

Рис. 1.1. Программная модель ЭВМ в С51.

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

В С51 предполагается **прямой (безадресный )** доступ к данным в памяти ЭВМ. Идентификатор переменной обозначает значение безотносительно к режиму доступа к конкретному типу памяти.

Как и в стандарте языка С используется косвенный адресный доступ к данным в ЭВМ по указателю **pointer** или тип памяти **Idata**.

В С51 определена иерархия памяти Ram из нескольких блоков **Data, bit, Sfr, Idata**.

**Data** – память данных – неявно доступная при определении переменных,

объем 128 байт

**char x;**  //байт данных со знаком

**int y;** //формат 2 байта со знаком

long z; //формат 4 байта со знаком

Принципиальное значение имеет размещение формата в памяти – существуют в схемотехнике два варианта – последовательное со старших байтов к младшим **(BigEndian**) и обратный – с младших к старшим**(LittleEndian).**

целое число long a3 a2 a1 a0

04 8F F4 EA

адреса в Data 8 9 A B

**Регистровая память** **Sfr**(8-разрядные регистры специальных функций) – 128 байт адресное пространство. В С51 и А51 **резервированы** идентификаторы для специальных регистров, используемых в периферии и системных модулях. В том числе 8-разрядные порты ввода-вывода **{P0, P1,P2,P3},** регистры управления таймерами **{Tmod,Tcon**}, регистры управления последовательным интерфейсом **Usart** и др.

**Битовая память Bit –** поле из 256 битов , размещаемых последовательно в выделенных форматах данных и доступные в них по номерам битов, доступ – чтение и запись.

Доступ к битам в **С51**

**bit x1,x2; //**битовые переменные в поле бит

**sbit y1=P1^2**; //бит порта P1 (не смешивать с операцией ^ - исключающее ИЛИ

**char bdata** mem //байт данных с битовой адресацией в сегменте bit памяти Datа

y1: **bit mem.0** ;0 бит ячейки mem

y2: **bit mem.1**

В **А51**

x1 bit P1.0 ;бит порта

z2 bit acc.1 ; бит аккумулятора

**bseg at 10** ;абсолютный сегмент битов с10-го адреса сегмента битов

x3**: dbit 2 ;** поле из двух бит

**2)Расширенная память данных Xdata** - 65 Кбайт адресное пространство, доступ к данным – чтение и запись

**char xdata var;** // доступ к байту в памяти **Xdata**

**char pdata var**; //доступ к переменной в странице P2 из 256 байт

**3) Постоянная (энергонезависимая) память** программ **Сode** – 65 Кбайт адресное пространство, доступ – чтение данных, чтение и исполнение команд

Компилятор формирует исполняемый код в этой памяти, запись-загрузка выполняется специальными средствами и невозможна при исполнении программ.

**1.2. Программная модель в А51**

В программной модели А51 уточняется организация памяти с учетом режима адресного доступа к данным.

**Ram**

**r0-r7**

Data

**Stack**

**Bit**

**SFR**

**SP**

**th0,tl0,th1,tl1**

**tmod**

**Ex0,ex1,et0,et1**

**Acc,B**

**PSW**

**Code**

**Xdata**

**PC, DPTR**

**scon, sbuf**

**P0 P1 P2 P3**

.

Рис.1.2. Организация памяти mcs51 в А51.

**1.4. Размещение и адресация данных в памяти ЭВМ**.

1. **Прямая адресация к данным в памяти Data**.

**В С51** неявно подразумевается в задании типа данных

char x,y;

int z[20];

**В А51** может быть определена в **абсолютном сегменте**

**Dseg at 0x30** ; адрес начала сегмента

**X: ds 2 ;X-**адрес первого (0) байта блока данных из 2-х байт

**Dm: ds 4**

Команды **mov X,Dm** ;Data[Dm]🡪Data[X]

**mov a, Dm+2** ; Data[Dm+2]acc

Для отдельных байтов можно определять адреса в Data псевдокомандами

**Ss equ 0x22**

**Ее data 0x66**

1. **Регистры*****SFR*** *в* ***Ram (80-FFh)****,* 128 байт

Все регистры прямо доступны по именам (**Acc, B, PSW,Sp**,..).

**В С51** адреса определяются загрузкой адресного файла

**#include <reg51.h>** { reg515.h, ADuC812.h,..из каталога С51/INC}

В меню проекта **Project/Option../Device/Inеel** выбирается конкретная фирма (Intel,Infenion,ADI,TI,...) и микросхема (80с51BH,Sub515, ...).

**В А51** достаточно в меню выбрать микросхему.

По адресам **(0x80-0xFF)**  в А51 свободные адреса в SFR могут быть определены как  **TT equ 0x95**

В С51 **sfr TT=0x95** – свободный регистр

**sfr16 y=0xA1** определяет адрес двух смежных регистров в SFR(Big Endian-размещение). В компиляторе и симуляторе все регистры доступны. К сожалению, в реальных микросхемах не гарантирован доступ к конкретным свободным адресам.

Зарезервированные ячейки (**Acc, B, P0,P1,P2,**,..) также дублируют неявно доступные рабочие регистры и могут рассматриваться как **теневые** при выполнении операций с неявными рабочими регистрами. Это значит, что запись по адресу (в программе) и неявная в рабочие регистры(в микропрограмме) **когерентны**. Только один регистр АСС имеет различимый идентификатор с рабочим регистром А(в программе С51 запрещен). Все остальные имеют общий идентификатор и адресный доступ и неявный к рабочему регистру(например, регистр B можно использовать в С51 ).

**DPTR** – 16-разрядный адресный регистр (Data Pointer) доступа к памяти программ Code и данных Xdata. Доступен в SFR по прямым адресам образующих его 8-битовых регистров **DPTR=DPH.DPL** .

**3) Системный неявно доступный регистр PC .**

16-разрядный программный счетчик или регистр адреса команды. При включении питания автоматически устанавливается РС=0.

РС не входит в число адресуемых регистров SFR, но может контролироваться косвенно и модифицируется неявно.

**4)Регистры общего назначения Ri={ R0,R1,..R7** } (регистровая адресация) – активный регистровый банк в памяти Data.

Доступны 4 банка, совмещенные с начальными ячейками памяти **Data**. Активный банк **RS[1.0]** выбирается в регистре **состояния**

**PSW= c ac f0 rs1 rs0 ov . p**

7 6 5 4 3 2 1 0

Адрес соответствующей ячейки Data определяется смещением относительно банка (**RS1.RS0).Ri (**например, в 3-ем банке регистр R2 имеет в **Data** адрес

R2~000**1 1**010~ 0x1A.

В команде **mov 01, r2** прямой адрес Data 01 совпадает с регистром 00.r1

В С51 предусмотрено автоматическое переключение банков и сохранение

**контекста** при обращении к подпрограммам и в прерываниях.

**void func( ) using 2**

**………………….**

**ret**

Здесь назначен второй банк рабочих регистров при обращении к функции **func( )**, что позволяет сохранить регистры текущего банка и переключиться к банку 2, при выходе **ret** восстановить текущий банк.

Схема доступа к регистру Ri при исполнении команды

**mov A,Ri**

**7 0 7 0**

**PSW IR-**регистр команд

**2 0**

**4 3**

rs[1.0] Ri={0,1,..7}

7 4 3 2 0

A

Ram

D

A

Рис.1.4. Схема доступа к регистру Ri

Исполнимое в С51 описание элементарной схемной операции называем **функциональной микрокомандой**

A=**Ram[(PSW&0x18)|(IR&0x07)];**

**5)** **Косвенная регистровая адресация**

В С51 обозначается явно косвенное обращение к памяти IDATA**~Data(0-0x7f)**

**char idata x;**

**x=55;**

В А51 исполняется как

**Iseg at 0x10** ;размещение сегмента в свободной области Data

**x ds 1**

**………………**

**mov R0,#x ;** косвенный доступ к сегменту

**mov @R0,#37**

Косвенное обращение по адресам **Sfr** (0x80-0xff**)** не определено и в симуляторе контролируется.

**6) Стековая адресация –** вид косвенной регистровой адресации в **Data** с **автоиндексацией** с использованием регистра-указателя SP.

**В С51** стек формируется неявно компилятором при обращении к подпрограммам и в прерываниях.

**В**  A51 размещение памяти под **стек** в Data выбирает пользователь .

По сбросу при включении питания указатель стека SP =7.

Неявная индексация: пре-автоинкремент **(+SP)** при записи и пост-автодекремент **(SP-)** при чтении.

Размещение Стека в абсолютном сегменте Data

**Dseg at 0x10** ;сегмент стека

**Stack: ds 6**

**……………**

**Start:** ;начало программы

**mov sp,#Stack-1** ; начальное значение стека с учетом **пре-инкремента** (+SP) при записи в стек по команде **Push ad,** при чтении в команде **Pop ad** **пост-декремент**.

**7**) **Режимы адресации в памяти данных Xdata**

**В С51** - прямой доступ по имени **char xdata xx,yy;**

**-** страничный доступ **char pdata x; (в** регистре порта Р2 предполагается адрес страницы из 256 байт)

В **А51 -** два режима адресации

**А)** Доступ неявный **косвенный через dptr**

**xseg at 0x100** ;абсолютный сегмент

**mm: ds 50 ;**адрес первого байта массива 50 байт

............................

**mov dptr,#mm** ;адреc

**movx a, @dptr** ;Xdata(dptr)  А

**movx @dptr,a**

**Б)** Доступ неявно **страничный**

mov r0,#xx

**movx a, @r0** ; **Xdata(**[**P2.@r0**](mailto:P2.@r0)**)** А, в P2 адрес страницы,

@r0 –смещение в странице

**8) Режимы адресации данных в памяти** **Code**.

**В С51** размещение констант – прямой доступ по имени строки

**char code x[]="abcdef";**

**int code y[]={221,332,44,55};**

**В А51** Непосредственная адресация

**mov a,#55 ;** Code[PC+]  a

**mov a,#55h = mov a,#0x55= mov a,#01010101B**

Относительнаяадресация

**movc a,@a+pc ;** Code[PC + a]  a **;** относительно

текущего РС**,** в АСС размещается индекс

Базовая индексная адресация (базовый адрес в DPTR, индекс в АСС)

**movc a,@a+dptr;** Code[dptr+а]  a

Определение абсолютных сегментов

**cseg at 0** ; старт при сбросе и включении питания в памяти Code

jmp start

....................

**cseg at 0x40 ;** абсолютныйсегмент памяти Code – прикладная программа

start: jmp first

**yy: db “abcde” ;** адреспервого байта константы в виде строки

**.................**

**first:**

**mov dptr,#yy ;** сохранение адреса

**movc a,@a+dptr**

**9)Адресный указатель pointer** определяет в С51 косвенный адрес данных.

Возможность переадресации с использованием указателя является важным достоинством языка С, позволяет организовать работу со сложными структурами данных.

**В С51** необходимо различать, в какой памяти размещаются данные, формат данных . Реально при компиляции в ассемблере используются рассмотренные режимы адресации к конкретной памяти и понятие **pointer** определяет соответствующую процедуру компиляции.

К указателю в С51 применимы целые арифметические операции **(+, -,\*)**

**\*xx++= \*(yy+2\*z)**

По умолчанию, указатель и операнд размещаются в памяти Data

**char \* xx =”abcdef”; //xx –** адрес начала строки в Data

Обращение косвенное к байтам **\*xx=0x55;**

Обращение косвенное  **\*(int \*)xx=0x1234;** //возможность обращения к //одному и тому же массиву данных в разных форматах

**char code \*aa**=”abcdef”; //константа размещается в памяти Code, указатель-адрес начала константы в **Data**

**char xdata \*aa;** //данные в **xdata**, указатель-адрес в **Dat**а

Константы определять в Data и Xdata не целесообразно, так как реально они хранятся в памяти Code.

.

Доступ к данным в А51 адресный в отличие от прямого в С51, где идентификатор переменной определяет конкретный тип памяти.

movc a,@a+Pc mov P3,P2

Code

Xdata

ACC

Ri

Data

SFR

P0─P3

mov P0,#55

mov a,#22 mov SFR,a mov P2,r1

mov R0,#33 mov a,SFR mov P3,P2

mov Ad,#11

movc a,@a+dptr mov a,Ri

mov a,@daptr

mov @dptr,a

mov Ad1,Ad2

mov R1,Ad mov a,Ad

Рис.1.2. Диаграмма доступа к данным в командах **mov**

Все представленные типы памяти различаются режимом доступа к данным (прямой или косвенный) и типом данных (бит, байт)

**1.4. Управление программой.**

**Операторы управления программой в C51**

**Goto** метка;

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

**while**(условие) оператор **;**

**do {** оператор **} while (условие) ;**

**for (i=0;i<N; i())** оператор **;**

**switch** (ss) {

**Case** 0x55 **of :** [ ]; **break;**

…………………..

**Case** 0x66 of : [ ]; **break;**

**Default:** [ ];

}

В C51 могут быть определены **рекурсивные (рекуррентные) вычисления,** в которых циклы заменяют рекурсии.

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

Например, вычисление факториала определяетcя рекурсивной функцией

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

1. **рекурсивная программа**

**int f=1; char i=0;**

**fact(char n) reentrant**

**{ if(i<n) {i++;f\*=i;**

**return fact(i);}}**

**main( ){ fact(5);}**

Рекурсивное описание задачи – формальное в теории алгоритмов. Вывод формулы является прямым доказательством правильности алгоритма.

**2)** Эквивалентная по смыслу циклическая программа вычислений

**char n,i=0;**

**int f=1;**

**main()**

**{ n=5;**

**while(i<=n)**

**{i++; f\*= i;}**

**}**

**Команды управления программой в А51**

Определение абсолютных сегментов

**cseg at 0** ; старт при сбросе и включении питания в памяти Code

jmp start

....................

**cseg at 0x40 ;** абсолютныйсегмент памяти Code – прикладная программа

start: jmp first

**yy: db “abcde” ;** адреспервого байта константы в виде строки

**.................**

**first:**

**mov dptr,#yy ;** сохранение адреса

**movc a,@a+dptr** ; Code(dptr + a)  a, базовая адресация- база в DPTR, в АСС смещение

Команды управления формируют состояние программного счетчика PC

**jmp метка ;** метка  PC

Компилятор А51 выбирает одну из модификаций – **sjmp** (короткое смещение PC+(+/- 7-битовое смещение)), **ajmp** (11-битовый адрес в странице с номером

PC[15..11]), **ljmp** (16-битовый адрес)

**jmp @a+dptr ;**функциональный **switch** переход PC=a+dptr

**call метка ;** PC  Data(+SP), метка  PC и переход к

подпрограмме. Компилятор выбирает одну из модификаций **lcall**(16-битовый адрес перехода) или **acall**(11-битовый адрес в странице)

**ret ;** Data(SP-)  PC возврат из подпрограммы

В следующих командах компилятор формирует (+/- 7-битовое смещение)

**jc/jnc метка**

**jz/jnz метка,** переход, если АСС (=0)/(!=0)

**jb/jnb bit, метка ;** переход по значению бита

пример **jb ACC.0,start** переход по значению бита ACC.0

**djnz {ri,ad}, метка ;**  [{..}-1, if ({..}#0), PC+ смещение]

**cjne (ri,@rj,ad} ,#d, метка ;** if ({..}#d) PC+смещение;

**reti** – возврат из прерываний

**1.3.Проектирование алгоритмических схем.**

В логическом синтезе схем ЭВМ в ПЛИС используются стандартные текстовые языки проектирования высокого уровня – **VHDL, VERILOG, AHDL** и визуальные графические схемы.

Одной из распространенных является система проектирования **MaxPlusII[4.5]** на основе ПЛИС фирмы Altera и ее расширенная профессиональная версия **Quartus\_II**.

Схемы строятся из стандартных функциональных элементов библиотеки , имеющих графическое и текстуальное описание. Текстуальное описание относится к **языкам регистровых передач RTL**. В графике могут быть использованы для ограниченного числа элементов памяти более простые изображения и элементы преобразования данных, которые относятся **к структурным схемам** и упрощенный язык регистровых передач.

Алгоритмы выполнения преобразования данных могут быть представлены в виде микропрограмм, а структурная схема является **микропрограммной моделью ЭВМ**.

Cтруктурная схема в графике позволяет наглядно представить всевозможные передачи и преобразования в микропрограммах.

Ограничимся при описании алгоритмов очевидными структурными схемами, а для описания микропрограмм, которые управляют этими схемами, языком Си,ассемблером, мнемоника которого, по- существу, эквивалентна используемым управляющим сигналам в структурной схеме.

На рис.1.4. приведена упрощенная структурная схема mcs51

Выполнение двух байтовой команды **add a,#10** можно представить функциональной микропрограммой в С51

**char A,Pa, Pb, Ir,Code[1000];**

**int PC;**

**main()**

**{ Ir=Code[PC++];**

**Pa= Code[PC++];**

**Pb=A;**

**A=Pa+Pb;**

**}**

Здесь формирование признаков естественно контролируется реальной командой Add ассемблера.

Более сложные алгоритмы исполнения команд (умножения и деления) можно демонстрировать в С51 и в ассемблере А51.

**A D acc P0-3pin**

**Xdata**

**Code P0-P3**

**A Do**

Dph.Dpl **+1 Ir**

**adc Rom**

Pch.Pcl **+1**

F

pa @a+pc

psw **А @a+dptr**

**0 alu F**

Pb B

&,≠

**wrk**

**B**

**Wrk \*. / M[15.0]**

**busA[7.0] A F ACC**

**Ram xdata ACC**

**D M**[**7.0]**

ral,rar **+/-1**

**SP ]**

**Asfr Tcon >**

**Abit > vect**

**Riram clk IP**

**wrk**  intra

**busB[7.0**] clk IE

Рис.1.4. Структурная схема

***II. Курс Лабораторных работ***

Целью курса является изучение и работа с разнообразными типами памяти и режимами адресации, выбор решения алгоритмической задачи – интуитивное или аналитическое, высокоуровневое программирование в С51 и ручная трансляция в А51.

**2.1. Иерархия памяти ЭВМ**

**Задание.**

Требуется выполнить преобразование записи десятичного целого числа в виде текстовой строки в ASCII-кодах в двоичную текстовую строку.

“25” 🡪 “11001”

Метод преобразования десятичного числа в двоичную пересчетом в двоичной системе можно представить рекуррентной формулой

**N= 65432=a4a3a2a1a0  =a4\*104 +a3\*103+a2\*102+a1\*101+a0** = ∑ai10i

= (0\*10+a4)10 +a3)10+a2)10+a1)10+a0 =

S1= (S0\*10+ 6) i=0, S0=0

S2= (S1\*10+5)

S3= ( S2\*10+4)

**……………………**

**(1.1) Si+1= Si\*10+an-i** S0=0 ,i=0,1,..n-1 **a0 a1 .. ai ai+1**цифры , упорядоченные в памяти Big Endian

,

1. **Программирование в С51**

Загружаются библиотечные файлы

**#include <reg51.h>** //адреса регистров SFR модели, выбранной из библиотеки MCU

**#include <string.h>** // строчная подпрограмма из библиотеки

Выбирается размещение данных в памяти программной модели и символические ссылки, которые обозначают прямой доступ к данным.

**сhar i,x;** //двоичные числа в памяти Data,

**char code** y[ ]= “123”; //символьная строка в памяти Code

char **xdata** yy[8]; //двоичная строка в расширенной памяти Xdata

В программе формализован алгоритм преобразования строк. Перевод 10/2 –запись рекуррентной формулы циклической программой. Перевод двоичного числа в строку –интуитивный алгоритм замены битов ASCII кодами.

**main()** //преобразование десятичного числа из ASCII в машинное char x

{  **x=0;**

**for (i=0; i<strlen(y); i++)**

**x=x\*10+(y[i]&0x0f);**

**for(i=7;i>=0; i--)** //преобразование двоичного числа **char x** в ASCII

**{ yy[i]= (x&0x01) ? ‘1’ : ‘0’ ; x=x>>1;}** }

Синтаксический контроль алгоритма осуществляется при компиляции программы и семантический тестированием в системе Кейл.

Объем программы в памяти Code при компиляции 114 байт.

**2)Программирование в А51**

Алгоритм в С51 используем как спецификацию и комментарии в А51. Выбирается размещение данных в памяти и символические ссылки, которые определяют абсолютные адреса. При выборе памяти учитываем интенсивность

использования, адресацию (время доступа и кодирование режима в командах, доступный объем памяти) . Преимущество имеет регистровая адресация Ri,

затем прямая и косвенная адресация в Data, относительно медленная память Xdata большого объема, константы хранятся в памяти Code.

**A)** Схема преобразования в двоичный формат . с микропрограммным управлением

**for (i=0; i<strlen(y); i++)**

**x=x\*10+(y[i]&0x0f);**

#Y wrY

Dptr incd

#3

rdCod Code clra Ci wri

A wra I inci

v #0x0f

anl mul #10

add

Память – постоянная память микрокоманд Code c адресным регистром dptr

* регистр аккумулятор A,
* счетчик тактов i

Функциональные элементы : anl ~поразрядная схема &.

add~параллельный сумматор

mul ~8\*8 умножитель

Управляющие сигналы **wrY**🡪запись адреса в **dptr, rdCod🡪V** чтение символа из памяти **Code, wra🡪A=**A\*10+V&0x0f, **clra🡪**A=0**,**

**wri🡪**i=3**,inci🡪**i+1

Ci – сигнал переноса (признак завершения цикла i=8.

Микропрограмма в А51

i **equ** r0

**cseg at 0** ;начало сегмента Code после сброса

**jmp start** ; команда старта

**Y: db “123”** ;текстовая константа в сегменте

**start:** **clr a**

**mov i,a**

**mov dpl,#yy** ;wrdp

**cikl*:* mov b,#10**

**mul ab**

**mov b,**a ;b=ab=a\*10

**clr a**

**movc a,@a+dptr** ;a=rdCode

**anl a,#0x0f** ;a&0x0f

**add a,b** *;wra🡪 a=* b+ a&0x0f

**mov b,a**

**inc i**

**cjne i,#3,cikl** *;for (i=3; i>0; i--)*

**jmp $ ;**a=0xFB =123АИ

end

Объем программы в памяти Code 30 байт.

**Б)Схема** преобразования в двоичный текстовый ASCII формат с (микро)программным управлением.

**for(i=0;i<8; i++)**

**{ yy[i]= (x&0x01) ? ‘1’ : ‘0’ ; x=x>>1;}** }

7 0 rra

A 0 #YY

#0x30 wrdp

Or incd

dptr

clri

D A Ci i inci

wrx

Xdata

Память

- регистры **A, dptr, i**

- память **Xdata**

**- A=0xFB**

Функциональный элемент OR – поразрядное логическое A[0] v 0x30

Управляющие сигналы **rra**🡪сдвиг А вправо,

**wrx**🡪Xdatа[dptr] запись в память Xdata

(Микро)программа алгоритма в А51

**Xseg** **at 0** ;сегмент данных в Xdata

**Yy: ds 8** ;char xdata yy[8]

i equ r0

**cseg at 0x10**

**part2: clr i**

**сikl2*: mov A,#0xFB***

**mov dpl,#YY**

**anl a,#01**

**orl a,#'0'**

**movx @dptr,a** ;wrx

**mov a,b**

**rr a**

**mov b,a**

**djnz i,cikl2 *;****for(i=7;i>=0; i--)*

**jmp $ ;**”11111011”строка в Xdata

**end**

Размер программы в памяти Code 20 байт .

Итого вся экспериментальная программа занимает 50 байт,что более чем в 2 раза меньше соответствующей программы в С51.

**3)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

}

Объем программы 238 байт в памяти Code, что в два раза больше, чем с прямым доступом.

**Задание**

Выполняется преобразование символических строк в различных типах памяти в С51 и А51. Сравнить требуемые объемы памяти Code в C51. Пояснить причины существенных отличий в объеме программ.

1. Упорядочить текст лексикографически, в порядке возрастания ASCII- кода

“This programmator”  “ aaghimmootTrrs”

1. Вставить пробелы после символа “r”

“This programmator”  ”r”  “This pr ogr ammator”

1. Заменить прописную букву “x” на заглавную в тексте

“This programmator” ”a”  “This progrAmmAtor”

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

“01001001110”  “0x24e”

1. Символьное (в ASCII) преобразование шестнадцатеричного числа в двоичное

“01001001110”  “0x24e”

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

“ 590 ”  “0x24e”

7.Символьное (в ASCII) преобразование шестнадцатеричного числа в десятичное

“590”  “0x24e”

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

“-25,023”  “e+2 - 0.25023”

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

“256,54” +” 75, 56” =” 332,10”

10. Сформировать сдачу минимальным количеством монет достоинством

**50, 10, 5, 1** копеек и проверить обратным преобразованием

“132”  “2 ,3,0, 2”

11. 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

12. 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”

13. 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”

14. 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”

15.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

16 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

**2.2. Ввод-вывод**  данных

**Устройства ввода** цифровых данных c ( **клавиатуры (К)**, цифровых датчиков и др.) преобразуют или считывают представление цифровых данных в ASCII-кодах и выполняется преобразование совмещенных в ASCII-кодах двоично-десятичных кодов в машинные двоичные форматы данных.

**Датчики (сенсоры S)** в управляющих и измерительных вычислительных системах формируют данные в двоичном или двоично-десятичном форматах на

входных портах контроллера. Непрерывное информационное поле, таким образом, дискретизируется в численные данные для обработки в ЭВМ .

**Устройства вывода** (визуализации данных (**дисплеи**(D), индикаторы, принтеры и др)) преобразуют ASCII-коды двоично-десятичных чисел в символьное десятичное изображение.

S

mcu P0

P1

P2 K

P3

M

D

Рис.1.5. Устройства ввода и вывода данных

**2.2.1. Типы данных**

**Численные данные** **(количество)** – основная форма представления информации в математических моделях .

Тип данных определяет множество допустимых значений и применимые к ним операции преобразования. Кодирование различных типов данных

двоичными словами, размещение и доступ к ним в памяти ЭВМ

определяются **форматом**.

1. **Числа с плавающей точкой**

Вязыке **С51** принятыформаты

**float** x= 12345,6789 число с плавающей точкой в стандарте IEEE 754

**double** x= 12345,6789 число с плавающей точкой 64-разрядного формата

К числам с плавающей точкой в С51 применимы операции **(+,-,\*,/ )** и функции стандартной библиотеки  **math.h**

Формат с плавающей точкой называют научным, здесь диапазон практически не ограничен и фиксирована **относительная** погрешность форматом.

Для MCU использование FP-формат связан с большими затратами памяти и временем вычислений.

2) Числа с фиксированной запятой.

Числа различаются как

**целые** – запятая условно фиксирована после младшего бита формата и

**дробные** – запятая фиксирована перед старшим разрядом формата.

Целые можно рассматривать как дробные с масштабом 2n в n-разрядномформате.

Отрицательное число в дополнительном коде (дополнение до двух

**( 2–|x|,**  |x|<1.0**)**  для дробных , n=8-битовый формат с фиксированной точкой, .

Для целых отрицательное число в дополнительном коде ( **28–|x|, |x|<27 )**

7 6 0

Формат целого s +/- |x| (,)

0 1 2 7

Формат дробного s +/- |x|

(,)

**S-** старший знаковый бит двоичного числа (0-положительное число и 1-знак отрицательного числа можно рассматривать как целую часть дробного числа.

**|x| -** семиразрядный модуль числа)

При вычислениях может быть использован “беззнаковый” формат числа, или

двоичный код.

**unsigned char x;**

**unsigned int y;**

Обозначает положительное число – формат числа на один бит больше.

Используя определение формата

**typedef uchar unsigned char;**

**typedef uint unsigned int;**

можно в дальнейшем этот тип многократно использовать**.**

**uint y;**

**uchar x;**

**В С51** к числам применимы элементарные знаковые операции **(+, -, \*, /, %)** и "беззнаковые" **(+,-,\*,/, %,<<,>>),** возможное при этом переполнение формата не контролируется.

К двоичным кодам применимы поразрядные логические операции **(~,&,|,^)**

**В А51** к числамприменимы

**Арифметика 8 битовая знаковая**

Формат числа – двоичное число в дополнительном коде.

**addc a, {Ri,@rj,#d,ad}** ; a + {..}+ C  a

**add a, {Ri,@rj,#d,ad}** ; a + {..}  a, Признаки C,OV,P в PSW

**subb a,** {Ri,@rj,#d,ad} ; a - {..}- C  a

**add a,P2** ; a +P2 a **P2-**регистр порта Р2

**PSW==C.AC.F0.RS1.RS0.OV.-.P** содержитпризнаки результата арифметических операций – **С**(перенос, заем), **AC** – полуперенос из 3-его разряда используется в двоичо-десятичном сложении

**OV**(знаковое переполнение), **P**(бит четности)**,**

**Беззнаковая арифметика**

**inc {a, ri, @rj, ad, dptr} ; {..}+1,** признаки не меняются в PSW

**dec r0, {a, ri, @rj, ad} ;{..}-1**

**mul ab ; a\*bb.a,** признаки **ov=(b#0), 0C, P**

**div ab ; a/b a, b=(a%b)** признаки **ov=(b=0),p**

**rrc a, ; RR(c.a)  (a.C)** признаки **C,P**

**rlc a**, ; **RL(a.C)  (C.a)** признаки **C,P**

**clr a**, ; **0a**

К двоичным кодам применимы поразрядные логические операции

**{anl,orl,xrl} a, {ri,@rj,#d,ad}**

**{anl,orl,xrl} ad,{a,#d}**

**В С51** форматы с фиксированной точкой **целые.**

Числа с фиксированной запятой характеризуются **абсолютной погрешностью усечения =1**  и диапазоном в форматах **char, int, long**.

Целые числа могут быть представлены в разных масштабах с округлением и заданной погрешностью как результат **усечения**  (например, соответствующие значениям в некотором масштабе единиц измерения (расстояния, длины , время, ….).

В А51 можем соответствующие форматы трактовать как дробные, если проще их использовать в вычислениях. Например, в дробном умножении фиксированная погрешность сохраняет форматы сомножителей и произведения, в делении – частное целое.

**3)Двоичные коды** могут использоваться как **символы** в ASCII и форматах текстовой строки

**uchar str[]=”библиотека”;**

В А51 необходимо явно указывать **нуль** в конце строки.

В С51 применимы библиотечные стандартные функции **(cmp, cat, cpy,len,clr,spn,str….)** со строками в библиотеке **string.h**

**4)Cтруктура** определяет древовидную иерархию данных

В структуре могут быть определены элементы различного типа данных (char,int.long,float,..)

Описание модуля MCU.

**struct mcu {** //имя типа структуры

**char name[8];**

**int format;**

**int pin;**

**}MM; //**переменная типа **mcu**

**MM.name ="80c51BH" ;**

**2.2.1. Параллельные цифровые интерфейсы ввода-вывода**.

**В С51 адресный ввод-вывод** представлен адресуемыми цифровыми 8-битовыми портами **P0-P3** в SFR.

Порт адресуется в SFR и включает регистр и схему драйвера, связывающего регистр с контактами. Драйвер осуществляет вывод состояния регистра или передает альтернативные сигналы на контакты и чтение данных с контактов.

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

Порты **P1,P2,P3** в реальных схемах квази-двунаправленные включены как **однонаправленные** и настраиваются на соответствующий режим обмена.

При выводе (**P2=0x55**) константа из памяти Code записывается в регистр порта P2 и передается на внешние контакты соответствующими уровнями

Формат **unsigned** порта может быть представлен как целое или дробное число

7 0 (,)

(,) 1 8

**P2=0x55;** //загрузка (вывод) двоичного кода-константы из памяти Code в регистр P2

**char bb=P1**; //чтение байта с контактов порта P1 с типом **char**

**bb+=P1;**

При вводе данные считываются с контактов порта и сохраняются в памяти, интерпретируются в приложениях в положительном кодировании двоичными кодами (H~1, L~0).

Порт представляет двоичный 8-битовый код и приобретает смысл типа данных

при вводе с записью в конкретный формат (например, число со знаком) или при выполнении операций с конкретными типами данных.

**P1=0x80;** //загрузка (вывод) двоичного кода-константы из памяти Code в регистр P1

**char x=P1;** //x= -128 интерпретируется как знаковое число и читается с контактов порта

**int y=P1**; //y=0-255 в формате **int** интерпретируется как целое unsigned и читается с контактов

**y=x\*P1**; //x-со знаком и порт P1– положительное число 0-255

Операторы **Чтение-модификация–запись**

**P2+=0x55; P3&=0x0f; P1++; P0++;**

**anl P1,#0xAA**

**inc P1**

Символьный вывод данных в файл оператором **sprintf(“ “, mas)**  может быть использован для прямого вывода в алфавитно-цифровой дисплей.

Порты содержат адресуемый в SFR регистр данных, входные и выходные буферные схемы, подключаемые к внешним контактам MCU.

Разряды портов P1,P2,P3 могут быть использованы для ввода либо вывода, если

через них не передаются **альтернативные сигналы Altf:** сигналы чтения /rd и записи /wr внешней памяти Xdata, входные внешние сигналы \int0, \int1 и др.

Если используются альтернативные сигналы, то они однозначно определяют направление обмена.

Altf

P3

Pin 7 0

Рис. 1.4.. Структура квази-двунаправленного порта

На контакте порта **Pin** выход усилителя и сигнал с внешней цепи c общей нагрузкой **Pull-up** резистором, поддерживающим порт в нормальном состоянии H после сброса MCU

Положительный уровень формируется на нагрузочном Pull-up резисторе , допускает малый ток нагрузки для вытекающего тока(sourse)– менее 1 ма.

На контактах порта уровни сигналов (H.L) интерпретируются в положительной логике (1,0).

**P3 7 6 5 4 3 2 1 0** ввод либо вывод

rd wr - - - - txd альтернативные выходные

- - t1 t0 int1 int0 - rxd альтернативные входные

**P2 7.............0** ввод либо вывод

Adress[15..8] выходные альтернативные разряды адреса

**P0 7………..0**

DA Adress[7..0] выходные альтернативные разряды адреса

data [7..0] входные или выходные данные

В **квази-двунаправленных** портах P1,P2,P3 режим работы порта определяет схема включения порта .

Регистр порта – независимый и может быть использован как буферный для временного хранения данных. С другой стороны, состояние регистра по

низкому уровню L(Gnd) не совместимо с высоким уровнем **H** на контакте**,**

так как возможно подключение активного H(+Vcc).По умолчанию, при сбросе регистры Pi устанавливаются для ввода с контактов в единицы.

Системный (внешний) параллельный интерфейс(DA/A[7.0],A[15.0],wr,rd.psen) реализуется через порты по следующей схеме альтернативными сигналами

DA[7..0]/A[7..0]

MCU P0

ale PA

P2

A[15..8]

P3(/R**d,/Wr)** a d port

P3(txd,rxd) XRAM Code

Psen psen

Рис.1.5. Мультиплексированная системная шина

Передача адреса по шине P0 синхронизируется сигналом **ale,** который записывает адрес в регистр адреса РА. Старший байт адреса передается через порт P2. Сигналы управления записью или чтением Xram – через порт P3.

Для управления внешними расширенными портами можно совместить режимы с адресацией внешней памяти Xram по совмещенным адресам и декодировать, например, только старший байт в P2.

При обращении к внешней памяти данных **Xram**  по командам

**movx** a,@rj в MCS51 формируются инверсные сигналы **Rd и Wr, порт P2** используется для выдачи старших разрядов внешней памяти, а **порт P0** для обмена данными и совмещен с младшими разрядами адреса.

При этом исключается одновременный ввод и вывод, что может привести к

конфликту на шине.

Если используется внешняя программная память Code, то в командах **movc** и

при чтении команд формируется сигнал чтения **psen,**

Управление Xram и Code разделены (логически и электрически различимы) и, соответственно, разделены и независимы адресные пространства памяти данных и программ (Гарвардская архитектура)

**2.2.2. Ввод целых чисел с цифровых портов**

Десятичная и двоичная системы счисления позиционные однородные

являются формальным способом записи количества. Методы преобразования

чисел из одной системы в другую изменяют только форму записи целых чисел, в которой сохраняется информация о количестве

Соотношение между числом разрядов двоичного формата (**n)** и десятичного **(m**) –разрядного задается условием **2n>10m** или **n= ]m\*3.33[**

В большинстве компьютеров в системе команд и в Си для простых вычислений используются машинные форматы целых двоичных чисел с фиксированной точкой и округление усечением в арифметических операциях .

Машинные преобразования и вычисления с целыми числами **сохраняют точность(масштаб) M=1**, но ограничены диапазоном, который определяется разрядностью двоичного формата (**n)**.

Если диапазон превышает допустимый, то всегда можно увеличить абсолютную погрешность масштабированием **N/M** и сохранить формат хранения в памяти (М > 1) и перейти в диапазон целых значений с меньшей точностью (большим масштабом).

**А/ Преобразование unsigned целых при вводе c портов**

Форматы ввода двухразрядного целого десятичного числа с 8-разрядного порта (m=8)

a1

a0

7

4

3

0

1001

0101

7

4

3

0

(,) (,)

Рис. 2.6. Формат двоично-десятичного целого числа в порте.

**Преобразование** целых 10/2 могут быть выполнены пересчетом **количества N** в двоичной системе по обобщенной рекуррентной формуле (1.1)

При вводе десятичного числа N=a1a0

**N=B2=( a1 \*10 + a0 )2**

**typedef** **unsigned char uchar;**

**Fd2( uchar x)**{

return (x>>4)\*10 + (x&0x0f); }

**Б/ Преобразование unsigned целых при выводе через порты**

Обратное машинное преобразование 2/10 в двоичной системе могут быть найдены делением двоичного числа на основание 10.

Если **B2=( a1 \*10 + a0 )2**  , то обратное преобразование целая часть **a1** **= B2/10**

в остатке **a0= B2%10** и

**F2d(uchar x)**{

return ((x/10)<<4) | (x%10); }

**2.2. 3. Ввод и вывод unsigned дробных численных данных через порты.**

Схемотехника и алгоритмы выполнения арифметических операций ЭВМ с дробными (**n)**-разрядными форматами проще, чем с целыми.

1. **Ввод дробных чисел**.

Информация (**количество N**) в **записи** (**m)**-разрядного дробного десятичного числа **A=0,a1a2a3…am**

**m-1**

**(2.8) N= ∑ aid-i** = **a110-1 + a-210-2 + ….+ am10-m**

**i=0**

где ai -двоично-десятичная цифра, m-количество разрядов в записи десятичного числа. Запятая фиксирована в машинном формате перед старшим разрядом.

a1 a2 1001 0101

(,)7 3 0 (,) 0.95

a1 a0 (,) 1001 0101 (,) 95,0

Рис. 2.8. Форматы двоично-десятичного числа .

Десятичное дробное число А= **0,a1a2a3…am**  в m-разрядном формате можно рассматривать как целое **A’= am-1am-2..a0** с масштабом 10m.

Тогда, применяя рассмотренный выше метод преобразования целых (2.7), получим (**n)**-разрядное двоичное целое число B2

Для получения дробного двоичного n-разрядного числа необходимо разделить целое A’ на масштаб Q=**B2 /10m в** двоичной системе. В целочисленной арифметике для этого требуется cначала выполнить масштабирование с

двоичным масштабом (**B2\*2n ) и** Q= **B2 /10m**  = (**B2\*2n )/10m** = **Q2\*2n** , чтобы результат деления оказался дробным двоичным в масштабе **2n**

**unsigned int B2;**

**unsigned char b2;**

**main()**

**{**

**{B2=(((P1&0xf0)>>4)\*10 + (P1&0x0f))<<8;** //дробное B2 в масштабе M=**100\*28**

**b2=B2/100;** //двоичное дробное в масштабе **28**

}

Преобразование десятичной дроби в двоичную (n=8) при округлении усечением имеет погрешность **2-8**

В рассмотренной программе ошибку в младшем разряде получаем в двоичном делении на 100 и округлении усечением.

Соотношение между числом разрядов двоичного формата **(n)** и десятичного **(m)** при условии сохранения точности при переводе с усечением **n>=]m3.3+1[ .**

**Б) Вывод дробных чисел**

Если известно двоичное **Q2=N=Sn=0,a1a2a3…am**, то из (2.8) неизвестная двоично-десятичная цифра a1 = N\*10 - целая часть двоичного произведения

**Qn-1=a1, a2a3..am**

Последняя цифра в этом преобразовании 2/10 может иметь погрешность не более **10-m** .

Преобразования дробных чисел из одной системы счисления в другую в фиксированных форматах в общем случае приближенные.

Умножая двоичное целое в масштабе 2n на основание 10, получим **a1** в (**n..n+3)** разрядах целого произведения.

В произведении 10\* (**B2\*2n** )= **a1, a2a3..am\*2n** сохраняется масштаб **2n** и положение запятой.

**{ uint y;**

**y=b2 \*10;**

**P2=((y&0xf00)>>4);** // **a1**- старшая десятичная цифра

**P2 | =(((y&0xff)\*10)&0xf00)>>8;** } //a2 цифра

7 0 q1 q7

a1

(,)

**Задание.**

Разработать программу ввода и вывода целых чисел со знаком в С51 и А51 через порты и ввода дробных в С51 в формате

+/- a2 a1 a0 формат десятичного

**2.3. Арифметические операции с фиксированной точкой**

Основные машинные двоичные арифметические операции: сложение, вычитание, умножение и деление

Операция **двоичного сложения(вычитания) – элементарная арифметическая операция** выполняется схемой **арифметико-логического устройства (ALU)** ЭВМ.

Операции **умножения и деления** двоичных чисел в MCU могут выполняться

программами в соответствии с известными алгоритмами, в которых

применяются элементные операции ALU сложения и вычитания.

Предлагается познакомиться с практическим использованием некоторых алгоритмов в программах умножения, деления и квадратного корня.

Реализация в С51 имеет смысл для демонстрации и отладки алгоритма.

Программа в Ассемблере имеет отношение к аппаратной реализации и позволяет показать исполнение алгоритма в программной модели ассемблера ЭВМ.

При этом используется **микропрограммная модель ЭВМ ( структурная схема) ,** которая определяет форматы используемых регистров, организацию доступа к памяти и управляемые элементарные операции(микрокоманды).

**2.3.1. Сложение и вычитание двоичных чисел**

Основные свойства операции сложения в фиксированных форматах с фиксированной запятой без знака

**unsigned char x,y,z;**

**z=x+y;** ограничение формата вызывает переполнение, если

z>28-1 для целых или z > 1.0-2-8 для дробных. Переполнение контролируется признаком переноса С=1 в PSW.

**char x,y,с;**

отрицательные числа в дополнительных кодах **(2-|с|)**  для дробных или

**(28-|с|)** для целых. В 8-разрядных форматах эквивалентно 0-|с|, |с|≤27 для целых и |с|≤1. Cтарший бит двоичного кода совпадает со знаком.

Признаки результата при сложении

**x,y >0 z=x+y ,** переполнение , OV&**ך**C в PSW

**x≥0,y<0 z=x+2-y=2-(y-x)<0 , (y<x),если (ך**OV)&С

**x<0,y≥0 z=2-x+y=2-(y-x)<0, (x<y),если (ך**OV)&С

**x<0,y<0 z=2-x+2-y=2-(x+y) ,** переполнение, если OV& С

2.3.2. **Умножение**

**1)Умножение двоичных дробных чисел в С51**

Как следует из раздела 1.2, **двоичные коды** **(n)**-разрядных дробных чисел и соответствующих целых в масштабе **2n** совпадают. Следовательно, двоичные

коды дробных произведений в масштабе **22n** также совпадают с

целыми произведениями и алгоритмы дробных произведений применимы к целым – возможно при записи целых и дробных

чисел в форматах одинаковой размерности различная нумерация разрядов.

Форматы дробных чисел с фиксированной точкой в mcs51

**A \* B 🡪 S**

(,) 1 8 (,) 1 8 (,) 1 16

Приведение полиномиальной формы представления чисел к

итерационной для вычисления произведения

**а)Вычисление с общим членом ряда** со стороны старших разрядов множителя

**S=A\*B=A\*(B=0.b1b2..bn ) =A\*(b12-1 + b22-2 .. +bn-12-n+1+bn2-n)=**

**= Ab12-1 + Ab22-2.. +Abn-12-n+1 + Abn2-n = ∑ Ribi**

(2.1) S= **∑ Ribi ;Ri+1 = Ri\*2-1;i=1,2,..8; R1=A**

Схема вычисления в С51

1 16

add **(,) S** wrs

**+ 1 16** wra,shra

**R**

**(,)**

**1 8 3 0**

P0   **i** wri,inc

**1 bi 8 #8**

**B shlb**

**wrb**

P1

Рис.2.1.Схема умножения с общим членом ряда для дробных чисел

Если bi=1, то S=S+ Rbi - разрешение записи (**wrs**). Регистр А сдвигается вправо-разрешение **(shrа)**

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

Программа в С51

**#include <reg51.h>**

**сhar i; //** счетчик шагов,

**unsigned int R,S;**

**unsigned char B;**

**main()**

**{**

**while(1)**

**{ R=(P0<<8);** //ввод и выравнивание форматов

**B=P1;**

**for(i=8; i>0; i--)**

**{S= (B&0x80)? S+(R>>1) : R>>1; B<<1;}**

**P2^=0x80;}** //временная метка

**P2=S>>8;**

**P3=S; //**вывод младшего байта произведения

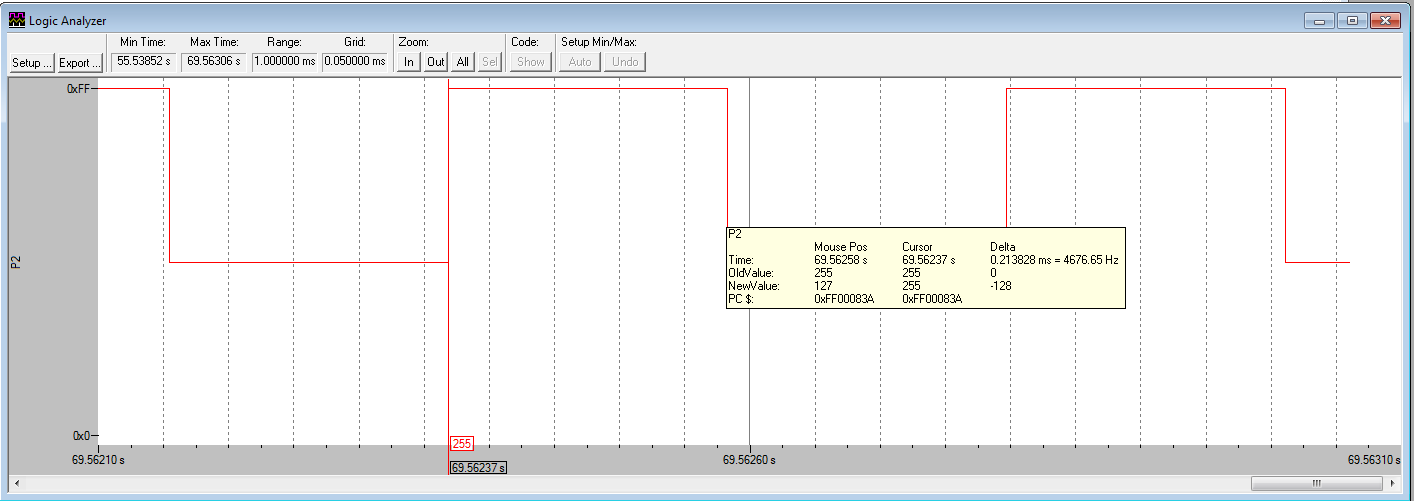
}

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

Измерение времени выполнения операции.

Для измерения времени можно использовать функцию **Analizer** в Кейл [Приложение 2].

Командой **LA P2** в командном режиме (или в Setup в Анализаторе) значение P2 передается в окно Анализатора. Временная диаграмма временной метки в окне Анализатора.



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

**б) Вычисление по схеме Горнера** со стороны младших разрядов множителя

**S=2-1  \*(Ab1 + 2-1(Ab2 +… +\_2-1(Abn-1 + 2-1(Abn +0))..)=>**

**S0=0 => S1=2-1(S0+Abn) S2= (S1+Abn-1)2-1**

**(2.2) Si+1=2-1(Si +Abn-i) , S0 =0, i=0,..n-1, B ={ b1,b2,., bn-i ,..,bn}**

Рекурсивная функция **S(i+1)=(S(i) + Abn-i)2-1, S(0)=0**.Схема вычисления.

**P0**

bi

31 01 8 9 16 shrs

**(,) S (,) 1 S/B 8** wrs

**+ 16** wra

**A**

**(,)**

**1 8**

**3 0**

P1 **i** wri,inc

**#8**

Рис. 2.2. **У**множение дробных чисел по схеме Горнера

Множитель B размещается в младших 8 разрядах регистра-произведения S, младший разряд B[16] в дробном формате при сдвиге вправо **(shrs)** сохраняет текущее значение **B(8)**=**bi**

При суммировании в С51 возникает неконтролируемое переполнение - перенос С. По этой причине используется удвоенный 32-битный формат S.

16-битовый формат A28 согласован с положением запятой в формате 16-битового произведения .

Суммирование выполняется в [15-8]-разрядах формата регистра S частичных произведений .

**#include <reg51.h>**

**сhar i; //** счетчик шагов,

**long S;**

**unsigned int A;**

**main()**

**{**

**A=P0<<8;** //ввод и выравнивание форматов

**S=P1;**

**for(i=8; i>0; i--)**

**{S= (S&1)? (S+A)>>1) : S>>1; }**

**P2=S>>8;**

**P3=S; //**вывод младшего байта произведения

**while(1);**

}

При умножении дробных **(n)**-разрядных чисел с фиксированной запятой 2n-разрядное произведение имеет абсолютную погрешность

**2(-2n) <=  < 2((-n)-1)**

Максимальное произведение дробных n-разрядных чисел меньше любого сомножителя и не превышает 2n-разрядный формат. Например, при одном минимальном сомножителе **2-n** и другом максимальном (1-**2-n) 2-n=2-n- 2-2n < 2-n**

При этом для дробных можно ограничиться вычислением и сохранением

только старших (**n)** разрядов произведения.(погрешность **= 2(-n)),** если в дальнейшем не используется деление дробных.

Для **умножения целых чисел**  можно использовать те же схемы и алгоритмы, если интерпретировать множитель как дробное с масштабом 2n**.**

**S=A\*B=A\*(0,B)\*2n= (0,A)\*(B)\*2n**

Например, для формулы (2.1)

**S=A\*B=A\*(B=bn-1bn-2..b0 ) = A\*(b12n-1 + b22n-2 .. +bn-121+bn20) =**

**A\*(b12-1 + b22-2 .. +bn-12-n+1+bn2-n)2n=**

**= (Ab12-1 + Ab22-2.. +Abn-12-n+1 + Abn2-n )2n = (∑ Ribi)2n**

Умножение целых с общим членом ряда со стороны младших разрядов

S= **A\*(bn=12n-1 + bn-22n-2 .. +b121+b020) =**

**= Abn-12n-1 + Abn-22n-2 .. +Ab121+Ab020 = ∑ Ribi**

**(2.3)** S= **∑ Ribi ;Ri+1=Ri\*2, i=1,2,..8; R1=Ab0**

**Схема вычисления в С51**

1 16

**S** wrs

**+ 1 16** wra,shla

**A**

**(,)**

**1 8 3 0**

P0 bi  **i** wri,inc

**1 bi 8 #8**

**B shrb**

**wrb**

P1

**#include <reg51.h>**

**сhar i; //** счетчик шагов,

**unsigned int A,S;**

**unsigned char B;**

**main()**

**{**

**A=P0;** //ввод и выравнивание форматов

**B=P1;**

**for(i=8; i>0; i--)**

**{S= (B&1)? (S<<1)+A : A<<1; B>>1;**

**}**

**P2=S>>8;**

**P3=S; //**вывод младшего байта произведения

}

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

Схему и программу с масштабированием множителя можно применить и к дробным числам.

1. **Микропрограммное умножение в А51.**

Размещение операндов в команде **mul ab : B-**множитель**, Aa-**обозначает аккумулятор **A**

Требуется вспомогательный рабочий регистр **wrk** для записи множимого.

Если использовать для вычислений схему (2.2), то она оказывается наиболее простой с микропрограммным управлением.

clra

rrca,wrb Bi

**7 0 7 0**  wrp1,wra

**С Aa C B**

**xch 3 0**

**+ 7 0 P1 i inci**

**wrk** wrp0 **Ci**

**add P0 wr( 8)**

Рис.2.3. Схема умножения с микропрограммным управлением

**(+)**сумматор**, i**-счетчик тактов.

Управляющие микрокоманды: **clra**🡪Aa=0,  **wrp1**🡪B=P1, **wrp0**🡪wrk=P0,**wr(8)🡪**i=8, **inci🡪** инкремент счетчика. При сдвиге **rrca** признак С записывается в старший разряд Aa, а млаший разряд Aa запоминается в С.

**Ci**-перенос со счетчика~(i==0).

**xch={wrb,wra}**🡪{Aa=B, B=Aa} в этой микрооперации совмещены две записи по разным шинам.

В (микро)программе A51 для тестирования умножения 8\*8 команды по смыслу совпадают с управляющими микрокомандами.

**i equ r0**

**wrk equ r1**

**sbit Bi=B^0 ;**определение бита SFR

**cseg at 0**

**start: mov wrk,P0**

**clr a**

**mov B,P1**

**mov i,#8**

;S= (S&1)? (S+A)>>1 : S>>1;

**cikl: jnb Bi,m1**

**add a,wrk**

**m1: rrc a**

**xch a,B**

**rrc a**

**clr c**

**xch a,B**

**djnz i,cikl ;**for(i=8,i>0;i--)

**mov P2,a**

**mov P3,B**

**nop**

**end**

Объем программы Code=32 байта, время выполнения программы 90 мкс

1. **Функциональный элемент параллельного умножения**

Матрица умножения S=A\*B, где используются 4-разрядные множимое A и множитель B, 8-разрялное произведение целых со стороны младших разрядов множителя вычисляем по формуле (2.3). Вычисления эквивалентны

традиционной записи выполнения операции S[8..1]=A[4..1]\*B[4..1] таблицей

A4A3A2A1

1 0 1 0 b1 =1

1 0 1 0 b2 =1

1 0 1 0 b3=1

1 0 1 0 b4=1

1 0 0 1 0 1 1 0

s8 s7 s6 s5 s4 s3 s2 s1

Последовательная схема при ручных вычислениях может быть реализована в однoтактной параллельной схеме.

A[8..1] b8 b3b2b1

0

b1

9 S[9..1] 1

b2 S[9..1]

9 1

b3 9 1

S[9..1]

b8 S3

**S7**

**9 1**

S[9..1]

**S [ 16 15 … 10 9 8 7 … 3 2 1]**

Рис. Матрица умножения 8\*8 🡪16

**S[9..1]** – девятиразрядный сумматор, 9-ый бит формирует перенос в следующий разряд.

Разряды множителя b8-b1управляют вентилями, которые подключают множимое А[8..1] к входам сумматора.

**4)Знаковое умножение в С51**

В mcs51 аппаратно реализовано умножение mulab по общему алгоритму для положительных двоичных чисел. Для умножения отрицательных чисел необходимо создать программу на ассемблере, где могут быть учтены форматы кодирования со знаком.

**А)**Простой способ учета знака - с преобразованием отрицательных сомножителей в прямой код и определением знака произведения сравнением знаков сомножителей.

**B)**Известны методы автоматического учета знаков (например, метод Бута)

**C)**Если сомножители рассматривать как дробные и A<1 и B<1 – модули чисел, то:

- для положительных сомножителей S= A\*B=AB , для максимальных A=B=1-2-n произведение AB < A,B

- при разных знаках сомножителей S= A\*(2-B)= 2 A – AB. Требуется коррекция результата,

- для отрицательных сомножителей

**S=(2-A)\*(2-B)=4 -2A -2B +AB= -2(A+B)+AB .** Требуется коррекция результата.

**D)** Метод автоматической коррекции из библиотеки С51 для 8-разрядных целых чисел

Программа преобразования числа **char** в формат **int** с кодом знака {0,-1}

**Mov r7,x ;x<0= (**2n  - x)=0x9C, x=-63

**MOV A,R7**

**RLC A ;знак в С**

**SUBB A,ACC ; A-A-C={0,-1}**

**MOV R6,A ; r6,r7=(-1).x=sx.x =0xff9C**  -63 в формате **int**

Умножениев масштабах с целыми числами (**x,y-**модули**, s-**знаки**)**

**(sx.x)\*(sy.y)= sx\*y\*2n+x\*y+sy\*x\*2n +sx\*sy\*22n**

**= (-2n) \*(2n-y)\*2n+(2n-x)\* (2n-y)+(-2n) \*(2n-x)\*2n**

В формате **2n = y\*2n + (-x\*2n - y\*2n +xy)+ x\*2n = xy**

При этом сумма всех единиц в целой части компенсируется и результат прямого умножения **xy** положительный **.**

Во всех операциях умножения используется команда **mul ab**

**2.3.3.Деление.**

1. **Программа в С51.**

Если рассматривать деление дробных чисел B=S/A как обратную операцию для умножения S=A\*B, то делимое S должно быть меньше любого сомножителя как делителя.

Делимое предполагается в удвоенном формате – запятая фиксирована перед старшим(левым) байтом. На последнем шаге вычисления (рекуррентная формула 2.2) дробного произведения

**S=A\*B=Sn=2-1(Sn-1+Abi)** и делимое S меньше делителяA.

Тогда цифра частного bi =1 при условии, что  **Sn-1 = 2Sn-A >=0**, иначе bi =0.

Изменяя нумерацию остатков Si+1 при вычитании **2Si-A** , приходим к следующей рекуррентной формуле деления

(**2.3) Si+1= 2Si-A и bi=1 , если 2Si-A >=0, где S0=S-делимое**

**Si+1= 2Si и bi=0 , если 2Si-A <0,**

Схема деления

P0 P1

bi

31 01 8 9 16 shls , incS

**(,) S (,) 1 S/B 8** wrs

**-- 16** wra

**A**

**(,)**

**1 8**

**3 i 0**

P2wri,inc

**#8**

Рис.2.5. Схема деления дробных.

Делимое **long S**, 8-разрядный делитель **A**

Регистр **B** -частное совмещено с младшими разрядами делимого при сдвиге влево **(shls)** . Сигнал **wrs 🡪 S=(P0<<8)|P1, incs🡪**S+1**,** если разность при вычитании положительная.

Первое вычитание контролирует переполнение, если (**S>=A).** Тогда целая часть b0=1иформируется признак переполнения OV=1 и деление завершается.

Расширен формат делителя 16 бит и формат делимого 32 бит для сохранения битов при сдвиге влево при фиксированном расположении запятой.

Программа деления в С51.

**#include <reg51.h>**

**main()**

**{ unsigned int Aa;**

**long S;**

**char i;**

**S=(P0<<8)|P1; //**делимое

**Aa=P2<<8 ; //**делитель

**if ((S-Aa)>=0) ? {OV=1; goto out;}**

**for (i=0 ; i<8; i++ )**

**S= (((S<<1)-Aa)>=0) ? (S<<1)-Aa +1 : S<<1 ;**

**out:**

**P3=S; //**частное, OV=Psw[2]

**}**

1. **Микропрограмма в А51**

**Схема деления 16/8 в mcs51:**

**wrp0,wrb**

**P0 rlcA P1**

**7 0 7 0 wra,wrp1**

**С Aa c B**

**xch**

**S +/- \_ 7 0 wrp2 3 0 inci**

**add wrk Ci i**

**subb P2 wri**

**ov setbov F0 setbf0 #8**

**wrk**-делитель

**Aa.B**-делимое

**B**-частное

**OV**-признак переполнения, **F0**-бит пользователя в PSW

**S**-сумматор/вычитатель **add🡪**wrk+Aa,**subb**🡪Aa-wrk

Первое вычитание контролирует переполнение **(Aa>=wrk)~C=0**, деление завершается, признак переполнения **setbov**🡪OV=1.

Сигналы управления **wrP0** 🡪Aa=P0**,**  **wrP1**🡪B=P1, **rlcA**🡪циклический сдвиг влево (c.A**), wri🡪**запись i=8,

**add🡪**wrk+Aa,**subb**🡪Aa-wrk,

**xch**🡪Aa🡨🡪B в этой микрооперации совмещены две записи по разным шинам

Aa=B и B=Aa.

Микропрограмма деления в А51.

**i equ r0**

**wrk equ r1**

**cseg at 0**

**mov A,P0 ;**делимое-старший байт

**mov B,P1 ;**делимое-младший байт или частное

**mov wrk,P2** ;делитель

**mov i,#10**

**subb A,wrk ;**if(A>=S) {OV=1; goto out;} ;пробное вычитание

**jc m1**

**setb ov**

**jmp out**

**m1: add a,wrk** ;восстановление остатка

**clr c**

**cikl: djnz i,m2** ;for (i=0 ; i<8; i++ )

;A= (((A<<1)-S)>=0) ? (A<<1)-S +1 : B<<1 **;**

**m2: xch a,B**

**rlc A**

**xch a,B**

**rlc A**

**mov F0,C** ;сохранение бита С равного биту частного, разность >0

**clr c**

**subb a,wrk ; F0 C bit** таблица истинности

**; 0 0 1**

**; 0 1 0 ;**восстановление остатка

**; 1 0 1**

**; 1 1 1**

**anl C,/F0**

**jnc m1 ;**бит частного 0 **- ;**восстановление остатка

**setb c**

**jmp cikl ;**бит частного 1

**out: mov P3,B ;частное**

**sjmp $**

**end**

В ассемблере совмещаются схемы деления и умножения. В частности, для сдвига 16-разрядного произведения и делимого используются команды **rrc a, rlc a** и команды **xch A,B,** гдеобмениваетсясодержимое регистров (A\*B🡪B.A и (A🡨🡪B) )

1. **Функциональный элемент деления**

В целом делении формула (2.3) трактуется как вычисление разности

**Si+1= 2Si-A** из предположения, что **bi=1 , если 2Si-A >=0** . Признак положительной разности – инверсия **заема** при вычитании в старшем разряде или **перенос** **true** при суммировании с дополнительным кодом делителя.

Если **Si+1= 2Si-A** <0, то предполагается восстановление положительного остатка (**2Si-A)+A**  и затем вычитание для получения следующего остатка

2((**2Si-A)+A)-A=2(**(**2Si-A))+A.**

Следовательно, при отрицательном остатке суммирование множимого и отрицательного остатка на следующем шаге эквивалентно восстановлению остатка и повторному вычитанию . Признак при суммировании с дополнительным кодом остатка сохраняется – перенос в старшем разряде равен **true**.

Традиционная запись выполнения операции деления S[8..1]/A[4..1]=B[4..1] таблицей

s8 s7 s6 s5 s4 s3 s2 s1

S 1 0 0 1 0 1 1 0

(-) 1 0 1 0 A

b4 1 0 0 0 1

(-) 1 0 1 0 A

b3 0 1 1 1 1

(-) 1 0 1 0 A

b2 0 1 0 1 0

(-) 1 0 1 0 A

B1 0 0 0 0

A[4..1]=1010

B[4..1]=1111

Матрица деления, если в таблице фиксировать размещение делителя – тогда влево сдвигается остаток или сумма.

**A[8..1] B8** S[8..1] B7

S1

B7 S[8..1] B6

**S2**

**B3** S[8..1] B2

S6

B2 S[8..1] B1

**S7**

B1S[8..1] **0**

**S16 s15 …. S10 s9 s8 s7 s6 s5 … s2 s1**

Рис. Матрица деления 16/8🡪8

**S[8..1]** – девятиразрядный вычитатель/сумматор (Bi-цифра частного, управление операцией в S (Bi=0-разность S-A,Bi= 1-сумма S+A))

B1- можно рассматривать как признак переполнения или добавить еще каскад для вычисления 9-разрядного частного

Форматы **целых** 8-разрядных делимого А и делителя В в mcs51 можно представить дробными в форматах

(,) 0 А

(,) В

В этих форматах применим метод деления дробных чисел, дробное частное А/B в масштабе 2-8 совпадает с целым. При этом пробное вычитание не требуется – на этом шаге переполнение возможно только при B=0.

**Задание.**

**Совместить программы последовательного умножения и деления по схеме Горнера и с общим членом ряда в С51. Выполнить программы c общим членом ряда в С51 и А51 . Измерить время исполнения.**

**3) Программа деления чисел со знаком из библиотеки в С51.**

При этом для выполнения деления можно использовать удвоенный формат делимого по сравнению с делителем.

В библиотеке С51 используется следующий метод нормализации при выполнении целого деления 16-разрядного формата на 16-разрядный.

Диаграмма выполнения деления – исходные операнды размещаются в регистрах S=r6.r7-делимое, в регистрах A=r4.r5 –делитель

Нормализация выполняется, если S**>A ((r6!=0)&(r4=0))** целым делением

div ab и затем дробным делением побитно формируется младший байт частного. Если (r6.r4 = 0.0), то выполняется команда целого деления div ab.

Если (r4!=0), то частное определяется дробным делением.

r6.r7 / r4.r5 🡪 r6.r7

преобразование

отрицательных

чисел в прямой код

контроль значений

r6, r4

r6.r4= 0.0 (r4!=0) (r6!=0)(r4=0)

A/B=r7/r5 (0.r6.r7)/(r4.r5.0) A/B=r6/r5

Ov 1

0 (r6 % r5.r7)/r5

Рис 2.5. Знаковое целое деление в С51

**2.3.4.** И**звлечение квадратного корня**

Обычно квадратный корень как функция **sqr( float x)** включается в стандартную библиотеку с плавающей точкой. Однако известны случаи, где в

систему команд ЭВМ включается специальная команда быстрого вычисления квадратного корня из дробного числа.

Формат подкоренного числа 16 бит. Результат занимает 8-разрядный формат

Алгоритм извлечения для дробного двоичного числа 0.B=√0.S0.

Пусть i+1-ое приближенное двоичное значение корня xi+1= xi bi+1 и bi+1 -младшая двоичная цифра в этом приближении , S0-дробное подкоренное

значение не равно 0, x0=0-начальное -целое значение дробного корня, bi+1- текущая двоичная цифра корня.

На первом шаге **S0>=(x0 +0.b1)2 =(x0 +0.1)2 =x02 +x0+0.01, x1 =0.b1 и b1=1-старшая цифра дробного корня, если**

**S1=S0-0.01 >=0**

Пусть **b1=1 и S1>=0,** тогда на втором шаге сдвинем остаток **S1** на 2 разряда влево и значение корня **x1** на один разряд влево – обозначим новое значение этого остатка Q1.S1, где Q1-целая часть и S1-дробная часть и **x1= b1.**

Предположим **x2= b1,b2**

Q1.S1>=(x1 + 0.b2)2 = **x12 +x1+0.01 и b2=1, если**

**Q2.S2=4Q1.S1-x1.01=x12 >=0**

Пусть **b2=1 и Q2.S2>=0,** тогда на третьем шаге сдвинем остаток на 2 разряда влево – получим Q3.S3, где Q3-целая часть и S3-дробная часть и корень сдвинем на один разряд влево **x2= b1,b2**  и получим **x3= b1b2,b3**

Q3.S3>=(x2 + 0.b3)2 = **x22 +x2+0.01 и b3=1, если**

**Q3.S3=4Q2.S2-x2.01=x22 >=0**

Рекуррентные формулы для вычисления корня методом “цифра за цифрой” без восстановления остатка

**(2.5) Qi+1.Si+1=4Qi.Si-xi.01=xi2 и bi+1=1, если 4Qi.Si-xi.01 >=0**

**Qi+1.Si+1=4Qi.Si=xi2 и bi+1=0, если 4Qi.Si-xi.01 <0**

Q

(сдвиг влево на 2 разряда)

1

n

Q

1

n

Q

1

n

Shl2

xi

01

**bi**

Shl (сдвиг влево на 1 разряд)

)

Рис. 2.7. Схема извлечения корня квадратного без восстановления остатка.

**2.4. Вычисление функций**

Распространенные функции вычисляются по формулам разложения в ряд Тейлора в диапазоне дробного аргумента **0-1.0**.

**2.4.1. Вычисление функции с плавающей точкой(FP).**

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 и переменной абсолютной .

Однако FP-формат не всегда приемлем в вычислениях и проектировании ЭВМ:

- сложные алгоритмы преобразования и схемотехника, значительные затраты времени при вычислениях

-постоянная относительная погрешность, зависящая только от разрядности мантиссы **∂ = 2(-n) (**двоичная (n)-разрядная мантисса, округление усечением) и переменная абсолютная

С фиксированной точкой программа вычислений проще и время вычислений существенно сокращается, что можно показать в следующих лабораторных работах .

***Пример*.**

Используя библиотечную функцию из библиотеки **math.h** языка С51, вычислить значения sin(x) в диапазоне аргумента 0-360о (2π радиан). При компиляции в Кейл записать параметры программы – объем требуемой памяти данных и объем программы.

В Логическом Анализаторе измерить среднее время вычисления функции.

Схема вывода значений функции через порт. - псевдо 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 28

Р3

Sin(x)

ЦАП

Окно Анализатора

Рис.2.9. Схема работы Анализатора

При выводе через порт P3 учитывается, что значение должно быть целым, положительным и не более **28**.

Для измерения времени вычисления sin(x) без учета промежуточных преобразований float в целые можно использовать вывод метки времени в порт P2.

**Реальное время** вычислений контролируется Симулятором и синхронизировано частотой работы компьютера.

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

#include <reg51.h>

#include <math.h>

float x,y;

char i;

**main()**

{

**i=0;**

**for(x=0; x<6.28 ;x+=0.0628)**

**{**

**P2=0; /**/метка для измерения времени

**y= sin(x); P2=0x80;**

**P3=y\*100+100; /\***масштабное преобразование переводит дробное в целое с точностью 2 знака после запятой и смещает в положительную область значений для вывода через порт\*/

}

Время автоматически измеряется симулятором и выводится на экране или как на осциллографе вручную.

Для измерения времени можно использовать функцию **Analizer** в Кейл.

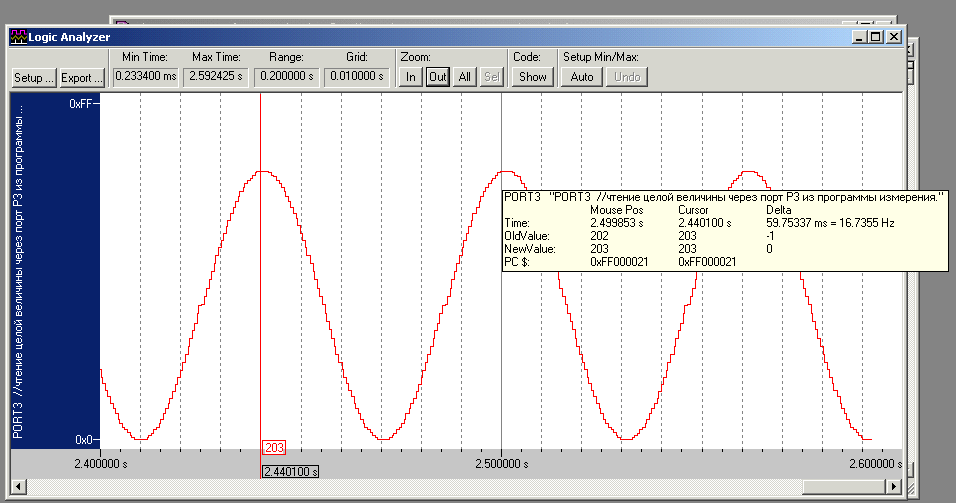
Командой **LA P2** в командном режиме значение P2 передается при выполнении программы в окно Анализатора. Временная диаграмма временной метки в окне Анализатора.

P2=0x80 P2=0

t=3.7 ms t

График функции, вычисляемой с плавающей точкой, в окне Анализатора.

В версии KeilArm (2012 г) Program Analizer позволяет определять автоматически минимальное, максимальное и среднее время без использования временных меток.

****

2.8. График функции в окне Анализатора.

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

**2.4.2. Вычисление функции с фиксированной точкой в целых числах и выбор масштабов**.

Вычисления с фиксированной точкой позволяют существенно сократить время вычислений и объем прграмм, если операнды имеют ограниченную область значений (например, только дробные), во встроенных микроэвм основной машинный формат целый или дробный.

Функции как и в библиотеке match.c для FP, представлены рядами Тейлора.

Некоторые приближения реально не применяются для вычислений и имеют смысл как учебные для демонстрации применения рекурсивных вычислений.

Например, 1/(1+x) включают две простые операции, но в приближениях существует также полиномиальная формула .

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

(2.8) **sinx ~ x/1 – x3/3! + x5/5! – x7/7! + при всех х<1**

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

1. **Вычисления по схеме Горнера**

Преобразование по схеме Горнера

**sinx ~ x/1 – x3/3! + x5/5! – x7/7! = x(1-x2/6(1-x2/20(1-x2/42)) 🡪**

**S1= 1-x2/a0\*S0 , a0 = 42, S0 =1**

**🡪 S2= 1-x2/a1\*S1🡪 , a1=20**

…………………………..

**🡪Si+1= 1-x2/ai\*Si**  , S0=1; i=0,1,

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

**Si+1= 1-x2/ai\*Si**  🡪 **Si+1=(m –(( (x2/m)\*m)/ai \*Si )/m )**

**A)Масштаб m=100**

typedef **unsigned char uchar;**

**uchar x,y, S,m;**

**uchar Si(uchar ai)**

**{ return S=m –((y\*m)/ai \*S)/m ;}**

**main()**

**{ m=100;**

**while(1)**

**for(x=0; x<m; x++)**

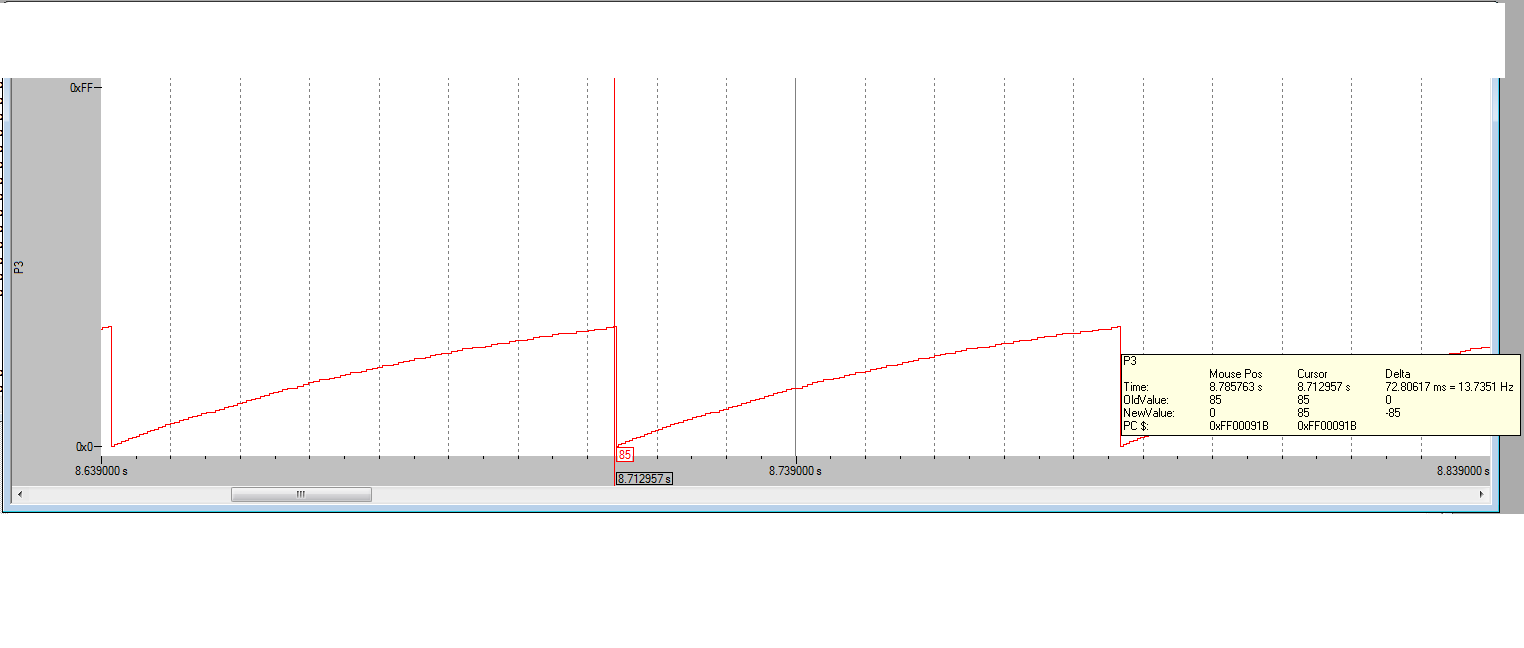
**{ y=(x\*x)/m; S=m;**

**S=Si( 42) ;**

**S= Si(20)  ;**

**S =Si(6);**

**P2=S\*x/m;}**

****

x – значение аргумента в диапазоне [0 – 100], от 0 до 1.0 радиан.

Объем программы 320 байт, среднее время 0.72мс

**Б)Вычисления в целых с двоичным масштабом m=28**

typedef **unsigned char uchar;**

**uchar x,y, S,m;**

**uchar Si(uchar ai)**

**{ return S=m –((y/ai \*S)>>8) ;}**

**main()**

**{ m=0xff;**

**while(1)**

**for(x=0; x<m; x++)**

**{ y=(x\*x)>>8; S=m;**

**S=Si( 42) ;**

**S= Si(20)  ;**

**S =Si(6);**

**P2=(S\*x)>>8; } }**

При вычитании в формате байта используем значение единицы 0xff в масштабе 28. Если вычитаемое не равно нулю, можно добавить единицу для повышения точности.

Объем программы 97 байт, среднее время вычисления 0.11мс

**Преобразование программы в А51**

Для отладки программы используем размещение переменных в памяти Data.

Затем для сокращения объема программы можно заменить переменные в Data регистрами.

ai equ r3

x equ r0 ;требуется регистр в команде cjne

S equ r1

y equ r2

cseg at 0x0 ; начало программы в сегменте Code

jmp start

**Si:** **;S=m –((y/ai \*S)>>8)**

mov a,y

mov b,ai

div ab ;y/ai

mov a,S

mul ab

mov a,#0xff

subb a,b ;S=a

mov S,a

ret

**start:** mov x,#0

**cikl:** ;y=(x\*x)>>8

mov a,x

mov b,x

mul ab

mov y,b

mov ai,#42

mov S,#0xff

call Si

mov ai,#20

call Si

mov ai,#6

call Si

mov b,x

mov a.S

mul ab

mov P2,b

inc x

cjne x,#0xff,cikl

jmp start

nop

**end**

Объем программы **51 байта**, среднее время вычисления функции **0.08 мс**

1. **Вычисления c общим членом ряда**

**sinx ~ x/1 – x3/3! + x5/5! – x7/7! = ∑Ri 🡪**

**R0=(-1)0x 🡪**

**R1=R0(-1)1x2/6 🡪**

**R2=R1(-1)0x2/20 🡪**

**R3=R2(-1)1x2/42 🡪**

**……………..**

**Ri+1=Ri(-1)ix2/(2i(2i+1))**

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

**Ri+1=Ri(-1)ix2/(2i(2i+1)) 🡪 Ri+1=(Ri(-1)ix2/m)/m/(2i(2i+1))**

**A)Масштаб m=100**

**#include <reg51.h>**

**typedef unsigned char uchar;**

**uchar i,x,y,R,m,n;**

**char S;**

**uchar Si()**

**{ return R=((R\*y)/m)/(i<<1)/((i<<1)+1) ;}**

**main()**

**{ m=100;**

**while(1)**

**for(x=0; x<m; x++)**

**{y=(x\*x)/m; S=x; n=4; R=x;**

**for(i=1;i<n;i++)**

**S= (i%2)? S-Si() : S+Si();**

**P2=S;**

**}**

**}**

**Б)Вычисления с двоичным масштабом m=28**

typedef **unsigned char uchar;**

**uchar i,x,y, S,m;**

**uchar Si(uchar ai)**

**{ return R=(R\*y)>>8)/(i<<1)/((i<<1)+1)) ;}**

**main()**

**{ m=0xff;**

**while(1)**

**for(x=0; x<m; x++)**

**{y=(x\*x)>>8; S=m; n=4;**

**for(i=1;i<n;i++)**

**S= (i%2)? S-Si : S+Si ;**

**P2=S;**

**}**

**}**

**Выполнить в С51 вычисления с плавающей и фиксированной точкой по схеме Горнера и с общим членом ряда с** масштабами m=100 и **m=28 Измерить объемы программ и время исполнения.**

**С общим членом ряда выполнить программу с масштабом m=28 в А51, показать зависимость положительной погрешности вычислений от числа членов ряда.**

1. (1+x)/((1-x)2) ~ 1/2 + x + x2 + x3 +

2. 1/(1+x) ~ 1 – x + x2  - x3 +

3. x0.5 ~ x/2 – x2/(2\*4) + 1\*3\*x3/(2\*4\*6) -1\*3\*5\*x4/(2\*4\*6\*9)

4. ax ~ 1 + (lna)\*x + (lna)2 x2/2! + (lna)3 x3/3! +

a=1/2

5. cos(x) ~ 1 – x2/2! + x4/4! – x6/6! +

6. tgx ~ x + x3/3 +2x5/15 + 17x7/315 +62x9/2835

7. ctgx ~ 1/x – (x/3 +x3/45 + 2x5/945 + 2x7/4725 + …)

8. ln(1+x) ~ x – x2/2 + x3/3 –x4/4 + x5/5 +

9. arcsin(x) ~ x + x3/(2\*3) + 1\*3\*x5/(2\*4\*5) + 1\*3\*5x7/(2\*4\*6\*7) +

10. arctg(x) ~ x – x3/3 + x5/5 – x7/7 +

11. (1-x)0.5 ~ 1 - x/2 – x2/(2\*4) - 1\*3\*x3/(2\*4\*6) -1\*3\*5\*x4/(2\*4\*6\*9)

12. (1+x)1/3 ~ 1 + x/3 -2x2/(3\*6) + 2\*5\*x3/(3\*6\*9)

13. (1+x)3/2 ~ 1 + 3x/2 + 3x2/(2\*4) – 3x3/(2\*4\*6) +9x4/(2\*4\*6\*8)

15. arsh(x) ~ x - x3/(2\*3) + 1\*3\*x5/(2\*4\*5) - 1\*3\*5x7/(2\*4\*6\*7)

16. ch(x) ~ 1 + x2/2! + x4/4! + x6/6! +

1. sh(x) ~ x/1 + x3/3! + x5/5! + x7/7! +
2. Si(x) ~ x – x3/(3\*3!) + x5/(5\*5!) – x7/(7 \*7!)+
3. Ci(x) ~ 1 – x2/(2\*2! + x4/(4\*4! – x6/(6\*6! +

**2.5.Булевские и битовые данные.**

**2.5.1. Битовый тип данных** **.**

Алгоритмы логического управления, программные модели конечных автоматов используют битовое кодирование событий и состояний. возможность выполнения операций с битами – уникальная особенность архитектуры MCS51.

Применимы логические операции с битами (&,|,~,^), эквивалентные битовым

командам (anl,orl,cpl,xrl) в А51**.**

Тип данных, доступный в С51 для ЭВМ с архитектурой MCS51.

В С51 и А51 подразумевается адресный доступ.

Биты упорядочены в поле **0x20-0x2F** из 128 бит (адреса 0x0-0x7F) в памяти **Data** и 128 бит в регистрах SFR (адреса 0x80-0xff)

Доступ к битам по адресу можно представить схемой

**char y1=mem^5**

**char y1=0x04D=0x09^5 x4 bit 0xAcc.5= 0xE5**

7 6 32 0 7 6 32 0

0 1 0 0 1 101 1 1 1 0 0 101

Bit Bit

0

7 4 3 0 7 3

**char bdata 0x09 0 0 1 0 1 0 0 1 1 1 1 0 0 000**

**char bdata mem**

Rd,Wr A

Ram

D

b7b6**b5**b4b3b2b1b0

Ms Dms

Bit

В **С51:**

**bit x1,x2;** //определение битовых переменных в битовом поле Data ( 00-7f)

char **bdata** mem,meme ; //слово в Data c битовой адресацией

**sbit y1= mem^0;** //0-ой бит ячейки mem

**sbit y1=P1^2;**  //второй бит порта Р1

**PSW=C.AC.F0.RS1.RS0.OV.-.P** - резервированные имена битов в регистре PSW

**Sfr** с адресом кратным 8, бит-адресуемый в поле бит 0x80 – 0xFF

SFR m=0xF8;

sbit m5=m^5;

***Программа в с51***

#include <reg51.h>

char bdata mem //бит-адресуемая переменная в формате **char**

sbit x1=mem^0; //биты двоичного набора

sbit y1=mem^1;

sbit y2=mem^2;

sbit z=P1^0;

**main()**

{

for(mem=0;mem<8;mem++)

{P1<<=1 ; z= y1&!y2|x1;}

}

В **А51** регистры **SFR(ACC. PSW, P0,..)** с адресами кратными 8, бит адресуемые Доступ к битам в SFR

**x4 bit ACC.5 ;**битовая переменная, соответствующая 5-ому биту АСС

**mov c, 0 ; бит С в PSW - прямой доступ**

Data(20h.0) С , 20h.0 – нулевой бит ячейки Data

**mov ACC.7, c ; c  Acc.7** прямая адресация

**mov c, x4**

Определен **Сегмент Битов** – 128 бит , прямой адрес бита 0-7f h, память совмещена с ячейками **0x20-0x2f** в Data, где i-ый бит находим в ячейке Data с

адресом **0x20+i/8**, номер бита **i%8,** если номер бита в типе **char**

**bseg at 0x10 ;**сегмент битов с 0x10-го бита в поле бит Data

**x0: dbit 4 ;**поле из четырех бит в сегменте

В SFR биты индексируют i=0..7 разряды бит-доступных регистров

**mov c, x0+2 ; x0-** адрес первого бита поля бит

**mov x4,c**

**Битовые операции** в А51

**anl c,{bit, /bit} ;bit – инверсия бита**

Например,  **anl c,/ACC.6**

**orl c,{bit , /bit} mov c,bit**

**setb bit, clr bit, cpl C**

Задания по разделу

1. z=(y1/x1 v y2x2)(/y1 v x2)

1. z= (y1 v /x1)(y2x2 v x1)
2. z=/x1(x2 v /x3) v x1x4
3. z=(x1 v /x2x3)(/x2 v x4)
4. z= /y1 v /y2(y1x1 v /x2)
5. z=(x1 v /x3x4)(/x1 v x2)
6. z=/y1x2 v y2(/x1 v /x2)
7. z=(/x1 v x2)(x1x3 v /x4)
8. z=(x1y1 v /x2y2)(/x2 v y1)
9. z=(/x1 v y1) (x2y2 v /y1)
10. z=y1(/y2 v /y3) v /y1y4

1. z=(y1 v /y2y3)(/y2 v y4)
2. z=/y1 y2 v /x1x2(y1 v /y2)
3. z=x1y1 v /x2(/y2 v /x1)

15. z=(x1 v /x2 v /x3x4)(/x1 v /x4)

**2.5.2.Булевский тип данных BOOL,**  **неявно** используемый в логических выражениях с предикатами (явно определяется в стандарте Си ) – значения 0 или ≠0 (0 и 1 – промежуточные при вычислениях предикатов)

Применяются логические операции c предикатами (&&, ||, ==, !, !=)

Операнды в следующем выражении могут иметь различные типы, но

истинность для значений (0, не0) имеет смысл и выражение вычисляется c использованием эквивалентных по смыслу программ с условными переходами

по таблицам истинности.

char aa,bb,cc,dd,S;

S=(aa<bb)&&(cc!=dd)||bb;

Пример интерпретации предиката (aa<bb)

**s.bb и s.aa** - двоичные числа в формате байта, где бит **s** обозначает знак. Представим их как дробные числа, в которых старший бит целая часть.

Тогда отрицательные **числа s.bb=1.0 - 0.|bb|= и s.aa= 1.0-0.|aa|,** где |bb| и |aa| **семи-**битовые модули **, s=1 –** старший бит в формате числа**.**

положительные числа **s.bb=1.0+ 0.|bb| и s.aa= 1.0+0.|aa| ,** 1-за пределами формата, знак s=0.

Следовательно,

1)для положительных чисел **a-b = (1.0+0.|aa| ) – ( 1.0+ 0.|bb| )= 1.|aa| - 1.|bb| , s=1** и если **(a<b)=true,** формируется заем С=1 в старшем разряде

2) )для отрицательных чисел **a-b = (1.0+(1.0- 0.|aa| )) – ( 1.0+(1.0- 0.|bb| ))= 0.|bb| -0.|aa|** и

a<b, если формируется заем С=1 в старшем разряде и модули |bb| < |aa|

3) )для чисел c разными знаками

**a-b = (1.0+ 0.|aa| ) – ( 1.0+1.0- 0.|bb| )= 1.|aa| + 0.|bb|** и перенос С=0

**a-b = (1.0+1.0 -0.|aa| ) – ( 1.0+ 0.|bb| )= - 0.|aa| - 1.|bb|** и перенос С=0

Таким образом, для интерпретации арифметического предиката для любых знаков в А51 выполняется инверсия знаков операндов и вычитание (a-b). Если заем С=1 при вычитании, то (a<b) =1(true)

Пример программы в А51

**(aa<bb)**

CLR C

MOV A,bb

XRL A,#0x80 ; s.bb^1.00 инверсия знака

MOV R0,A

MOV A,aa

XRL A,#0x80 ; s.aa^1.00 инверсия знака

SUBB A,R0 ; s.aa – s.bb

JNC M1 ; if( aa<bb) goto M1

(aa<bb)&&(cc!=dd)

MOV A,cc

CJNE A,dd, M2 ; if(cc!=dd) goto M2

(bb!=0)

M1: MOV A,bb

JZ M3 ; if (bb==0) goto M3

M2: MOV R7,#0x01

SJMP M4

M3: MOV R7,#0x00 ; S=0

M4: MOV S,R7 ; S=1

**Задание.** Построить таблицу истинности из 8 наборов значений чисел со знаками, где каждый из предикатов {**(aa<bb), (cc!=dd)}** **true** или **false**

**Вычисление предикатов.**

**1. [(a!=b)||(c==d)]&&(b<d)**

**2. (a!=d)||(b>c)&&(d<b)**

**3. (a>d)&(b!=c)||(d>=c)**

**4. (a<c)||(b&c)||d>a)**

**5. (a<b)||(a>c)&&(d>a)**

**6. (a<b)||(a>c)&&!(d=a)**

**7. ![(a!=b)||(c==d)]&&(b<d)**

**8. (a&b)||(b>c)&&(d<b)**

**9. (a&c)&(b!=c)||(d>=c)**

**10. (a=c)||(b&c)||d>a)**

**11. (a<b)||!(a>c)&&(d>a)**

**12. (a<b)||(a>c)&&(d<b)**

**13. (a!=b)||(c==d)&&(b>d)**

**14. a||(b>c)&&(d<b)||(c<=d)**

**15. (a<d)&(b!=c)||(d>=c)**

**2.5.3. Вероятностная логика**

В работе [] предлагается арифметика вычисления вероятности, эквива-

валентная по смыслу логике, представленной логическими формулами.

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(ךx=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(ךxi=1)=(1-Ri)=Qi** –вероятность инверсного значения переменной ךxi, то

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

**(3) P{(ךxi&ךxj)=1}=(1-Ri)(1-Rj)=QiQj**

**P((xivxj)=1)=P(ך(ךxi&ךxj)=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

**(4) P(ךךxi=1)=1-(1-Ri)=Ri =1-Qi,** двойное отрицание

(5**)** правила де Моргана

**P(ך(xivxj)=1)=1-(1- QiQj)=QiQj=P{(ךxi&ךxj)=1}**

**P{ך(xi&xj)=1}=1-P{(xi&xj)=1}=1- Ri\*Rj=P((ךxivךxj)=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 (1-5), затем последовательной заменой логических операций вероятностными формулами и упрощающими алгебраическими операциями

Пример

**F= x1(x2vx3vךx4) vx5(x6vx7ךx8)=ךך [x1(x2vx3vךx4) vx5(x6v (x7ךx8))]=**

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

**= ך [ך(x 1(x2vx3vךx4)) &ך (x5(x6v (x7ךx8)))]=**

**= ך [ך(x 1 ךך(x2vx3vךx4)) &ך (x5 ךך (x6v (x7ךx8)))]=**

**= ך [ך(x 1 ך(ךx2ךx3x4)) &ך (x5 ך (ךx6ך (x7ךx8)))]=** формула ОПЗФ содержит только инверсии и конъюнкции

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

**1-[(1-R1(1-Q2Q3R4)) (1-R5(1-Q6(1-R7Q8))]**

**Задание.** Для заданного варианта функции выполнить ее преобразование в ОПЗФ и вывести таблицу истинности в порт. Значения False(0) и True(1) кодируются при вычислениях в байтах.

**Литература.**

1.Сташин В.В. Урусов А.В. Мологонцева О.Ф. Проектирование цифровых устройств на однокристальных микроконтроллерах, М: Энергоатомиздат, 1990.

2.Злобин В.К. Григорьев В.Л. Программирование арифметических операций в микропроцессорах, М:ВШ, 1991 г-303 с

3.Help в Keil (C51, Макроассемблер, Система команд MCS51).

4.Копченова Н.В., Марон И.А. Вычислительная математика в примерах и задачах, М:Наука, 1972, 367 c

5. Довгий П.С. Скорубский В.И. Организация ЭВМ Пособие к лаб. работам

Изд. ГУ ИТМО 2009г-56 с.

**Приложение 1**

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

**Аpифметика и логика Пеpесылки**

ь -------------------------------------------------

**add a,{ri,@rj,#d,ad} aa+{...}, пpизн c,v,p mov a,{ri,@rj,#d,ad} a{......}**

**addc a,{ ...........} aa+{...}+c, .... mov {ri,@rj},a {.....} a**

**subb a,{............} aa-{...}-c, . ... mov {ri,@rj},ad {.....} ad**

**inc {ri,@rj,ad,dptr,a} {....}+1 mov ad,{ri,@rj,#d,ad,a} ad{.....}**

**dec {ri,@rj,ad,a} mov {ri,@rj},#d**

**mul ab b.aa\*b v=(a\*b>255) 0c,p mov dptr,#d16**

**div ab aa/b, ba%b (b==0)ov, 0c movc a,@a+dptr aCode(dptr+a)**

**movc a,@a+pc aCode(pc+a)**

**anl a,{ri,@rj,#d,ad} a&{..}a 0c,p movx a,{@rj,@dptr} axram{..}**

**anl ad,{#d,a} movx {@rj,@dptr},a xram{..}a**

**push ad Data(+sp)Data(ad)**

**orl a,{ri,@rj,#d,ad} a v {…} a pop ad Data(sp-)Data(ad0 orl ad,{#d,a} orl ad, {#d,a} a v {…} Data[ad] xch a,{ri,@rj,ad} a{.....} xrl a,{ri,@rj,#d,ad} xrl a,{ri,@rj,#d,ad} a v {…} a xchd a,@rj a(3-0)@rj(3-0)**

**xrl ad,{#d,a} swap a a(3-0)a(7-4)**

**clr a -**

**cpl a не(a) команды булевского процессора**

**rl a rol(a) p**

**rlc a rolc(a,c) c,p mov bit,c mov c,bit**

**rr a ror(a) p clr {c,bit} anl c,{bit,/bit}**

**rrc a rorc(c.a) c,p cpl c orl c,{........}**

**da a коppекция (+,-)2 setb {c,bit} jbc bit,rel**

**jc rel jnc rel jb bit,rel jnb bit,rel**

**Упpавление пpогpаммой и ветвления обозначения битов SFR адрес**

**7 6 5 4 3 2 1 0**

**ljmp a16 PCa16 acc . . . . . . . . e0i**

**ajmp a11 PC(10.0)a11[10.0] b . . . . . . . . F0i**

**sjmp rel PC+2+/- rel[6.0] psw c ac f0 rs1 rs0 ov . p d0i**

**jmp @a+dptr PCa+dptr sp 81**

**jz rel PC+2+/-rel[6.0],если (a=0) dph 83**

**dpl 82**

**jnz rel .... ,если (a<>0) ie ea . . es et1 ex1 et0 ex0 a8i**

**jc rel .... ,если C p0 . . . . . . . . 80i**

**jnc rel .... ,если неС p1 . . . . . . . . 90i**

**jb bit,rel PC+3+rel,если bit=1 p2 . . . . . . . . a0i**

**jnb bit,rel .... ,если bit=0 p3 rd wr t1 t0 int1 int0 txd rxd b0i**

**jbc bit,rel ... ,если bit=1,bit<-0 ip . . . ps pt1 px1 pt0 px0 b8i**

**djnz {ri,ad},rel {}-1;**

**PC+1/2+/-rel[6.0],если {}<>0 tmod gate1 c/t1 m1 m0 gate0 c/t0 m1 m0 89**

**cjne {ri,@rj},#d,rel rel,если {}<>#d tcon tf1 tr1 tf0 tr0 te1 it1 ie0 it0 88i**

**lcall a16 стекpc, PCa16 th0 8c**

**tl0 8a**

**acall a11 . ..., PC(10-0)a11[10.0] sbuf 99**

**ret PCстек th1 8d**

**tl1 8b**

**reti PCстек,tf0 pcon smod . . . gf1 gf0 pd idl 87**

**nop пропуск scon sm0 sm1 sm2 ren tb8 rb8 ti ri 98i**

**обозначения адресов и признаки**

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

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

p - нечетное число единиц в аккумулятоpе

f0- пpизнак пользователя, rs1.rs0 - банк pегистpов

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

ad - адpес Data, имя pегистpа SFR

bit - адpес 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 - инвеpсия бита

rel - <метка>=смещение РС в доп коде

------------------------------------------------------------------------------------------------

**Приложение 2**

**Ассемблирование.**

При разработке ассемблерной программы может быть использована программа в С51 в качестве спецификации и в комментариях .

Структура ассемблерной программы

* 1. Выбор программной модели – по умолчанию подразумевается MCS51

если модель изменяется , то

**$nomod51** ;отмена стандартных режимов MCS51

**$include (reg515.inc)**  ;загрузка файла определения регистров SAB515

* 1. Распределение памяти данных **dseg, xseg, iseg, pseg**
  2. Формирование программного кода **cseg**

PC=0: Jmp Start – запуск программы с адреса 0000

Таблица векторов прерываний

Таблица констант

Подпрограммы

Программа

End

**Макроассемблер**

А51 является макроассемблером, позволяет заменить повторяющиеся небольшие фрагменты текста (3-5 команд ассемблера) одной **макрокомандой-**

ссылкой с параметрами .

Структура **макроопределения**

*<имя макрокоманды>* ***macro*** *<список формальных параметров>*

*<тело макроопределения – список ассемблерных команд параметрами)*

***Endm***

В программе используются макрокоманды с именем, обозначенным в MACRO, и фактическими параметрами, для которых **имеет смысл** подстановка в теле макроопределения.

Компилятор заменяет эти ссылки соответствующим отредактированным текстом (**макроопределением**).

Макрокоманды (макроподстановки) **сокращают текст программы** и позволяют использовать расширенную систему команд.

Учитывая ограниченный доступ к различным типам памяти, можно с использованием макрокоманд расширить список команд.

r**sadd macro ri, SS ;** SS-имя в памяти Data

mov a,ri

……………

add a, SS

mov ri,a

**endm**

**……………..**

rsadd r1,#55 ; обращение к макрокоманде

**Реассемблер и листинги компиляции**

В Кейл компиляция программы в Си и Ассемблере сопровождаются формированием листингов. Для программ в Си – в форме Реассемблирования,

что позволяет контролировать семантику исполнения соответствующих операторов программы.

В ассемблере листинг также полезен для контроля распределения памяти.

В окне VisialKeil выбрать меню Projects/**options/listing** задать **Assembly Code** - формирование листинга компиляции в Ассемблере. В файле .LST будет получен следующий листинг этой программы.

; FUNCTION main (BEGIN)

; SOURCE LINE # 4

; SOURCE LINE # 5

; SOURCE LINE # 6

0000 E590 MOV A,P1

0002 C4 SWAP A

0003 540F ANL A,#0FH

0005 75F00A MOV B,#0AH

……………………..

000d 2F ADD A,R7

000e F5A0 MOV P2,A

; FUNCTION main (END)

В файле с расширением **.M51** приведено распределение памяти для проекта

Пример размещения данных в C51

**#include <reg51.h> //подключение каталога элементов памяти в С51**

**char code cc[ ]=”abcde”;**

**char data x,y; //char x,y по умолчанию**

**char xdata yy[100];**

**main(){**

**x=0; y=20;**

**while(y--)**

**{ yy[y]=y;**

**P2=x++; } //вывод в порт**

**while(1); //динамический останов**

**}**

**Приложение 3**

Раcпределение памяти Code

0000 Jmp ?start

**Вектора прерываний**.

3 Jmp int0

b Jmp tm0

13 Jmp int1

1b Jmp tm1

23 Jmp usart

**Станд библ C51**

**Прикл библ**

int0: **обработчики**

tm0: прерываний

………………….

?start **Startup51**

Jmp main

Main: основная программа

Распределение памяти Data c абсолютной адресацией

dseg at 0x8

**stack: ds 4**

**y: ds 1**

**Абсолютная адресация**

**i equ r0**

**j equ r1**

**A equ 0xE0 ;определение адресов SFR обычно для новых моделей, если не доступен файл reg .inc**

**P1 equ 0x80**

**Программный код**

**cseg at 0**

**jmp start**

;таблица векторов прерывания

**cseg at 3**

jmp inte0

**cseg at 0Bh**

jmp intrtime0

;таблица констант

**tb1: db ‘abcd’**

**tb2: db 10110111b**

**tb3: db -2,25H,30**

**m1: dw 5330H**

**cseg at 40h**

**stsrt:** прикладная программа

Типовой порядок размещения различных сегментов данных в иерархической памяти Data

(R0-R7) 0-7

глобальные 0x08-0x17

переменные

стек 0x18 - 0x1F

бит-адресуемые 0x20 – 0x2F

ячейки

локальные 0x30 -

переменные

**Приложение 4.**

**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.с** (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**)

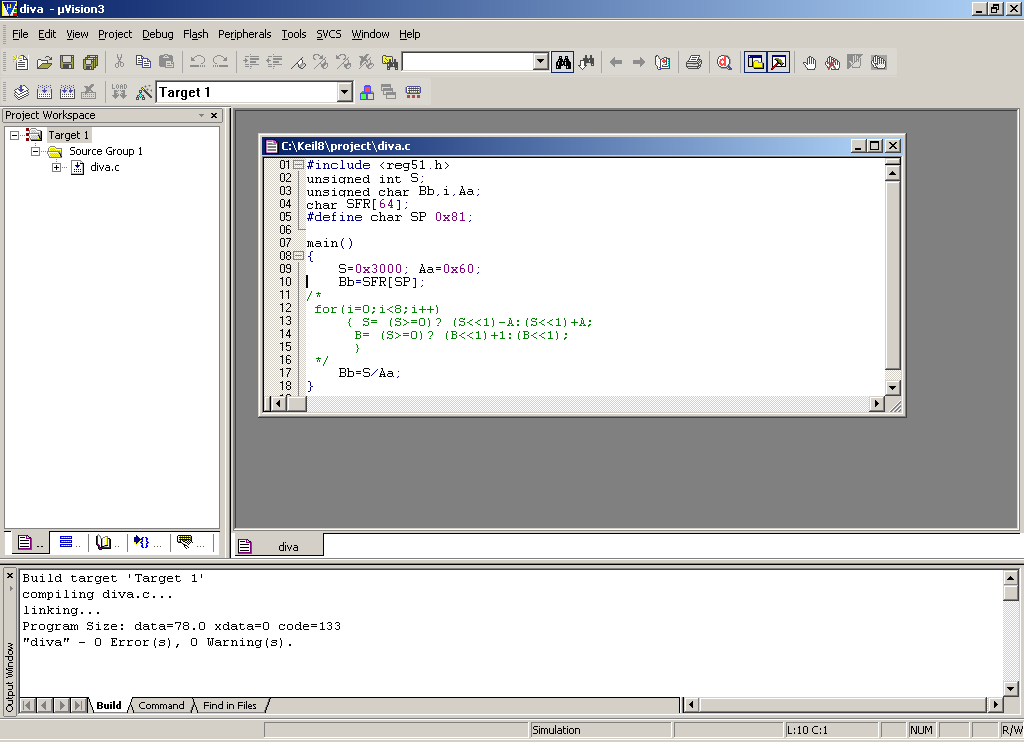
Система содержит полную библиотеку элементов с ядром MCS51, выпускаемых различными фирмами. Библиотека дополняется новыми элементами в последних версиях, которые можно загрузить из Интернета. Система работает во всех версиях ОС Windows.

**Окно** Vision

1

2

3



4

5

6

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  Vision 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

Cтандартная 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

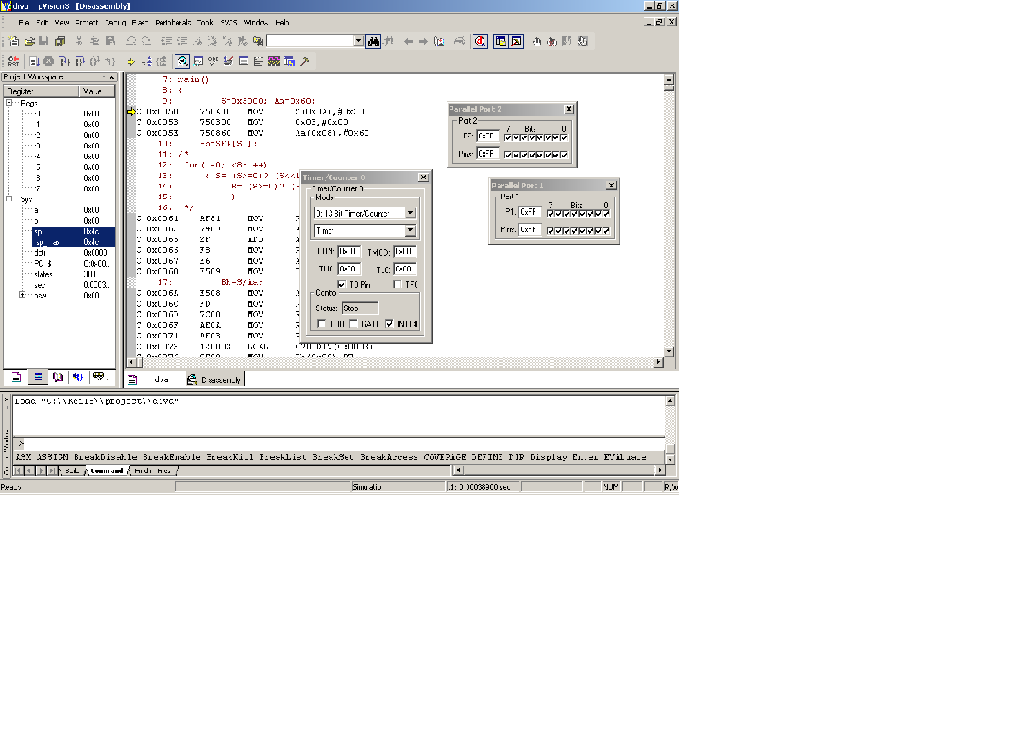
Содержит ссылки на периферию конкретной выбранной в проекте машины.

**Окно Загрузчика (Debug)**

1

2

3



4

5

10

6

8

9

7

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.**

**Приложение 5.**

**Вопросы по курсу лабораторных работ к зачету и экзамену.**

**1.**Структура и возможности системы Кейл.

2.Программная модель MCS51 в С51.

1. Программная модель MCS51 в Ассемблере.
2. Структура памяти – адресация.
3. Иерерхия памяти Ram
4. Микропрограммирование и структурная схема ЭВМ
5. Арифметические и логические операции.
6. Команды управления программой.
7. Организация ввода данных с клавиатуры
8. Преобразование 2/10, 10/2 целых чисел при вводе-выводе .
9. Микропрограмма преобразования
10. Преобразование 2/10, 10/2 дробных чисел при вводе-выводе .
11. Символьные преобразования 10/16.
12. Символьные преобразования 16/10.
13. Программа умножения в С51.
14. Микропрограмма умножения
15. Программа деления в С51.
16. Микропрограмма деления
17. Вычисление функций с дробными числами – масштабирование
18. Макроассемблирование, применение.
19. Вычисление предикатов
20. Битовые данные, адресация
21. Вычисления в вероятностной логике