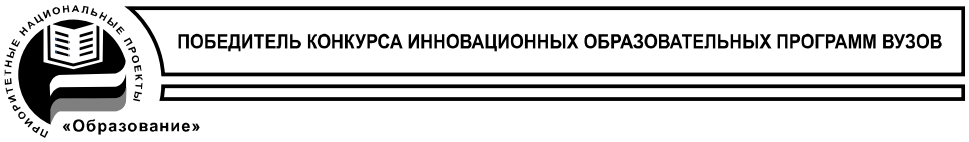
09.09.17

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

1

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 в 90-х годах была промышленным стандартом и до сих пор полезна как широко используемая и доступная в приложениях.

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

Содержание

стр.

3

Введение

МикроЭВМ **(MCU)** (mcs51, sab515, Pic16, Cortex..) в отличие от **микропроцессоров общего назначения (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 размещается 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, а язык А51 близкий по смыслу к микропрограммированию.

Для описания реализаций алгоритмов в mcs51 используется 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**.

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;

5

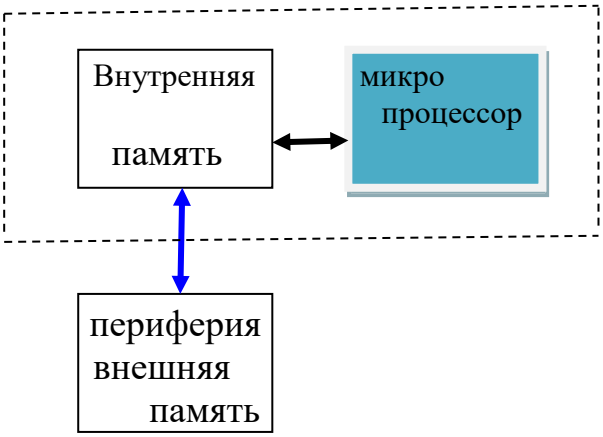
Во второй части:
- система прерывания, измерение времени и ШИМ
– прямое программное управление вводом с клавиатуры,
- ввод данных с использованием аналого-цифрового преобразования, вывод с цифро-аналоговым преобразованием
– последовательный канал передачи данных USART.
– Синхронный интерфейс I2C,SPI

При этом используются MCU Sab515 (Siemens,/Infinion), Aduc8xx (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)

I. Программные модели mcs51

Признанная схема организации любого компьютера



В одной микросхеме микроЭВМ интегрированы блоки внутренней памяти и микропроцессор. В более мощных компьютерных системах блоки интегрированы на одной печатной плате

Традиционно предполагается [Новиков, Таненбаум,Орлов] трех-уровневая система реализации алгоритма – язык программирования, ассемблер и схема с микропрограммным управлением.

В системе программирования используем язык C51 или ассемблер A51, транслятор алгоритма в программу машинных кодов и моделирование алгоритма в системе команд mcs51 при тестировании

Программирование в C51 использует программную модель памяти высокого уровня и для демонстрации семантики (программного управления при исполнении алгоритма) ассемблер

. Программирование в A51 использует программную модель памяти уровня функциональной схемы и для демонстрации семантики (программного управления при исполнении алгоритма) микропрограммирование.

1.1. Программная модель в C51.

Программная модель ЭВМ представляет ресурсы памяти, интерфейсы ввода-вывода, неeобходимые и достаточные для программирования в Си. Применение стандартного языка Си (стандарт ANSI) к конкретным MCU имеет ограничения, поэтому используются модификации с расширениями в C51, 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, ...)

В Keil для mcs51 предлагается аппаратное расширение стандарта C51 с ассемблером A51.

Диаграмма MCU(рис.1.1.) представляет доступную иерархию памяти и интерфейсы ввода-вывода в C51.

Все типы памяти отличаются объемом , способом доступа и временем доступа. В виде диаграммы приведена программная модель в C51

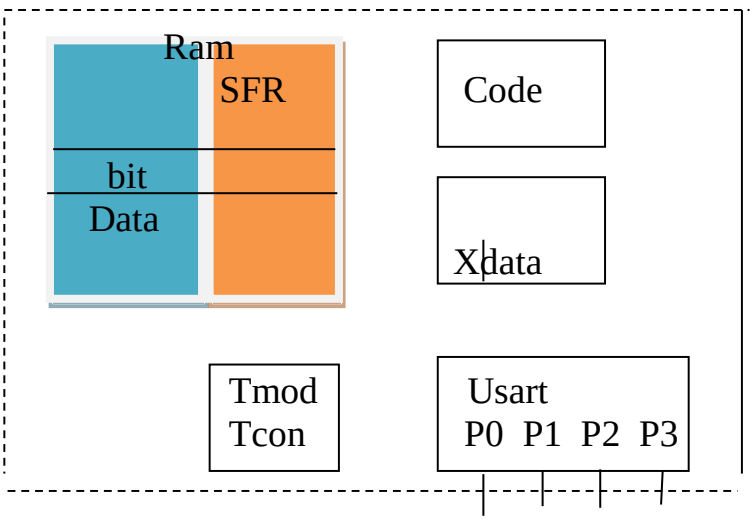


Рис. 1.1. Программная модель в C51.

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).

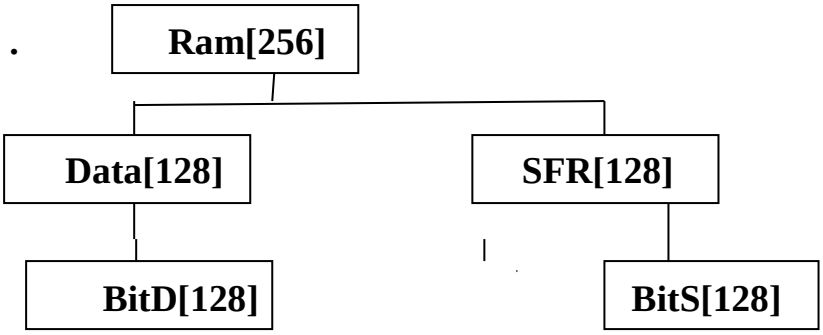
7

При размещении формата данных в памяти используются последовательное со старших байтов к младшим **(BigEndian) .**

целое число long	a3	a2	a1	a0 (BE)
	04	8F	F4	EA
адреса в Data	8	9	A	B

1) Две области памяти **Data и SFR** можно рассматривать как непрерывную **Ram[256] ,** которая имеет иерархическую организацию .

Ram можно представить в виде диаграммы.



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

а)Data – память данных – неявно доступная при определении переменных, объем 128 байт

```
char  x,z;   //байт данных со знаком
int  y;     //формат 2 байта со знаком
long  t;     //формат 4 байта со знаком
```

При компиляции используется прямой адресный доступ.

```
                mov x,z
char  idata  z;  //определение доступа с  косвенной адресацией
                mov R0,#z;
                mov  x,@R0
```

б)Регистровая память Sfr(8-разрядные регистры специальных функций) – 128 байт адресное пространство. В С51 **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.

Поле битов в Data и SFR

а)128 бит в поле, выделяемом в памяти **Data :**

```
bit x1,x2;  / битовые  переменные в битовом поле
char bdata  mem,meme ;     //относительные адреса  байтов (0-F) в массиве байтов Data[0x20-0x1F]  с битовой адресацией
sbit y1= mem^0;  //0-ой бит 0-байта mem
```

б) В **SFR** поле битов распределяется по регистрам

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

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

SFR m=0xF8; //бит адресуемый регистр SFR

sbit m5=m^5; //бит регистра **m**

sbit z=P1^2; //бит **sfr** порта P1 (не смешивать с операцией ^)

sfr TT=0xc0; //бит-адресуемый регистр в sfr

sbit T=TT^0;

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 память программ **Code** – 65 Кбайт адресное пространство, доступ – чтение данных, чтение и исполнение команд

Данные могут быть определены в виде констант

char code x=”abcв”;

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

1.2. Программная модель в A51 (система команд в Приложении 1)

В программной модели A51 уточняется организация памяти с учетом режимов адресного доступа к данным.

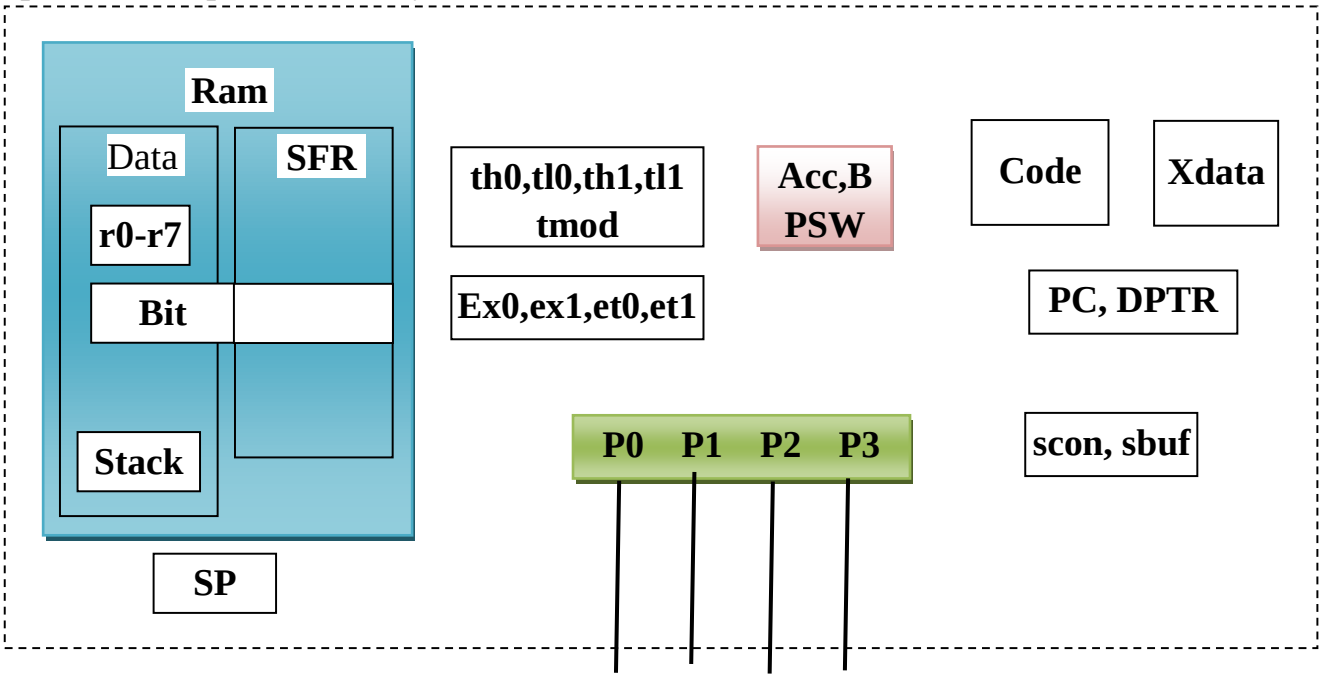


Рис.1.2. Программная модель в A51.

9

Уровень ассемблера – алгоритм можно видеть в виде структурной схемы и управление этой схемой микропрограммой. Основной способ контроля – компиляция и тестирование, представление результатов в графике.

Схемный уровень представлен функциональной схемой проекта в графике или специальным языком High Digital Dezign logic (HDL) и микропрограммой управляющего устройства.

Для тестирования,тестирования и кодирования микропрограмм предлагается программа МИКРО.

Завершающая стадия проекта ЭВМ поддерживается проектами на уровне HDL (системы Xilens или Altera), в которых для тестирования используются модели схем с временными диаграммами.

1) Размещение и адресация данных в памяти Ram.

a) Прямая адресация к данным в памяти Data.

В A51 используется сегментная организация памяти **(dseg, xseg, cseg).**

Для упрощения могут быть выделены **абсолютные сегменты**

```
Dseg at 0x30        ; адрес начала сегмента в Data
    X: ds 2  ;X-адрес первого (0) байта блока X данных из 2-х байт
   Dm: ds 4
```

Команды с прямым доступом к данным в этом сегменте

```
mov X,Dm      ;Data[Dm]→Data[X]
mov a, Dm+2   ; Data[Dm+2]→acc
```

Для отдельных байтов можно определять адреса в Data псевдокомандами без учета разделения на сегменты

```
Ss  equ  0x22
Ee  data  0x66
```

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

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

В C51 адреса определяются загрузкой адресного файла

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

В меню проекта **Project/Option../Device/Ineel** выбирается конкретная фирма **(Intel,Infenion,ADI,TI,...)** и микросхема **(80c51BH,Sub515, ...) и файл адресов**

. #include <reg515.h>

sfr TT=0x95 – свободный адрес в SFR

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.

По адресам **(0x80-0xFF)** в A51 свободные регистры в SFR могут быть определены как **TT equ 0x95**

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 происходит **когерентно** по адресу в SFR и неявно в рабочие регистры.

Только регистр ACC имеет различимый идентификатор с рабочим регистром **A** в A51.

В C51 **char A** – определяет независимый адрес в Data. Все остальные регистры имеют общий идентификатор с адресным доступом в **SFR** и неявный (по умолчанию) к рабочему регистру (регистры B, PSW, P0-P3 и другие SFR можно использовать в C51 и A51).

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** .

c) 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.

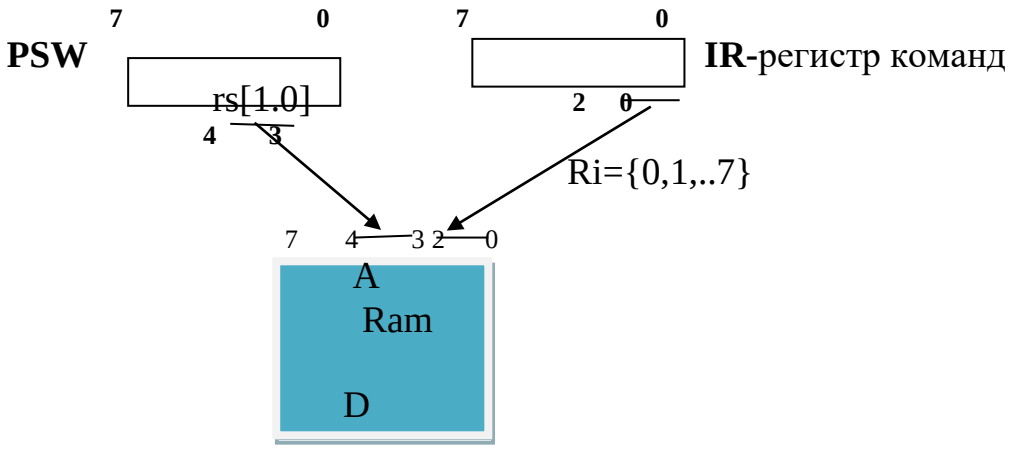
В команде **mov 01, r2** прямой адрес 01 в Data совпадает с регистром r1, однако команда **mov r1, r2** неверна.

В C51 предусмотрено автоматическое переключение банков регистров и сохранение **контекста в Ri** 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**



1

Рис.1.4. Схема доступа к регистру Ri

Исполнимое в **Си** описание элементарной операции в этой схеме определяем как **функциональную микрокоманду**

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

d) Косвенная регистровая адресация

В **C51** обозначается явно косвенное обращение к памяти IDATA~ **Data(0-0x7f)**

```
char idata x;
 x=55;
```

В **A51** 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 к адресам **Sfr** (0x80-0xff**)** не определено.

e) Стековая косвенная адресация в **Data** с **автоиндексацией** с использованием регистра-указателя SP.

В C51 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**.

f) **Битовая память Bit –** поле из 256 битов (биты **0-7F** в **0x20-0x2F** байтах **Data** и биты **0x80-0xFF** в **0x80,0x88,..0xF8** байтах **SFR**) , размещаемых последовательно в выделенных форматах данных и доступные в них по номерам битов, доступ – чтение и запись.

Работа с битовыми данными рассматривается в разделе 2.5.

2) Адресация в памяти данных Xdata

В С51 – прямой доступ

```
char xdata  xx,yy;
    xx=0x55;
```

реализация в **A51**

```
mov   dptr,#xx(0x0001) ;в dptr формируется прямой адрес  xdata
mov   a,#0x55
movx   @dptr,a
```

- косвенный страничный

```
char  pdata  x;  //переменные размещаются со смещением 0 по 0xff
      P2=0;  //задается адрес страницы

      x= 0x55;
```

реализация в **A51**

```
mov P2,#0
mov r0,#x
mov a,#0x55
movx  @r0,a       ; A→Xdata(P2.@r0) →A,  в P2 адрес
```

страницы, x=@r0 –смещение в странице

В **A51** - два режима косвенной адресации

прямой доступ

```
xseg at 0x100 ;абсолютный сегмент
      mm: ds 50    ;адрес первого байта 0x100  массива из 50 байт
.....
      mov dptr,#mm    ;адрес
      movx a, @dptr    ;Xdata(dptr) → A
      movx  @dptr,a
```

3)Адресация данных в памяти Code.

Память типа ROM(Read only), назначение памяти – хранение двоичного кода программ, чтение команд при исполнении программы.

При адресации команд используется 16-разрядный программный счетчик (PC), который при включении питания автоматически устанавливается в состояние PC=0. Регистр PC не входит в число адресуемых регистров SFR, автоматически инкрементируется при последовательном чтении байтов команд и модифицируется при исполнении команд передачи управления .

В памяти Code также могут храниться данные-константы.

В С51 предполагается прямой доступ к константам по имени

```
char  code  x[]="abcdef";  // x[0]-адрес первого байта строки
int code y[]={221,332,44,55};
```


1

Доступ по **i**-индексу **char xx= x[i];**
int yy=y[i++];

a) В A51 используется сегментная организация
Непосредственная адресация
mov a,#55 ; Code[PC+] → a
mov a,#55h = mov a,#0x55= mov a,#01010101B
Относительная адресация
movc a,@a+pc ; Code[PC + a] → a **;** относительно текущего PC**,** в ACC размещается индекс
Базовая индексная адресация (базовый адрес в DPTR, индекс в ACC)
movc a,@a+dptr; Code[dptr+a] → a

Определение абсолютных сегментов
cseg at 0 ; старт при сбросе и включении питания в памяти Code
jmp start
.....
cseg at 0x40 ; абсолютный сегмент памяти Code – прикладная программа
start: jmp first
yy: db "abcde" ; адрес первого байта константы в виде строки
.....
first:
mov dptr,#yy ; сохранение адреса
movc a,@a+dptr

Диаграмма доступа к данным в A51.

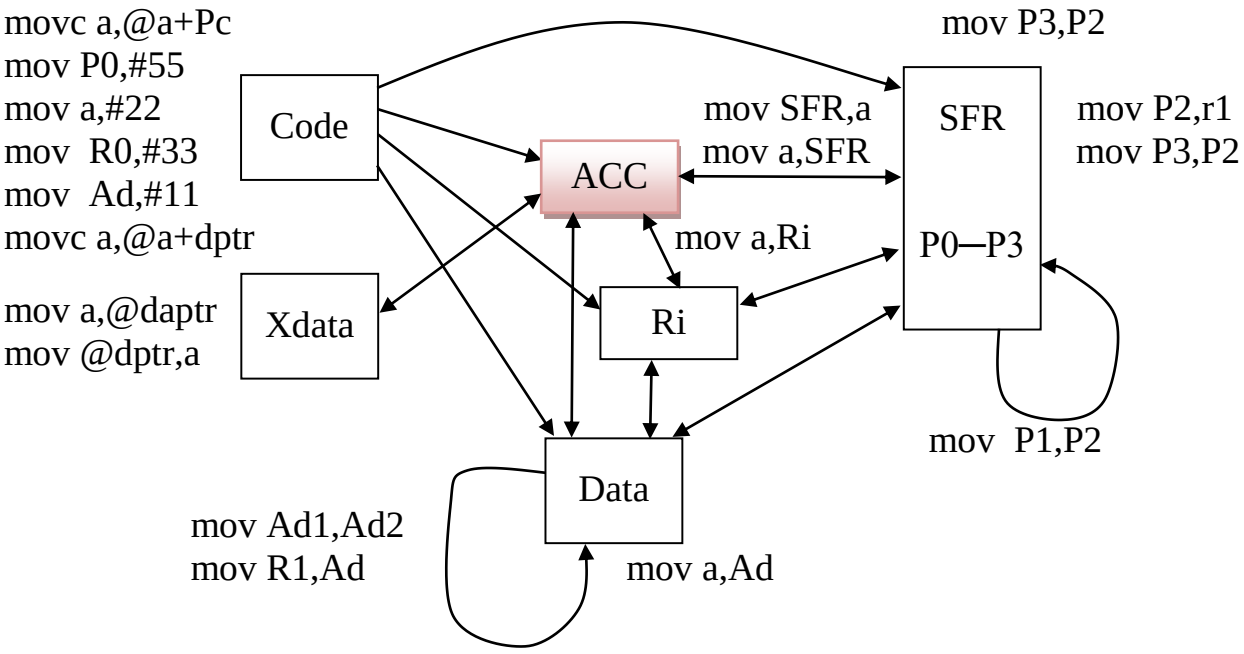


Рис.1.2. Диаграмма доступа к данным в командах **mov**

b) Адресный указатель **pointer(*)** определяет в **C51** косвенный адрес данных и возможность прямой модификации целого адреса с использованием арифметики.

В C51 различают память, где размещаются данные и указатель .

char code *aa=”abcdef”; // текстовая константа размещается в памяти **Code**, указатель с именем **aa** определяет два байта адреса первого символа константы в **памяти Data[].**

char x

data *qq=0x200;

***qq=0x44;** //константа 0x44 размещается в памяти **Xdata**, указатель qq—адрес константы 0x200 формируется в двух байтах адреса в ячейках памяти **Data**[].

char *tt=0x40;

***tt=0x33;** //константа 0x33 размещается в памяти **Data**, указатель tt—адрес константы 0x40 формируется в регистре R3**.**

c) 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**.

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

Например, вычисление факториала определяется 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);}
```


1

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

2) Эквивалентная **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 ; абсолютный сегмент памяти Code – прикладная программа

start: jmp first

yy: db "abcde" ; адрес первого 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 ACC 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 ACC (=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

d) Структурная схема.

В логическом синтезе схем ЭВМ в ПЛИС используются стандартные языки проектирования высокого уровня(High Design Language), которые соединяют две части алгоритма (структуру памяти или структурную схему) и управление (микропрограммное управление, или в явном виде конечный автомат) . Стандартные языки **VHDL, VERILOG, AHDL** и графические схемы проектируются на функциональных элементах.

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

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()
 {   PC=0x100;
     A=0xff;
     Code[PC]=0x24;
      Code[PC+1]=0x10;

      Ir=Code[PC++];  //чтение команды
      if(Ir==0x24)       // декодирование команды
     {
     Pa= Code[PC++];  //чтение второго байта команды
     Pb=A;
     PSW=0;
     A=Pa+Pb;
    P2=PSW;    //вывод PSW в выходной порт
      P1=A;
        }
   }
```

Здесь формирование признаков в **PSW** контролируется реальной командой **Add** ассемблера, а смысл признаков может быть определен на верхнем уровне – в **C++** и **C51**..

Вычисление в **C++** признаков **C** и **OV** в **PSW** функциональными микрокомандами

```
 typedef  unsigned char  uchar;
typedef  unsigned int  uint;
  uint A=0;
```


1

uchar PA, //рабочий регистр в RALU
PB, //рабочий регистр в RALU
PSW; //слово состояния
main(){
A=PA + PB;
PSW= (A&0x100)? PSW|0x80 : PSW&0x7f; //формирование **C**
PSW=((~(PA^PB)&0x80)&(PA^AB))? PSW|0x04 : PSW&0xfb; //признак переполнения **OV**

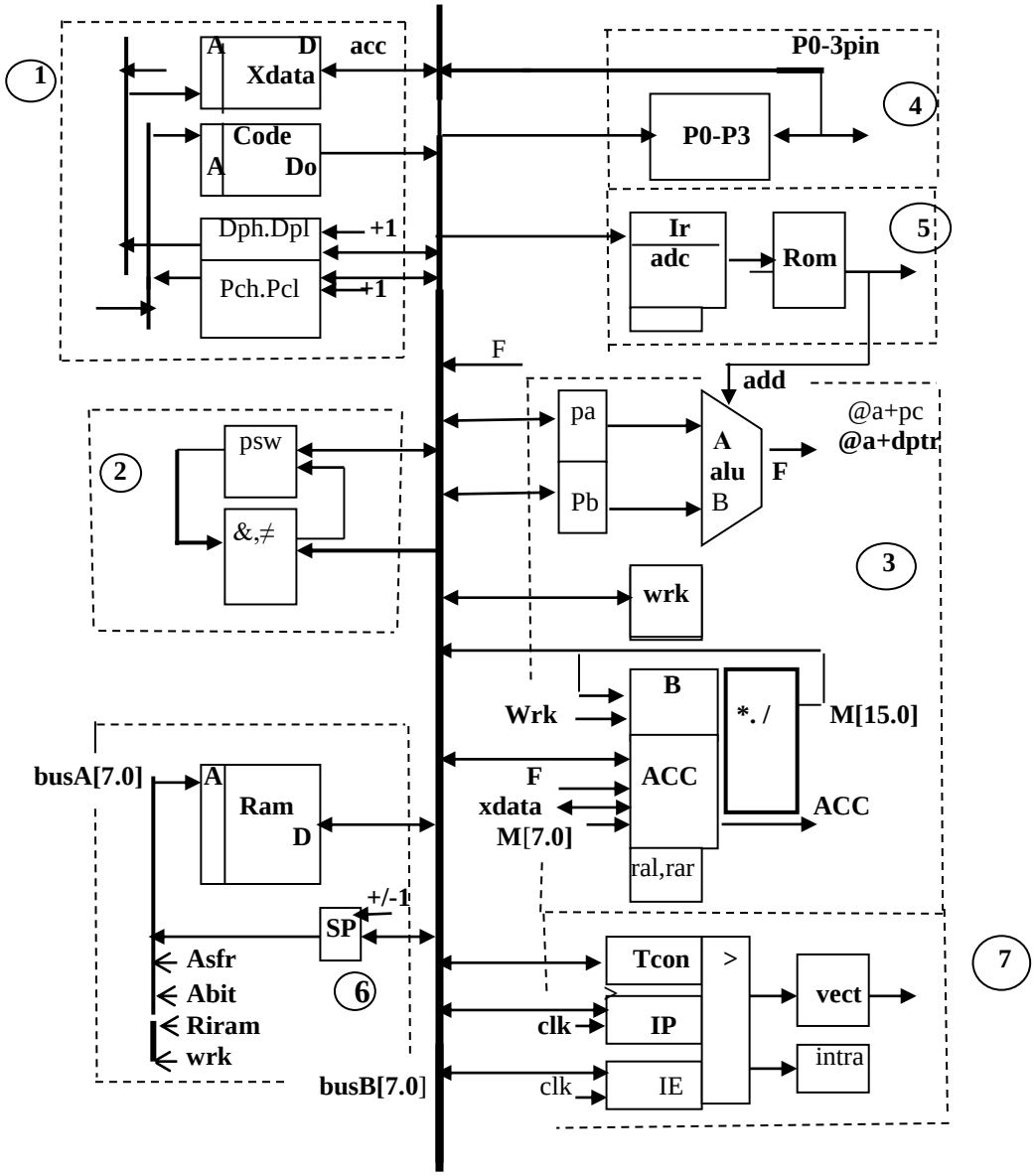


Рис.1.4. Структурная схема

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

Пусть 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 BigEndean –записи в двоичную запись пересчетом в двоичной системе по схеме Горнера можно представить рекуррентной формулой пересчета

N= 65432=$a_4a_3a_2a_1a_0$ =a_4*10^4 +a_3*10^3+a_2*10^2+a_1*10^1+a_0 = $\sum a_i 10^i$

=((((0*10+a_4)10 +a_3)10+a_2)10+a_1)10+a_0 =

S_1= (S_0*10+ 6) i=0, S_0=0

S_2= (S_1*10+5)

S_3= (S_2*10+4)

……

(1.1) **S_{i+1}= S_i*10+a_{n-i}** S_0=0 ,i=0,1,..n-1 **a_0 a_1 .. a_i a_{i+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.

char i,x; //двоичные числа в памяти Data,

char code y[ ]= "123"; //символьная строка в памяти Code

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

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

1

```
main()  //преобразование  десятичного числа из ASCII в машинное char x
        {     x=0;
A:  for (i=0; i<3; i++)
          x=x*10+(y[i]&0x0f);        //двоичное число
B:   for(i=7;i>=0; i--)   //преобразование  двоичного числа char x  в ASCII
     {  yy[i]= (x&0x01) | '0' ; x=x>>1;} }
           }
```

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

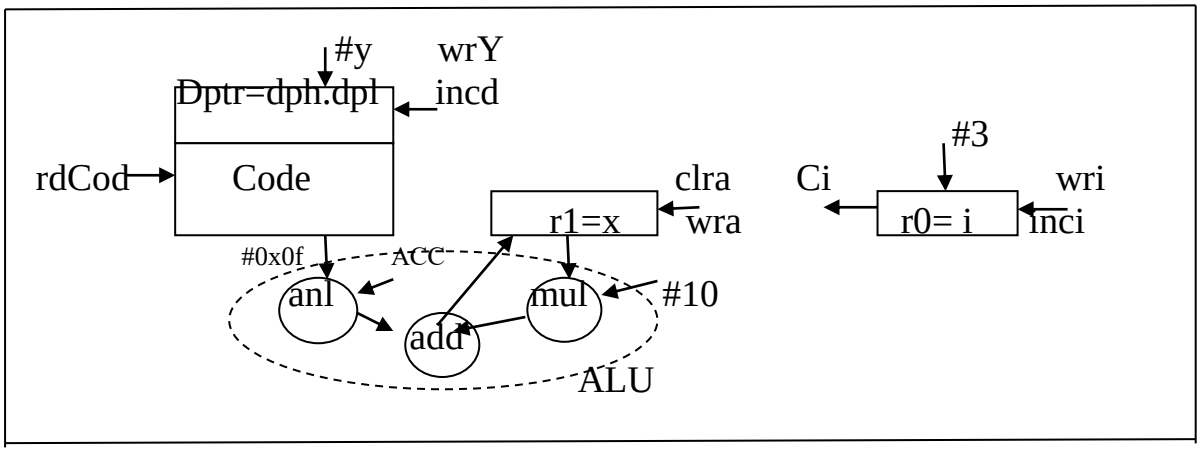
```
#include  <reg51.h>
#include  <string.h>

char  i,x;
char  code y[ ]= "123";
char  xdata yy[8];

main()
        {     x=0;
A:  for (i=0; i<3; i++)
          x=x*10+(y[i]&0x0f);
B:   for(i=7;i>=0; i--)
     {  yy[i]= (x&0x01) | '0' ; x=x>>1;} }
           }
```

2)Программирование в A51

Алгоритм в C51 используем как спецификацию и комментарии в A51. 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



Память – постоянная память микрокоманд Code с адресным регистром dptr

- регистр аккумулятор **x equ r1**,
- счетчик тактов **i equ r0**

cseg at 0 ;начало сегмента Code после сброса

jmp start ; команда старта

y: db "123" ;текстовая константа в сегменте

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 A51 (A--преобразование 10/2)

```
start:
        mov i,#0
        mov x,#0
        mov dpl,#y       ;wrdp
    // x=x*10+(y[i]&0x0f);
cikl:   mov b,#10
        mov a,x
        mul ab
        mov x,a              ;b=ab=a*10
        clr a
        movc  a,@a+dptr   ;a=rdCode
```


2

```
anl a,#0x0f          ;a&0x0f
add a,x       ;wra→a= b+ a&0x0f
mov x,a
inc i
cjne  i,#3,cikl      ;for (i=3; i>0; i--)
end
```

B: Преобразования в двоичный текстовый ASCII формат

Используемая память

- регистры **A, dptr, i**
- память **Xdata**

Микропрограмма в A51 (B—преобразование 2/симв)

```
   Xseg  at 0   ;сегмент данных в Xdata
        Yy: ds 8      ;char  xdata yy[8]
        i equ r0
cseg  at 0x10
       mov i,#8
      mov dptr,#(Yy+7)
   // yy[i]= (x&0x01) | '0' ; x=x>>1;
cikl2: mov a,x
       anl a,#01
       orl a,#'0'
       movx @dptr,a  ;wrx
       mov a,x
       rr  a
       mov x,a
       dec dptr
       djnz  i,cikl2  ;for(i=7;i>=0; i--)
         ;"11111011"строка в Xdata
       end
```

3)Программирование в C51 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 C51 и A51. 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"

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

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

4. Символьное (в ASCII) преобразование двоичного числа в шестнадцатеричное
   "01001001110" → "0x24e"

5. 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 с (**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.

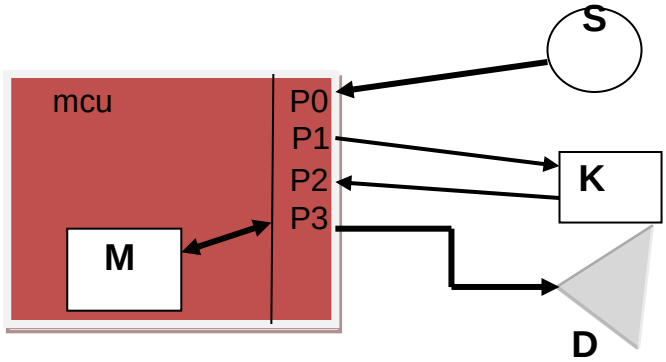


Рис.1.5. Устройства ввода и вывода данных

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**.

В ЭВМ применяются числа с двоично-десятичным (BCD) кодированием в форматах с **естественным размещением запятой**, целые или дробные двоичные коды чисел с фиксированной точкой, двоичные числа с плавающей точкой, двоичные коды.

1) BCD формат

2

Двоично-десятичный код				Десятичный код
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

1	1	0	0	+ плюс
1	1	0	1	-- минус
1	1	1	0	, запятая

Формат применяется в калькуляторах как естественная запись данных при вводе и выводе, а время вычислений с одиночными данными в реальном времени не критично.

В высокопроизводительных компьютерах (IBM360-370) в **экономической системе команд** применение BCD кодирования с форматом переменной длины связано с простыми вычислениями, большим объемом данных, широким диапазоном значений.

Суммирование и вычитание выполняются с высокой скоростью последовательно в двоично-десятичном сумматоре с коррекцией суммы **Hex**-цифр по правилам

1. При переносе бита в старшую цифру необходимо к сумме цифр добавить корректирующее значение 6 = 16 — 10, где 16- значение переноса в следующий разряд в двоичном суммировании тетрад.
2. Если сумма цифр больше 9 и нет переноса в следующий разряд, необходимо к сумме добавить корректирующее значение 6 (необходимый при этом десятичный перенос имеет в Hex-суммировании значение 16.
3. При вычитании двоично-десятичных цифр, если формируется заем из старшей цифры с значением 16, необходимо вычесть избыток 6.

2

Пример сложения :

3927_{10} = 0011 1001 0010 0111$_{BCD}$
4856_{10} = 0100 1000 0101 0110$_{BCD}$

```
                          1101
                          0110
                    Ci=1 0011
                    0010
                    0101
                    1000
               0001
               0110
          Ci= 1 0111
          0011
          0100
          1000
```

8783 = 1000 0111 1000 0011

Суммирование тетрад в С51

```
 char a, b,s; //BCD-коды цифр
sbit Ci; //перенос входной
if  (a+b+Ci >9)  { s=(a+b+6+Ci)mod16; Ci=1};
    else  {s=a+b+Ci; Ci=0;}
```

Пример вычитания

4856_{10} = 0100 1000 0101 0110$_{BCD}$

3927_{10} = 0011 1001 0010 0111$_{BCD}$

```
                          1111
                          0110
                    Ci=1 1001
                    0010
               1111
               0110
          Ci=1 1001
          0000
```

929_{10} = 1001 0010 1001

Вычитание тетрад в С51

```
char a, b,s; //BCD-коды цифр
sbit Ci; //заем входной
if  (a-Ci<b)  {s =a-b-6-Ci;  Ci=1;}
```

else {s =a-b-Ci; Ci=0;}

2) Числа с плавающей точкой

В языке **C51** приняты форматы

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 C51 применимы операции **(+,-,*,/)** и функции стандартной библиотеки **math.h**

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

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

3) Числа с фиксированной запятой.

Числа различаются как

целые – запятая условно фиксирована после младшего бита формата и

дробные – запятая фиксирована перед старшим разрядом формата.

В настоящее время общепринято использование целых чисел – соответствующую арифметику в виде системы команд называют стандартной.

Целые можно рассматривать как дробные с масштабом $\mathbf{2^n}$ в n-разрядном формате.

Отрицательное число в дополнительном коде (дополнение до двух **(2–|x|,** |x|<1.0**)** для дробных , n=8-битовый формат с фиксированной точкой, единица в старшем бите обозначает знак (-), нуль обозначает (+).

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.

Используется следующее определение формата

typedef uchar unsigned char;

typedef uint unsigned int;

uint y;

uchar x;

В C51 к числам применимы элементарные знаковые операции **(+, -, *, /, %)** и "беззнаковые" **(+,-,*,/, %,<<,>>),** возможное при этом переполнение формата не контролируется.

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

В A51 к числам применимы

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 P2

PSW==C.AC.F0.RS1.RS0.OV.-.P содержит признаки результата арифметических операций – **C**(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 C51 форматы с фиксированной точкой **целые.**

Числа с фиксированной запятой характеризуются **абсолютной погрешностью усечения Δ =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 A51 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.

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

uchar str[]=”библиотека”;

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

К кодам в форматах **char,int,long** применимы логические поразрядные операции {&&, ||, ~}

5)Структура 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

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.

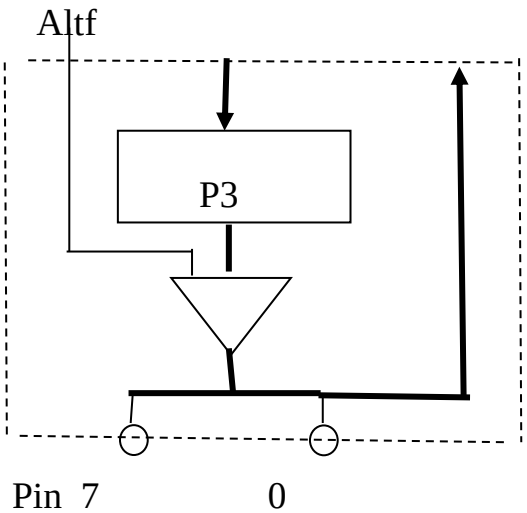


Рис. 1.4.. Структура квази-двунаправленного порта

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

3

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

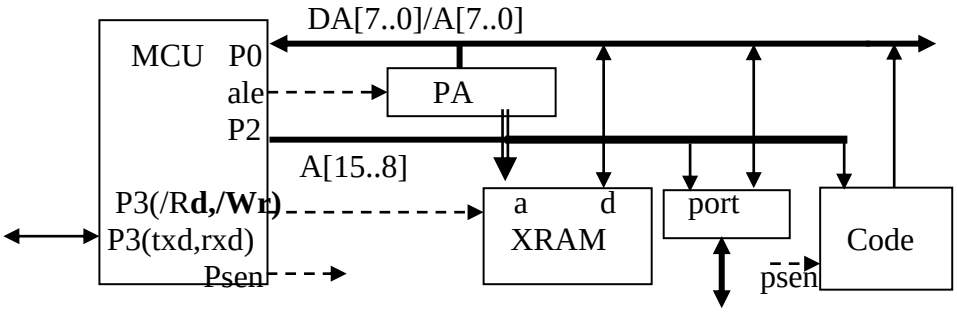


Рис.1.5. Мультиплексированная системная шина

Передача адреса по шине P0 синхронизируется сигналом **ale,** который записывает адрес в регистр адреса PA. 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 $\mathbf{2^n>10^m}$ 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 . При вычислениях форматы округляются до ближайшего с избытком - при m=2 получим n=8.

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 (M > 1) и перейти в диапазон целых значений с меньшей точностью (большим масштабом).

A/ Преобразование unsigned целых при вводе с портов

Форматы ввода двухразрядного целого десятичного числа с 8-разрядного порта (m=2)

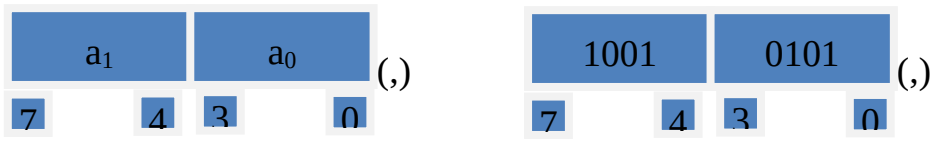


Рис. 2.6. Формат двоично-десятичного целого числа в порте.

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

3

При вводе десятичного числа $N=a_1a_0$

$$N=B_2=(a_1*10+a_0)_2$$

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

```
F2d(uchar  x){
   return  ((x/10)<<4) | (x%10); }
```

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

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

А) Ввод дробных чисел.

Информация (**количество N**) в **записи** (**m**)-разрядного дробного десятичного числа $A=0,a_1a_2a_3\ldots a_m$

$$N=\sum_{i=0}^{m-1} a_i d^{-i} = a_1 10^{-1} + a_{-2}10^{-2} + \ldots + a_m 10^{-m} \qquad (2.8)$$

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

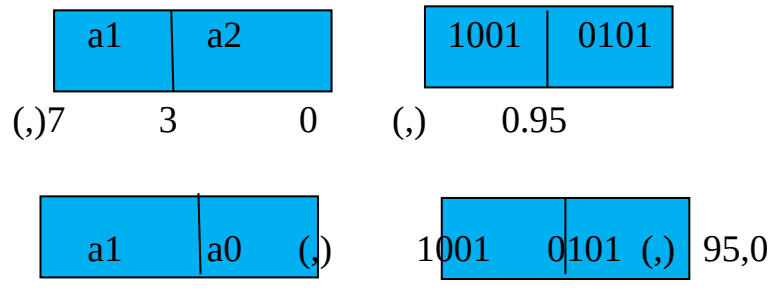


Рис. 2.8. Форматы двоично-десятичного числа .

Десятичное дробное число $A= 0,a_1a_2$ в 2-разрядном формате можно рассматривать как целое $A'= a_1a_0$ с масштабом 10^2 .

Тогда, применяя метод преобразования целых (2.7), получим (**n=8)**-разрядное двоичное целое число B_2
Для получения дробного двоичного 8-разрядного числа необходимо разделить целое A’ на масштаб Q=$\mathbf{B_2/10^2}$ **в** двоичной системе. В целочисленной арифметике для этого требуется сначала выполнить масштабирование с двоичным масштабом ($\mathbf{B_2*2^8}$) **и** Q= $\mathbf{B_2/10^2}$ = ($\mathbf{B_2*2^8}$)/$\mathbf{10^2}$ = $\mathbf{Q_2*2^8}$, чтобы результат деления оказался дробным двоичным в масштабе $\mathbf{2^8}$

```
unsigned int B2;
unsigned char  b2;
main()
{
 {B2=[((P1&0xf0)>>4)*10 + (P1&0x0f)]<<8; //дробное B2 в масштабе M=100*2^8
  b2=B2/100; //двоичное дробное в масштабе 2^8
  }
```

Преобразование десятичной дроби в двоичную (n=8) при округлении усечением имеет погрешность $\mathbf{2^{-8}}$
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[.**

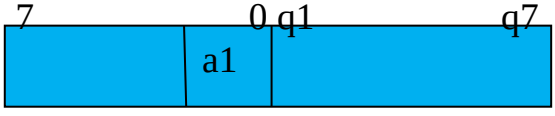
Б) Вывод дробных чисел

Если известно двоичное $\mathbf{Q_2=N=S_n=0,a_1a_2a_3…a_m}$, то из (2.8) неизвестная двоично-десятичная цифра a_1 = N*10 - целая часть двоичного произведения

$$\mathbf{Q_{n-1}=a_1, a_2a_3..a_m}$$

Последняя цифра в этом преобразовании 2/10 может иметь погрешность не более $\mathbf{10^{-m}}$.
Преобразования дробных чисел из одной системы счисления в другую в фиксированных форматах в общем случае приближенные.
Умножая двоичное целое в масштабе 2^n на основание 10, получим $\mathbf{a_1}$ в **(n..n+3)** разрядах целого произведения.
В произведении 10* ($\mathbf{B_2*2^n}$)= $\mathbf{a_1, a_2a_3..a_m*2^n}$ сохраняется масштаб $\mathbf{2^n}$ и положение запятой.

```
 {   uint  y;
     y=b2 *10;
  P2=((y&0xf00)>>4);          // a1- старшая десятичная цифра
 P2 | =(((y&0xff)*10)&0xf00)>>8; } //a2  цифра
```

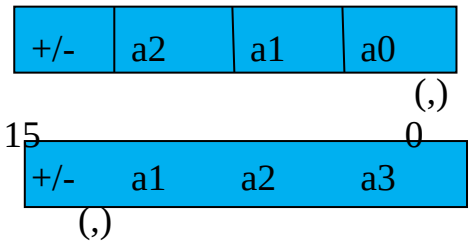


(,)

q1..q7 – условно разряды дробной части, 7..0 –разряды целой части при умножении на основание d=10

Задание.

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



В машинном формате **int** старший бит выделен для целой части (знака), полученный при переводе модуль дробного числа следует сдвинуть на один бит вправо.

2.3. Арифметические операции с фиксированной точкой [Орлов]

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 сложения и вычитания ALU.

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

Программа в С51 используется для демонстрации и отладки алгоритма.

Микропрограмма в А51 имеет отношение к аппаратной реализации и позволяет показать исполнение алгоритма на уровне близком к схемотехнике.

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 $z>2^8-1$ для целых или $z > 1.0-2^{-8}$ для дробных. Переполнение контролируется признаком переноса C=1 в PSW.

char x,y,c;

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

$(2^8-|c|)$ для целых. В 8-разрядных форматах эквивалентно 0-|c|, $|c|\le 2^7$ для целых и $|c|\le 1$. Старший бит двоичного кода совпадает со знаком.

Признаки результата при сложении

x>0,y >0 z=x+y , положительное переполнение , контролируется битами **(OV &¬C)** в PSW

x≥0,y<0 z= x+2-y = 2-(y-x) <0 , (y<x), если (¬OV)&C

x<0,y≥0 z=2-x+y=2-(y-x)<0, (x<y), если (¬OV)&C

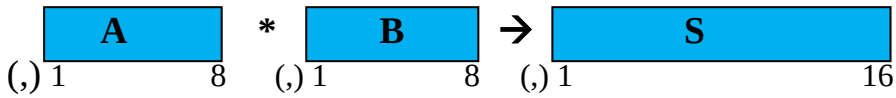
x<0,y<0 z=2-x+2-y=2-(x+y) , отрицательное переполнение, если **OV& C**

2.3.2. Умножение

1)Умножение двоичных дробных чисел в C51

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



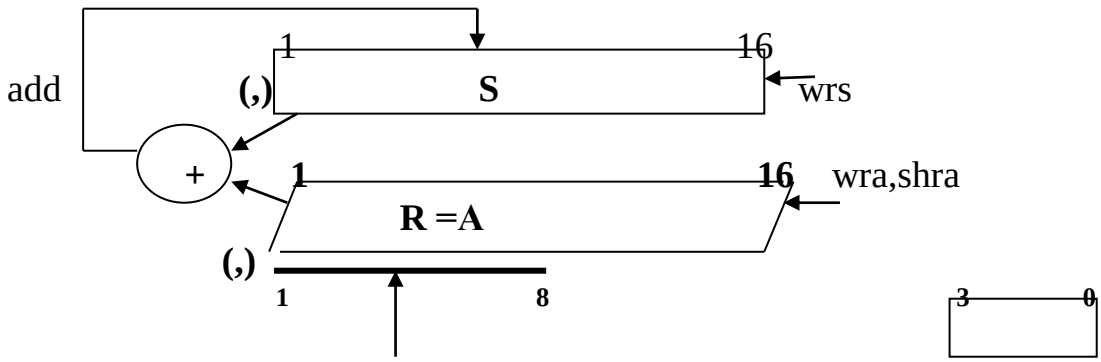
Приведение полиномиальной формы представления чисел к итерационной для вычисления произведения

A) Вычисление с общим членом ряда со стороны старших разрядов множителя

$$S=A*B=A*(B=0.b_1b_2..b_n) = A*(b_12^{-1} + b_22^{-2} .. +b_{n-1}2^{-n+1}+b_n2^{-n})=$$
$$= Ab_12^{-1} + Ab_22^{-2}.. +Ab_{n-1}2^{-n+1} + Ab_n2^{-n} = \sum R_ib_i$$

(2.1) $R_{i+1} = R_i*2^{-1}$ $i=1,2,..8;$ $R_0=A$

Схема вычисления в C51



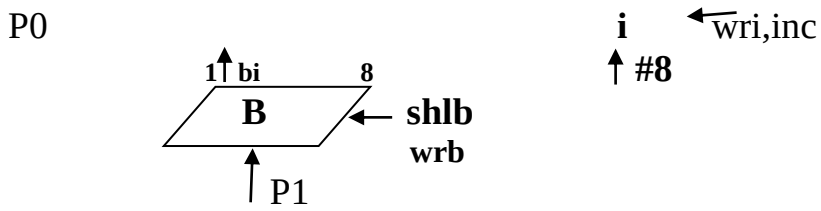


Рис.2.1.Схема умножения с общим членом ряда для дробных чисел в С51

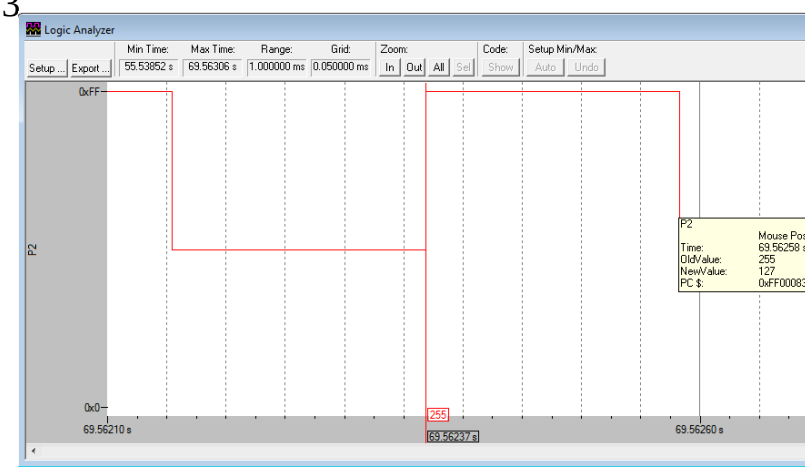
Требуемые ресурсы памяти – 16-разрядные регистры S и A. Регистр A со сдвигом вправо, 16-разрядный сумматор, 8-разрядный регистр B со сдвигом влево, счетчик циклов.
Если **bi=1**, то **S=S+ Ri*bi** - разрешение записи (**wrs=1**). Регистр R=**Ri>>1** сдвигается вправо (управление **shra=1)**
В регистре **B** множитель сдвигается влево (**shlb=1**), в старшем разряде контролируется текущее значение бита **bi**.

Программа в С51

```
#include <reg51.h>

 unsigned int S,A;



main()
 { unsigned char i,B=P0;
   A=P1<<8;
   S=0;
  for(i=0;i<8;i++)
    { A>>=1;
if(B&0x80) S=(S+A);
B<<=1;}
   P2=S>>8;
   P3=S;
}
```

Измерение времени выполнения операции.
Для измерения времени 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

3



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

В) Вычисление по схеме Горнера со стороны младших разрядов множителя

$$S=2^{-1}*(Ab_1 + 2^{-1}(Ab_2 +\dots + \_2^{-1}(Ab_{n-1} + 2^{-1}(Ab_n +0))..)=>$$

$$S_0=0 => S_1=2^{-1}(S_0+Ab_n) \rightarrow S_2= (S_1+Ab_{n-1})2^{-1}$$

(2.2) $S_{i+1}=2^{-1}(S_i +Ab_{n-i})$, $S_0=0$, $i=0,..n-1$, $B=\{ b_1,b_2,.., b_{n-i} ,..,b_n\}$

Рекурсивная функция $S(i+1)=(S(i) + Ab_{n-i})2^{-1}$, $S(0)=0$.Схема вычисления.

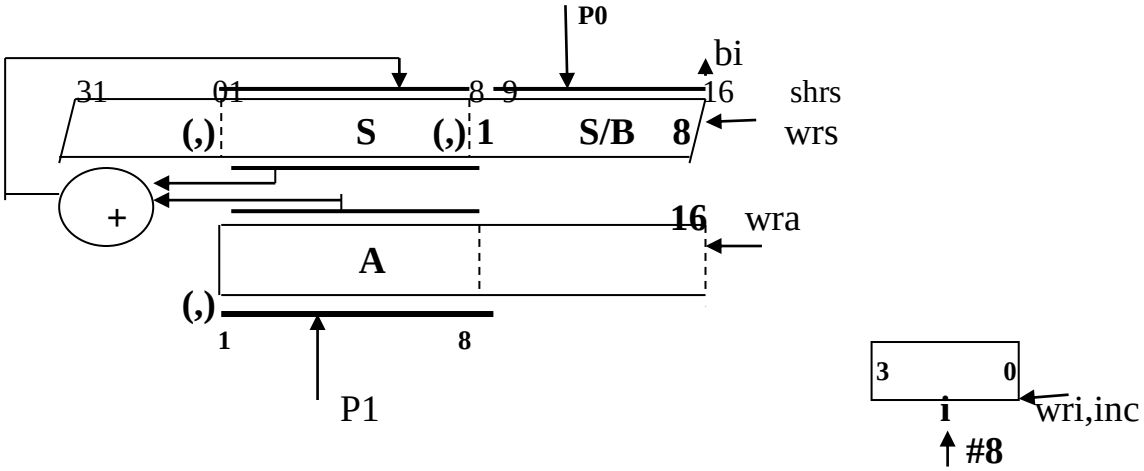


Рис. 2.2. Схема умножения дробных чисел по схеме Горнера в С51

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

3

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

16-битовый формат **$A2^8$** согласован с положением запятой в формате 16-битового произведения .

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

```
#include <reg51.h>
char  i;
unsigned  int A;
long S;
main()
{
  A=P0<<8;
  S=P1;
  for(i=8; i>0; i--)
 {S= (S&1)?  (S+A)>>1 : S>>1;  }
  P2=S>>8;
   P3=S;
  while(1);
}
```

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

$$2^{(-2n)} <= \Delta < 2^{((-n)-1)}$$

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 $\Delta= 2^{(-n)}$**),** если в дальнейшем не используется деление дробный делитель.

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

$$S=A*B=A*(0,B)*2^n= (0,A)*(B)*2^n$$

Например, для формулы (2.1)

$$S=A*B=A*(B=b_{n-1}b_{n-2}..b_0) = A*(b_1 2^{n-1} + b_2 2^{n-2} .. +b_{n-1}2^1+b_n 2^0) =$$
$$A*(b_1 2^{-1} + b_2 2^{-2} .. +b_{n-1}2^{-n+1}+b_n 2^{-n})2^n=$$
$$= (Ab_1 2^{-1} + Ab_2 2^{-2}.. +Ab_{n-1}2^{-n+1} + Ab_n 2^{-n})2^n = (\sum R_i b_i)\ 2^n$$

Умножение целых с общим членом ряда со стороны младших разрядов

$$S= A*(b_{n=1}2^{n-1} + b_{n-2}2^{n-2} .. +b_1 2^1+b_0 2^0) =$$
$$= Ab_{n-1}2^{n-1} + Ab_{n-2}2^{n-2} .. +Ab_1 2^1+Ab_0 2^0 = \sum R_i b_i$$

(2.3) $S= \sum R_i b_i\ ;\ R_{i+1}=R_i*2,\ i=1,2,..8;\ R_1=Ab_0$

Схема вычисления в С51

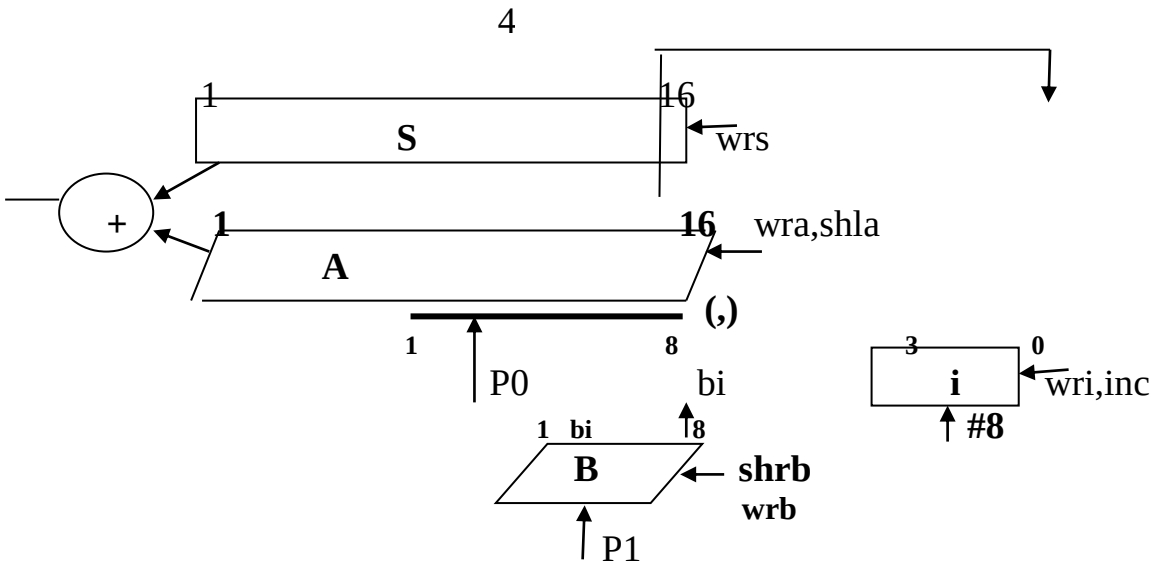


Рис.2.3. Схема умножения целых чисел с общим членом ряда в С51

Требуемые ресурсы – 16-разрядные регистры S и A. Регистр A 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.

```
 #include <reg51.h>
 char  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
}
```

3) Микропрограммное умножение в A51.

Размещение операндов в команде **mul ab : B**-множитель**, Aa**-обозначает аккумулятор **A**
Требуется вспомогательный рабочий регистр **wrk** для записи множимого.

4

Если использовать для вычислений схему (2.2) в A51, то она оказывается наиболее простой с микропрограммным управлением.

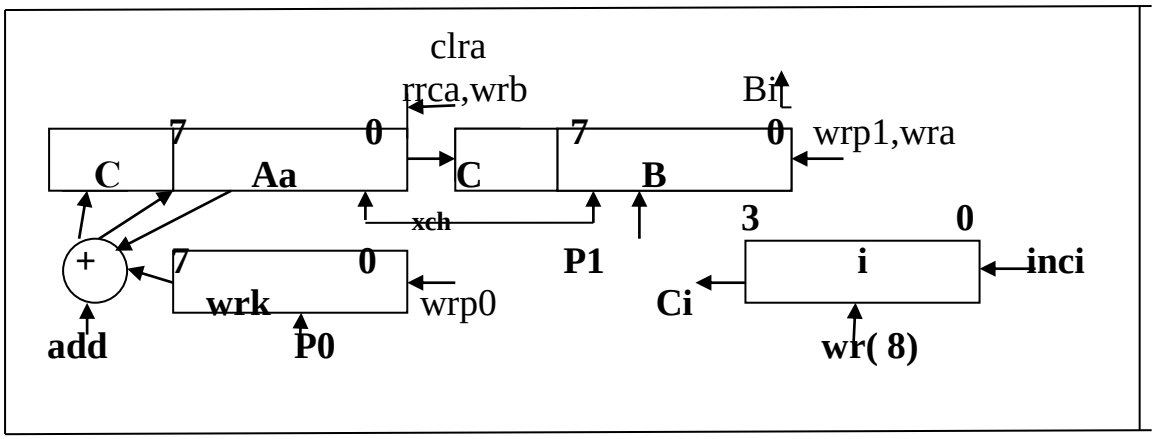


Рис.2.4. Схема умножения с микропрограммным управлением

Пара 8-разрядных регистров **Aa.B** вместо 32-разрядного S, для множимого используется 8-разрядный регистр **wrk** вместо **16**-разрядного **A**
Управляющие микрокоманды в A51: **clra→**Aa=0, **wrp1→**B=P1, **wrp0→**wrk=P0,**wr(8)→**i=8, **inci→** инкремент счетчика. При сдвиге **rrca** признак **C** записывается в старший разряд Aa, а млаший разряд Aa запоминается в **C**.
Ci-перенос со счетчика~(i==0).
xch={wrb,wra}→{Aa=B, B=Aa} в этой микрокоманде совмещены две записи по разным шинам.
Микропрограмма A51 для тестирования умножения 8*8 .

```
          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
```

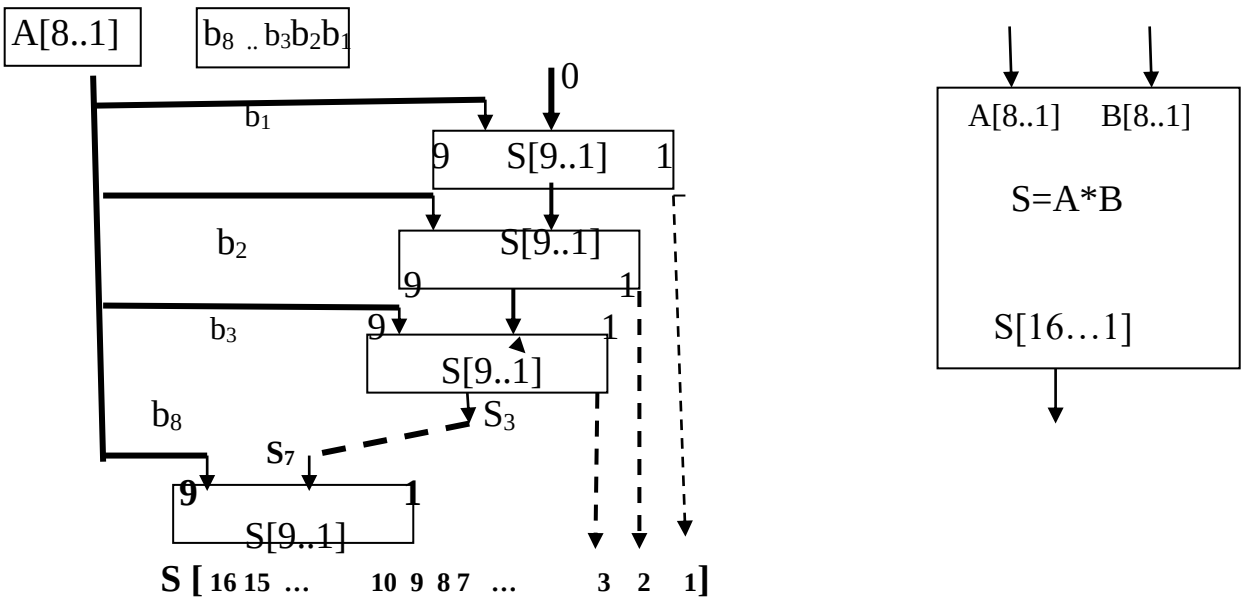
Объем программы Code=32 байта, время выполнения программы 90 мкс

4) 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

			A4	A3	A2	A1		
			1	0	1	0		b_1 =1
		1	0	1	0			b_2 =1
	1	0	1	0				b_3=1
1	0	1	0					b_4=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 однотактной параллельной схеме.



4

Рис. 2.5. Матрица умножения 8*8 →16

S[9..1] – девятиразрядный сумматор, 9-ый бит формирует перенос в следующий разряд.
Разряды множителя b_8-b_1управляют вентилями, которые подключают множимое A[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.

A) 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= (2^n  - 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
```

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

$$(sx.x)*(sy.y)= sx*y*2^n+x*y+sy*x*2^n+sx*sy*2^{2n}$$
$$= (-2^n)*(2^n-y)*2^n+(2^n-x)*(2^n-y)+(-2^n)*(2^n-x)*2^n$$

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

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

$$\mathbf{S=A*B=S_n=2^{-1}(S_{n-1}+Ab_i)}$$ и делимое S меньше делителяA.

Тогда цифра частного b_i =1 при условии, что $\mathbf{S_{n-1} = 2S_n-A >=0}$, иначе b_i =0. Изменяя нумерацию остатков S_{i+1} при вычитании $\mathbf{2S_i-A}$, приходим к следующей рекуррентной формуле деления

(2.3) $\mathbf{S_{i+1}= 2S_i-A}$ **и** $\mathbf{b_i=1}$ **, если** $\mathbf{2S_i-A >=0}$**, где** $\mathbf{S_0}$**=S-делимое**

$\mathbf{S_{i+1}= 2S_i}$ **и** $\mathbf{b_i=0}$ **, если** $\mathbf{2S_i-A <0}$**,**

Схема деления в С51

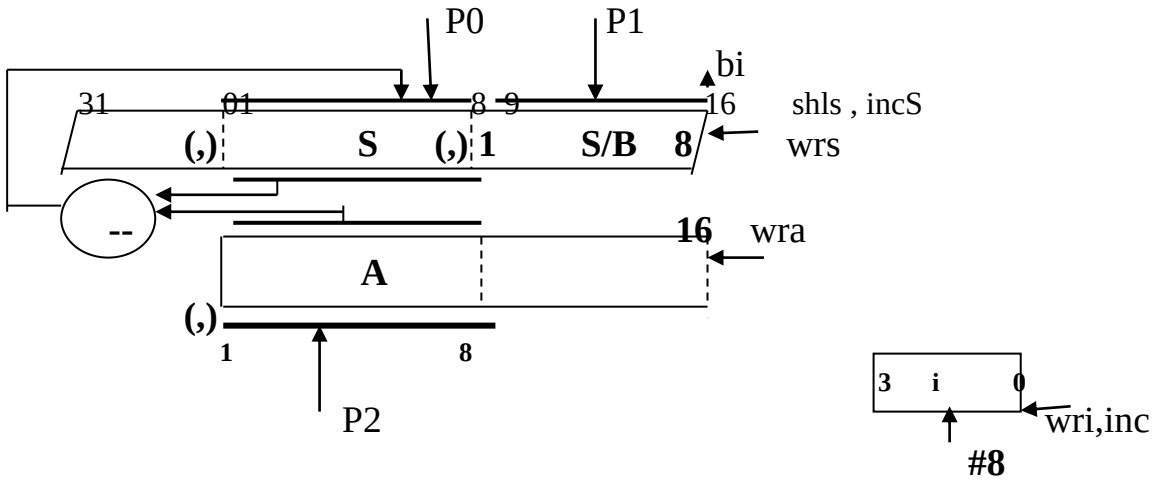


Рис.2.5. Схема деления дробных.

Делимое **long S**, 16-разрядный делитель **A**

4

Регистр **B** -частное совмещено с младшими разрядами делимого при сдвиге влево **(shls=1)** . 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=1 и формируется 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 C51.

```
#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]
}
```

2) Микропрограмма в A51

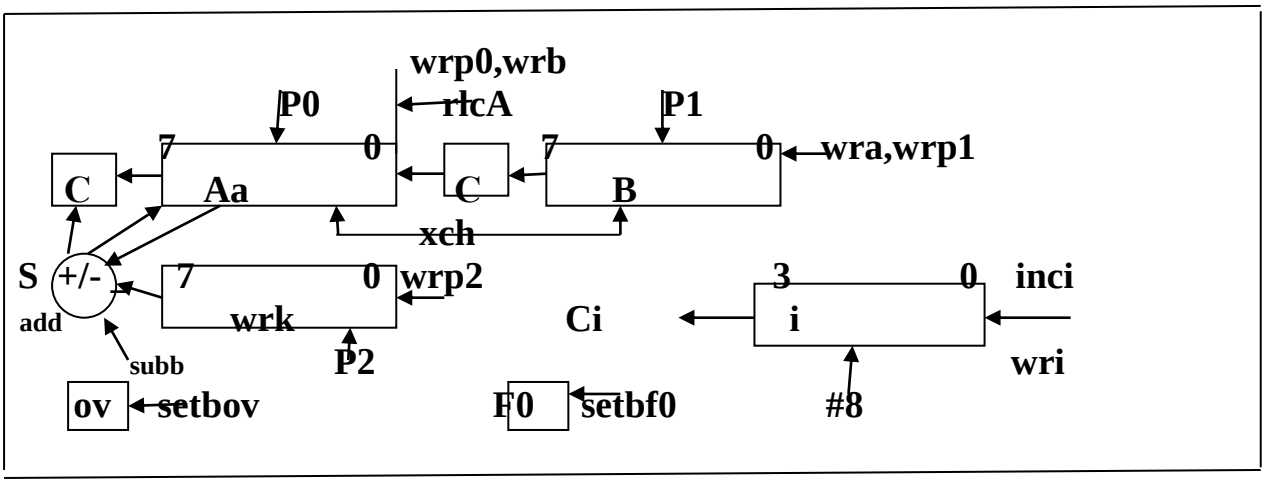


Рис.2.6. Схема деления 16/8 в А51

Используются регистры
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 (с.А**), 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 вычитание00
         jc m1
          setb ov
          jmp out
 m1:  add a,wrk  ;восстановление остатка
        clr c
cikl: djnz i,m2  ;for (i=0 ; i<8; i++ )
       jmp out
   ;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 bit(F0,C)
                        ;    0    0    1
                        ;    0    1    0     ;восстановление остатка
                        ;    1    0    1
                        ;    1    1    1
```


4

```
        anl  C,/F0
        jc m1  ;бит частного 0  -  ;восстановление остатка
         setb c
        jmp cikl  ;бит частного 1

out:    mov P3,B    ;частное
         sjmp $
        end
```

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

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

Задание.

Уточнить микропрограмму деления в А51 с учетом возможности исключения явного восстановления остатка.

Схемы деления (2.6) и умножения (2.4) совместимы. Для сдвига 16-разрядного произведения и делимого используются команды **rrc a, rlc a** и команды **xch A,B,** где обменивается содержимое регистров (A*B→B.A и (A←→B))

1) Функциональный элемент деления

В целом делении формула (2.3) трактуется как вычисление разности $S_{i+1}= 2S_i-A$ из предположения, что **b_i=1 , если $2S_i-A >=0$** .

Признак положительной разности – инверсия **заема** при вычитании в старшем разряде или **перенос true** при суммировании с дополнительным кодом делителя.

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.

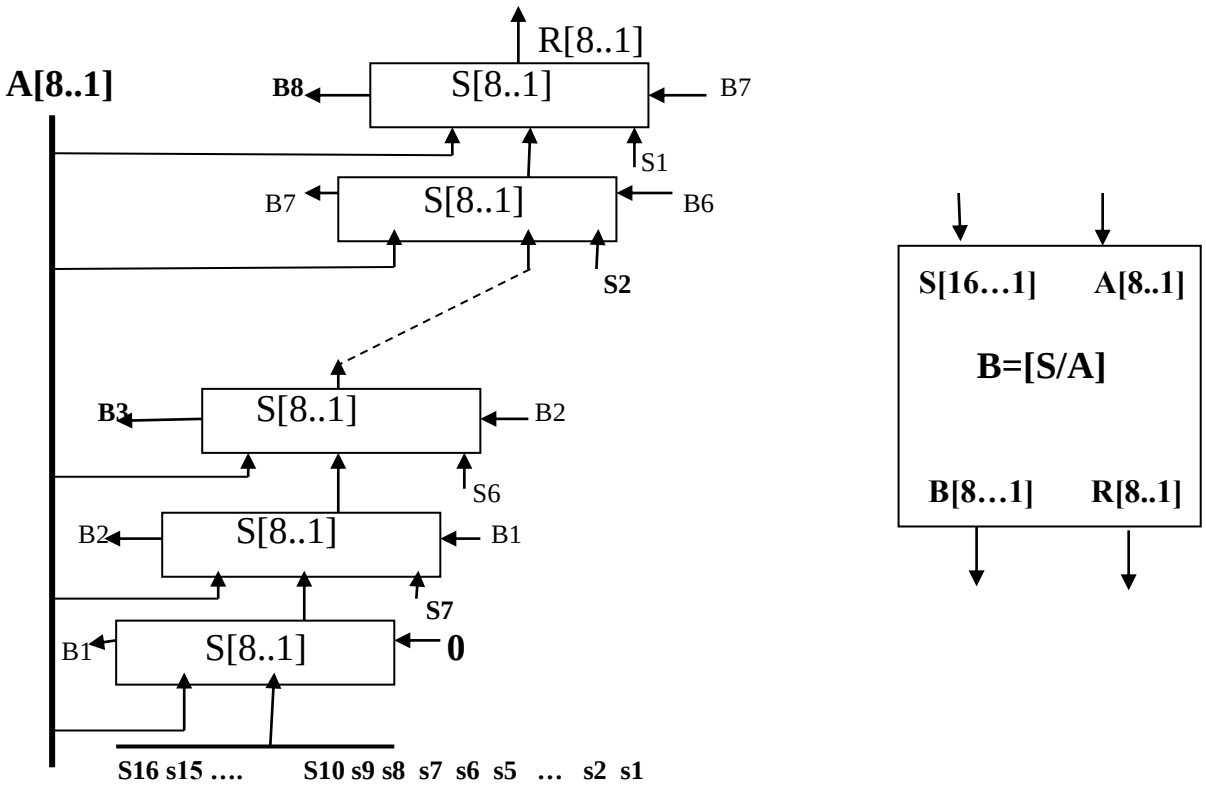
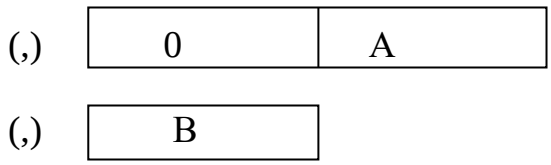


Рис.2.7. Матрица деления 16/8→8

Делимое S[16..1], делитель A[8..1], B[8,,1]-частное, R[8..1]-остаток
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 S[8..1] и делителя B[8..1] в mcs51 можно представить дробными в форматах



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

4

Задание.
Совместить программы последовательного умножения и деления по схеме Горнера и с общим членом ряда в С51. 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.

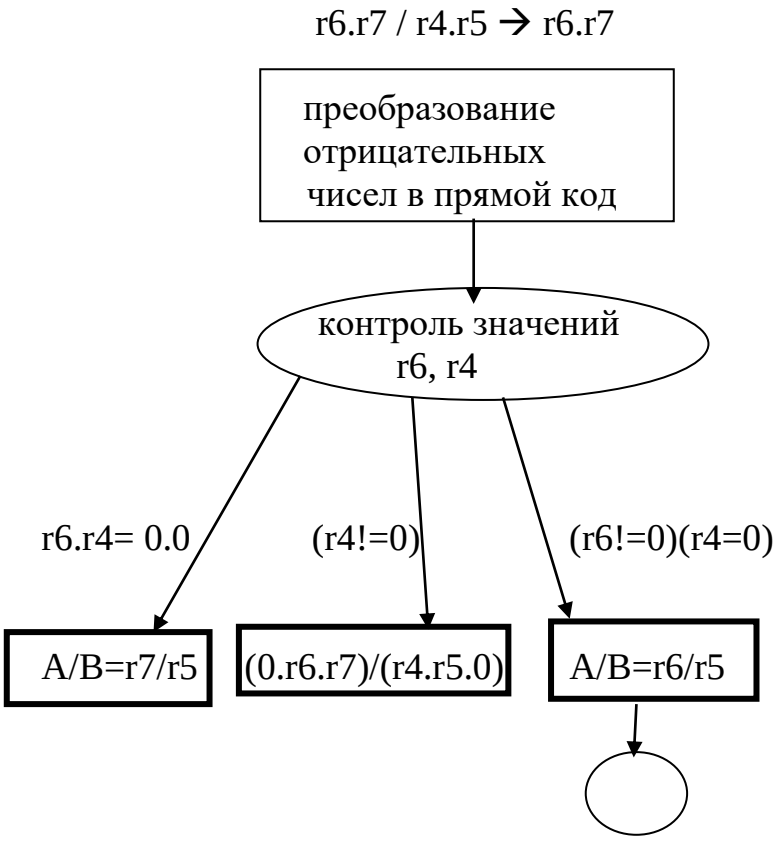


Рис 2.8. Диаграмма целого деления со знаком в С51

2.3.4. Извлечение 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 $0.B=\sqrt{0.S_0}$.

Пусть i+1-ое приближенное двоичное значение корня $x_{i+1}= x_i\, b_{i+1}$ и b_{i+1} - младшая двоичная цифра в этом приближении , S_0-дробное подкоренное

значение не равно 0, x_0=0-начальное -целое значение дробного корня, b_{i+1}-текущая двоичная цифра корня.

На первом шаге $\mathbf{S_0>=(x_0 +0.b_1)^2 =(x_0 +0.1)^2 =x_0^2 +x_0+0.01,\ x_1 =0.b_1}$ **и** $\mathbf{b_1}$**=1-старшая цифра дробного корня, если**

$$\mathbf{S_1=S_0-0.01 >=0}$$

Пусть $\mathbf{b_1=1}$ **и** $\mathbf{S_1>=0,}$ тогда на втором шаге сдвинем остаток $\mathbf{S_1}$ на 2 разряда влево и значение корня $\mathbf{x_1}$ на один разряд влево – обозначим новое значение этого остатка $Q_1.S_1$, где Q_1-целая часть и S_1-дробная часть и $\mathbf{x_1= b_1.}$

Предположим $\mathbf{x_2= b_1,b_2}$

$Q_1.S_1>=(x_1 + 0.b_2)^2 =$ $\mathbf{x_1^2 +x_1+0.01}$ **и** $\mathbf{b_2}$**=1, если**

$$\mathbf{Q_2.S_2=4Q_1.S_1-x_1.01=x_1^2 >=0}$$

Пусть $\mathbf{b_2=1}$ **и** $\mathbf{Q2.S_2>=0,}$ тогда на третьем шаге сдвинем остаток на 2 разряда влево – получим $Q_3.S_3$, где Q_3-целая часть и S_3-дробная часть и корень сдвинем на один разряд влево $\mathbf{x_2= b_1,b_2}$ и получим $\mathbf{x_3= b_1b_2,b_3}$

$Q_3.S_3>=(x_2 + 0.b_3)^2 =$ $\mathbf{x_2^2 +x_2+0.01}$ **и** $\mathbf{b_3}$**=1, если**

$$\mathbf{Q_3.S_3=4Q_2.S_2-x_2.01=x_2^2 >=0}$$

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

(2.5) $\mathbf{Q_{i+1}.S_{i+1}=4Q_i.S_i-x_i.01=x_i^2}$ **и** $\mathbf{b_{i+1}}$**=1, если** $\mathbf{4Q_i.S_i-x_i.01 >=0}$

$\mathbf{Q_{i+1}.S_{i+1}=4Q_i.S_i=x_i^2}$ **и** $\mathbf{b_{i+1}}$**=0, если** $\mathbf{4Q_i.S_i-x_i.01 <0}$

5

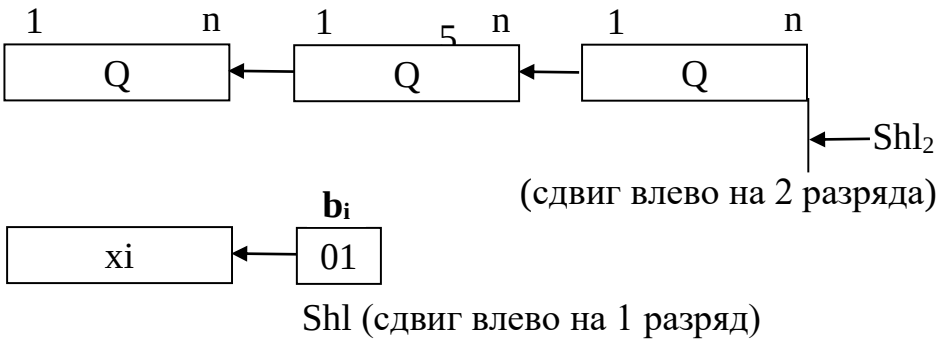


Рис. 2.7. Схема извлечения корня квадратного без восстановления остатка.

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 $\partial = \mathbf{2^{(-n)}}$ **(**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 C51, вычислить значения sin(x) в диапазоне аргумента 0-360° (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 2^8

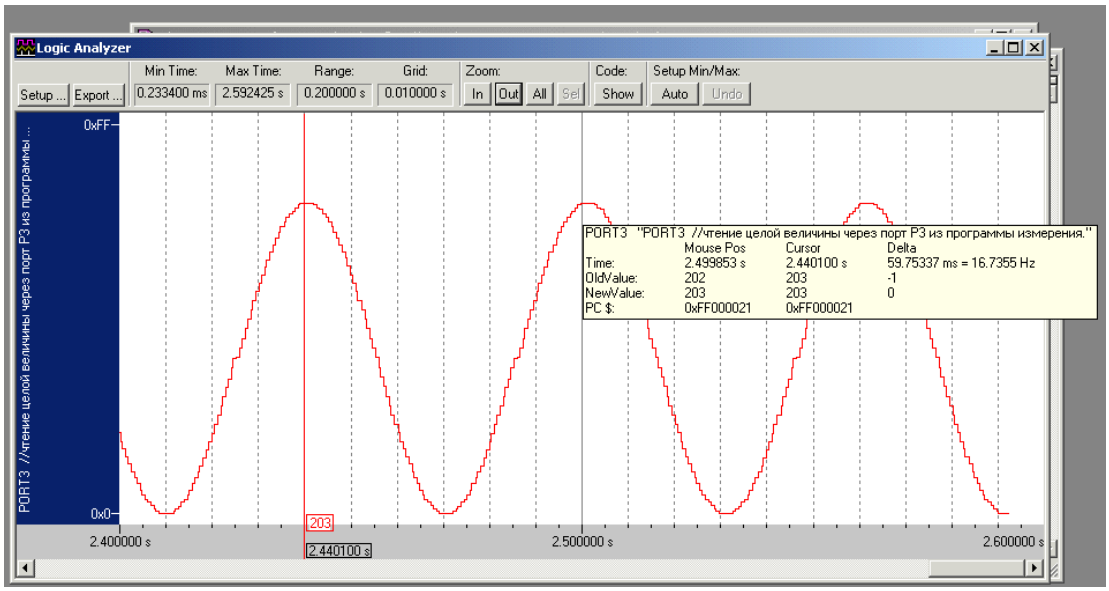
5

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;
main()
{
  while(1)
      for(x=0; x<6.28 ;x+=0.0628)
                  y= sin(x) + 1.0;
            }
```

Для измерения времени можно использовать функцию **Analizer** в Кейл. График функции, вычисляемой с плавающей точкой, в окне Анализатора в диапазоне опорнонго напряжения 5в.



5

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

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

1) Вычисления по схеме Горнера

Преобразование полинома приближения

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

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

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

……..

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

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

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

А)Масштаб m=100

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

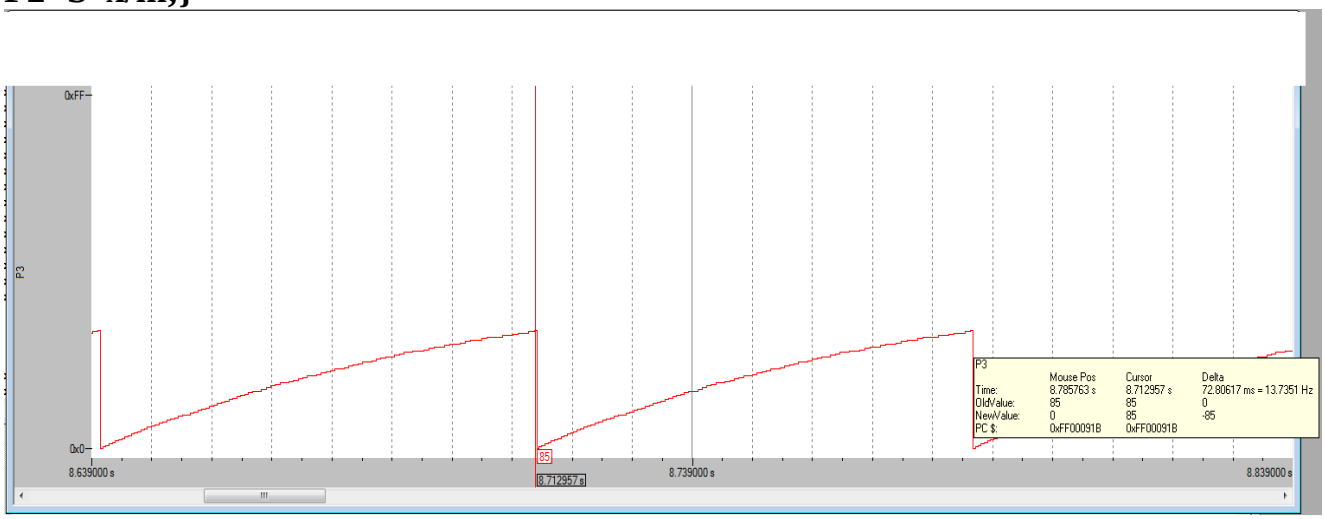


Рис.2.10. График значений sin(x) с приближением в масштабе m=100 в С51

x – значение аргумента в диапазоне [0 – 100], от 0 до 1.0 радиан.
Объем программы 320 байт, среднее время 0.72мс

Б)Вычисления с двоичным масштабом $m=2^8$

```
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; } }
```


5

При вычитании в формате байта используем значение единицы 0xff в масштабе 2^8. Если вычитаемое не равно нулю, можно добавить единицу для повышения точности.
Объем программы 97 байт, среднее время вычисления 0.11мс

Преобразование программы в A51

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 ; S=Si( 42)
       mov S,#0xff
       call Si
      mov ai,#20 ; S= Si(20)
       call Si
       mov ai,#6  ;  S= Si(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**

2) Вычисления с общим членом ряда

$\sin x \sim x/1 - x^3/3! + x^5/5! - x^7/7! = \sum Si \rightarrow$

$S_0=(-1)^0 x \rightarrow$

$CS_1=S_0(-1)^1 x^2/6 \rightarrow$

$S_2=S_1(-1)^0 x^2/20 \rightarrow$

$S_3=S_2(-1)^1 x^2/42 \rightarrow$

……..

$S_{i+1}=S_i(-1)^i x^2/(2i(2i+1))$, i=1.. S0=x,

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

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

A) Масштаб m=100

```
#include <reg51.h>
  typedef   unsigned char uchar;
   char   i,x,y,m,n;
   char S,S1;
   char Si()
{   return ((S*y)/m)/(i*(2*i+1)*2)  ;}
main()
{   m=100;
 while(1)
  for(x=0; x<m; x++)
     {y=(x*x)/m; S=x; n=5;
       for(i=1;i<n;i++)
        {S= (i%2)? S-Si() : S+Si();
       P2=S;
           }
}
```

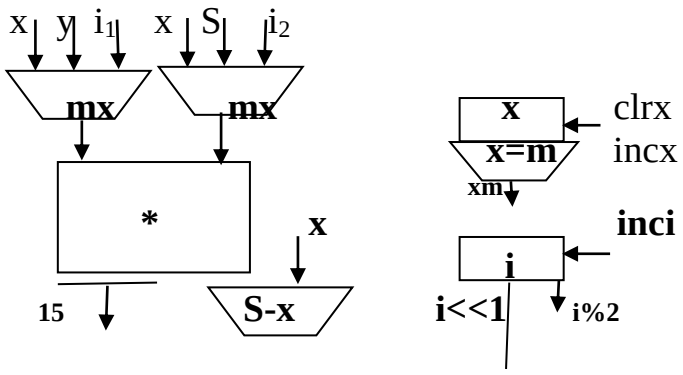
Оценка абсолютной погрешности

5

```
#include <reg51.h>
typedef   unsigned char uchar;
 char   i,x,y,m,n;
 char S,S1;
 char Si()
{   return ((S*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=5;
       for(i=1;i<10;i++)
        {S= (i%2)? S-Si() : S+Si();
         if (i<5) S1=S;}
       P2=(S-S1)*10;
           }
}
```

Б) Вычисление sinx с общим членом ряда с масштабом $m=2^8$

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



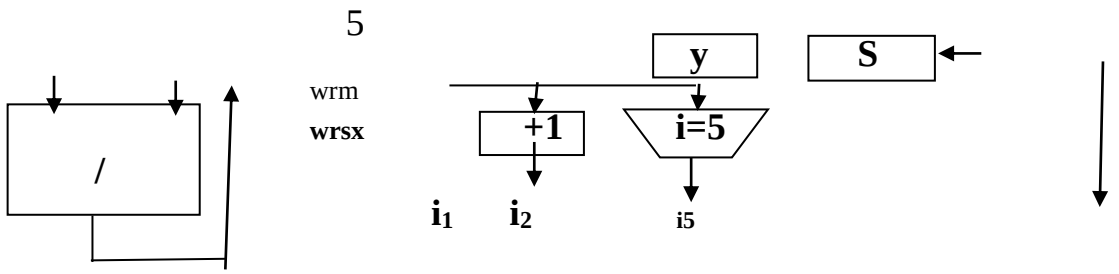


Рис.2.11. Схема вычисления sin(x) в С51

Выполнить в С51 вычисления с плавающей и фиксированной точкой по схеме Горнера и с общим членом ряда с масштабами m=100 и **$m=2^8$**
Измерить объемы программ и время исполнения.
С общим членом ряда выполнить программу с масштабом $m=2^8$ в А51, показать зависимость положительной погрешности вычислений от числа членов ряда.

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

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

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

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

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

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

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

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

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

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

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

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

5

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

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

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

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

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

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

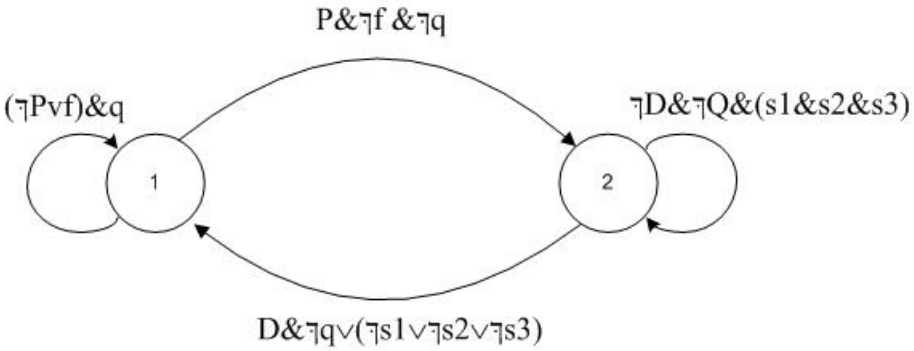
2.5 Логические данные

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.

Пример конечного автомата, который представляет алгоритм управления пресса [логика]



Работу конечного автомата можно записать системой уравнений, определяющих логику управления

Q1 = Q1&(¬P v f) v Q2& ¬D&q v (¬s1 v ¬s2 v ¬s3)

Q2 = (Q1&P&¬f) v Q2& D&(q&(s1&s2 &s3)

Q1 и Q2 две двоичные переменные, определяющие состояние пресса:
Q1 = 1 обозначает начальное состояние;
Q2 = 1 обозначает состояние штамповки.

Применимы логические операции с битами **(&,|,~,^**), эквивалентные битовым командам (**anl,orl,cpl,xrl**) в A51**.**
Тип данных, доступный в C51 для ЭВМ с архитектурой MCS51.
В C51 и A51 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

В **C51:**
bit x1,x2; //абсолютные адреса битовых переменных в битовом поле Data (00-7f)
char bdata mem,meme ; //номера байтов (0-F) с битовой адресацией
sbit y1= mem^0; //0-ой бит 0-байта mem
int bdata mm; //0-int с битовой адресацией (0-F)
sbit y1=P1^2; //второй бит порта P1
PSW=C.AC.F0.RS1.RS0.OV.-.P - резервированные имена битов в регистре PSW с прямым доступом
Sfr с адресом кратным 8, бит-адресуемый в поле бит 0x80 – 0xFF
SFR m=0xF8; //бит адресуемый регистр SFR
sbit m5=m^5; //бит регистра

Доступ к битам в **C51**
bit x1,x2; **//**битовые переменные в поле бит Data
sbit z=P1^2; //бит **sfr** порта P1 (не смешивать с операцией ^)
sfr TT=0xc0; //бит-адресуемый регистр в sfr
sbit T=TT^0;

char bdata mem; //**char**(байт) **int, long** данных с битовой адресацией в сегменте //Data
s**bit y0= mem^0** ; //0 бит ячейки mem
s**bit y1= mem^1;**

```
main()
  { x1=1;
```

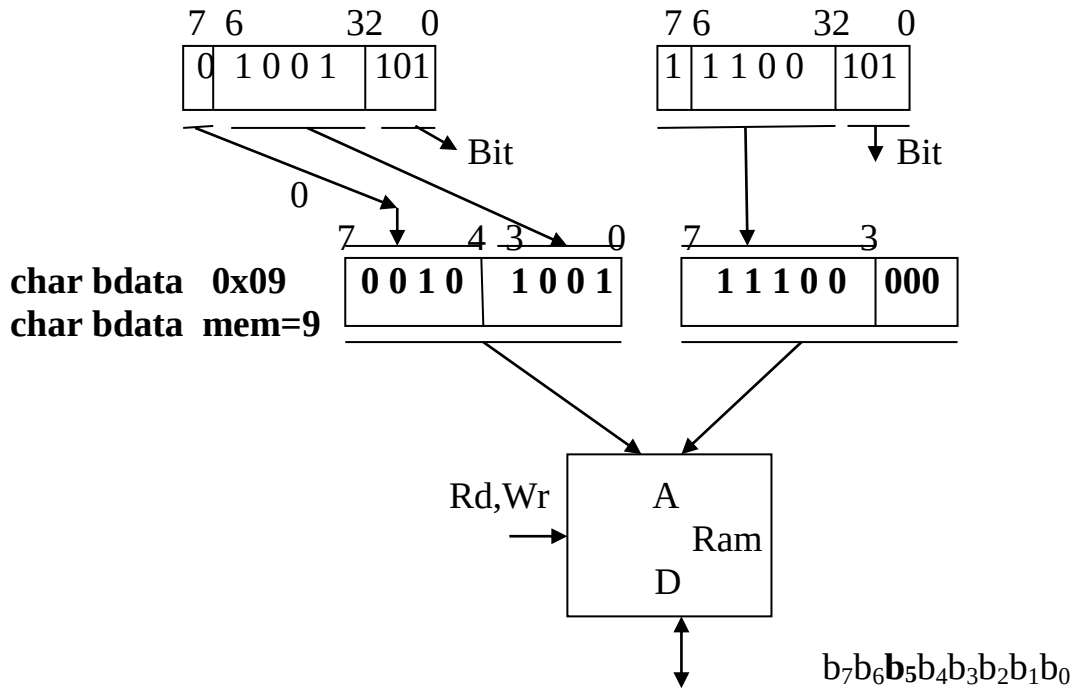

6

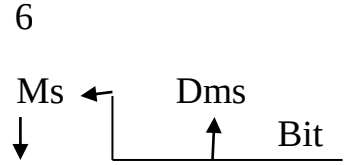
```
z=x1&y1;
 y2=z | x1^v1; }
```

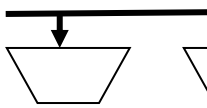
Адресация битов в **A51**

```
bseg at  0x06 ; абсолютный сегмент битов с 6-го бита сегмента битов
     y1: dbit 1    ;  y1=0x20.6=0x26
     y2: dbit  1   ; y2= 0x20.7=0x27

     y3:  dbit  2   ; y3= 0x21.0-0x21.1
      x1  bit P1.0  ;x1=0x90.0=0x90
      z2  bit acc.1  ;z2=0xe0.1=0xe0


xx equ 0x20
     y1 bit xx.0    ;y1=0x20
     y2 bit xx.1    ;y2=0x21
cseg at 0
     mov c,x1
     mov z2,c
     mov c,y1
     mov y2,c
 end
```

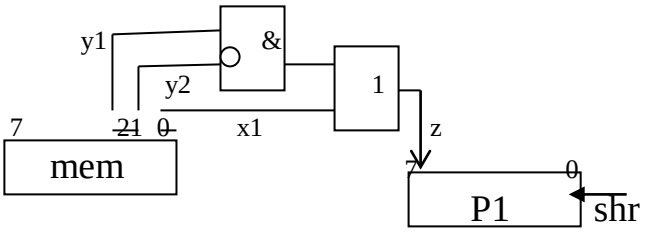
```
sbit  y1=mem^5;
bit  y1=0x04D=9^5
x4 bit  Acc.5= 0xE5
```







```
#include <reg51.h>
        char  bdata  mem=9;
        sbit x1=mem^0;
        sbit y2=mem^1;
        sbit y1=mem^2;
        sbit z=P1^7;
        main()
        { for(mem=0;mem<8;mem++)
           {P1>>=1 ; z= y1&!y2|x1;
                  }
                  }
```



7 6 5 4 3 2 1 0
1 0 1 1 1 0 1 0 **P1**

В **A51** 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 ACC

mov c, 0 ; бит C в PSW - прямой доступ
Data(20h.0)→ C , 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

```
dseg at  0x20
   mem: ds 1
       x1 bit mem.0
       y2 bit mem.1
       y1 bit mem.2
       z bit P1.7
     cseg at 0        ;z= y1&!y2|x1
```


6

```
      clr a
      mov  mem,a


   m1: mov c,y1
       anl c,/y2
       orl c,x1
       mov z,c

       mov a,P1
      anl a,#0xFE
       clr c
        rrc a
        mov P1,a
        inc mem
       mov R1,mem
        cjne R1,#8,m1
        nop
        end
```

Задания по разделу

1. $z=(y_1/x_1 \vee y_2x_2)(/y_1 \vee x_2)$
2. $z= (y_1 \vee /x_1)(y_2x_2 \vee x_1)$
3. $z=/x1(x2 \vee /x3) \vee x1x4$
4. $z=(x_1 \vee /x_2x_3)(/x_2 \vee x_4)$
5. $z= /y_1 \vee /y_2(y_1x_1 \vee /x_2)$
6. $z=(x_1 \vee /x_3x_4)(/x_1 \vee x_2)$
7. $z=/y_1x_2 \vee y_2(/x_1 \vee /x_2)$
8. $z=(/x_1 \vee x_2)(x_1x_3 \vee /x_4)$
9. $z=(x_1y_1 \vee /x_2y_2)(/x_2 \vee y_1)$
10. $z=(/x_1 \vee y_1)\ (x_2y_2 \vee /y_1)$
11. $z=y_1(/y_2 \vee /y_3) \vee /y_1y_4$
12. $z=(y_1 \vee /y_2y_3)(/y_2 \vee y_4)$
13. $z=/y_1\ y_2 \vee /x_1x_2(y_1 \vee /y_2)$
14. $z=x_1y_1 \vee /x_2(/y_2 \vee /x_1)$
15. $z=(x_1 \vee /x_2 \vee /x_3x_4)(/x_1 \vee /x_4)$

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

Применяются логические операции с предикатами (&&, ||, ==, !, !=) Операнды в следующем выражении могут иметь различные типы, но истинность для значений (0, не0) имеет смысл и выражение вычисляется с использованием эквивалентных по смыслу программ с условными переходами

по таблицам истинности.

char aa,bb,cc,dd,S;

S=(aa<bb)&&(cc!=dd)||(bb!=0) ~ x1 & x2 | x3, x1=(aa<bb), x2=(cc!=dd), x3=(bb!=0)

Возможная полная интерпретация предикатов в области натуральных чисел определяется таблицами истинности

aa	bb	**x1=(aa<bb)**	cc	d	**x2=(cc!=dd)**
z	1	d	1	D	d
1	z	d	-1	d	d
z	z	d			
-1	z	d		Bb	**x3=(bb!=0)**
z	-1	d		d	d

z={0,-1,1} – выбор значений из области интерпретации, d={0,1}

При выборе наборов используются признаки (z,d), в которых контролируются зависимости значений предикатов d от одного из аргументов по аналогии с D-алгоритмом тестирования [логика]

Таблица истинности

x1 x2 x3	x1 & x2 | x3	aa bb cc dd	d	
0 0 d	d	1 1 1 1	1	a1
		1 -1 1 1	1	a2
		1 0 -1 -1	0	A3
0 1 d	d	1 1 1 0	1	A4
		1 -1 1 -1	1	A5
		1 0 -1 1	0	A6
1 0 d	d	1 1 1 -1	1	A7
		-1 -1 -1 1	1	A8
		0 0 1 1	0	A9
1 d 0	d	1 1 1 0	1	A10
		-1 -1 0 0	0	A11
d 1 0	d	-1 0 0 1	1	A12
		0 0 1 0	0	A13
		-1 0 1 -1	1	A14

#include <reg51.h>

6

```
char code ab[14][5]={{1,1,1,1,1},
                     {1,-1,1,1,1},
    {1,0,-1,-1,0},

    {1,1,1,0,1},
    {1,-1,1,-1,1},
    {1,0,-1,1,0},
    {1,1,1,-1,1},
    {-1,-1,-1,1,1},
    {0,0,1,1,0},
    {1,1,1, 0,1},
    {-1,-1,0,0,0},
        {-1,0,0,1,1},
    { 0,0,1,0,0},
    {-1,0,1,-1,1}
        };
      char S,aa,bb,cc,dd,i,j,S1;
      main()
{   for (i=0;i<14;i++)
       for(j=0;j<5;)
         {aa=ab[i][j++]; bb=ab[i][j++];cc=ab[i][j++];dd=ab[i][j++];
S1=ab[i][j++];
          S=(aa<bb)&&(cc!=dd)||(bb!=0);
            }
    }
```

Для интерпретации предиката (aa<bb) необходимо учитывать знаки чисел и для контроля потребуются два набора значений переменных.
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| **7**-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 -старший бит в формате числа.
Следовательно,

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

2))для отрицательных чисел **a-b = (1.0+(1.0- 0.|aa|)) – (1.0+(1.0- 0.|bb|))= 0.|bb| -0.|aa|** и
если a<b, то формируется заем C=1 в старшем разряде и модули |bb| < |aa|

3))для чисел с разными знаками

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

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

Таким образом, для интерпретации арифметического предиката для любых знаков в A51 выполняется инверсия знаков операндов и вычитание (a-b). Если заем C=1 при вычитании, то (a<b) =1(true)

Приведен листинг программы после Disassembler'a и комментарии к ней

```
aa  equ  r1
bb  equ  r2
cc  equ  r3
dd  equ  r4
```

csed at 0 (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,#0   ; S=0
 M4:    MOV     S,R7         ; S=1
        Nop
        end
```

Задание. Построить таблицу интерпретации предиката . Программа в C51 и Disassembler с комментариями.

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 четное)
5.  (a<b)||(a>c)&&(d>a)
```


6

```
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!=0)||(b>c)&&(d<b)
15. (a<d)&(b!=c)||(d>=c)
```

2.5.3. Вероятностная логика

В работе [9] предлагается арифметика вычисления вероятности, эквивалентная 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(¬x=1) или P(x=1).
Эти состояния обозначают абсолютную “истинность” или “ложь”.

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

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

(3) P{(¬xi&¬xj)=1}=(1-Ri)(1-Rj)=QiQj

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

В этих определениях сохраняется смысл логических операций для дискретных значений **Ri,Rj є{0, 1}** и сохраняются таблицы истинности соответствующих логических операций
Очевидно, выполняются законы коммутативности и ассоциативности относительно конъюнкции.
Интерпретация некоторых законов алгебры логики

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

(6) правила де Моргана

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

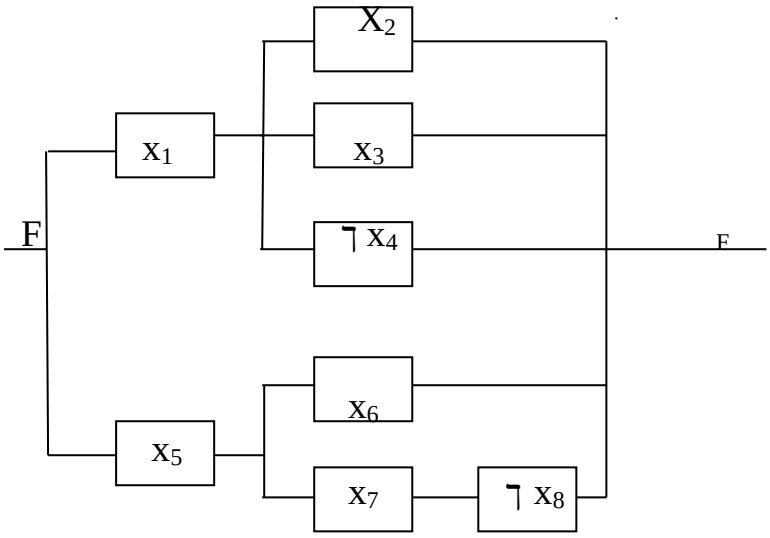
Эти законы позволяют преобразовать любую логическую формулу в ВФ

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

Булева функция может быть использована для описания структуры соединений блоков.

Пример

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



Переменные x_1 - x_8 интерпретируются как вероятности исправности блоков xi.

Выполняются для вычисления преобразования логической функции в вероятностную функцию ФВ.

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

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

= ¬ [¬(x_1(x2vx3v¬x4)) &¬ (x_5(x_6v (x_7¬x_8)))]=

= ¬ [¬(x_1¬¬ (x2vx3v¬x4)) &¬ (x_5 ¬¬ (x_6v (x_7¬x_8)))]=

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

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

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

Задание. Для заданного варианта функции выполнить ее преобразование в ФВ и вывести таблицу истинности S в порт. Значения False(0) и True(1) кодируются при вычислениях в байтах. Вероятность S<>{0,1}

6

Литература.
1. Новиков ГИ
2. Таненбаум
3.Орлов
4.Сташин В.В. Урусов А.В. Мологонцева О.Ф. Проектирование цифровых устройств на однокристальных микроконтроллерах, М: Энергоатомиздат, 1990.
5.Злобин В.К. Григорьев В.Л. Программирование арифметических операций в микропроцессорах, М:ВШ, 1991 г-303 с
6.Help в Keil (C51, Макроассемблер, Система команд MCS51).
7.Копченова Н.В., Марон И.А. Вычислительная математика в примерах и задачах, М:Наука, 1972, 367 с
8. Довгий П.С. Скорубский В.И. Организация ЭВМ Пособие к лаб. работам Изд. ГУ ИТМО 2009г-56 с.
9. Скорубский В.И. Поляков В.И. Зыков А.Г. Математическая логика, учебник, М:Юрайт=2016

Приложение 1

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

Арифметика и логика

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

Пересылки

mov a,{ri,@rj,#d,ad} a←{.....}
mov {ri,@rj},a {.....}← a
mov {ri,@rj},ad {.....}← ad
mov ad,{ri,@rj,#d,ad,a} ad←{.....}
mov {ri,@rj},#d
mov dptr,#d16
movc a,@a+dptr a←Code(dptr+a)
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)

pop ad Data(sp-)←Data(ad0

orl a,{ri,@rj,#d,ad} a v {…}→ a

orl ad,{#d,a} a v {…}→ Data[ad]

xrl a,{ri,@rj,#d,ad} a v {…}→ a

xrl ad,{#d,a}

clr a

cpl a не(a)

rl a rol(a) p

rlc a rolc(a,c) c,p

rr a ror(a) p

rrc a rorc(c.a) c,p

da a коррекция (+,-)$_2$

orl ad,{#d,a}

xch a,{ri,@rj,ad} a←→{.....} xrl a,{ri,@rj,#d,ad}

xchd a,@rj a(3-0)←→@rj(3-0)

swap a a(3-0)←→a(7-4)

-

команды булевского процессора

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

Управление программой и ветвления

ljmp a16	PC←a16
ajmp a11	PC(10.0)←a11[10.0]
sjmp rel	PC+2+/- rel[6.0]
jmp @a+dptr	PC←a+dptr
jz rel	PC+2+/-rel[6.0],если (a=0)
jnz rel	....,если (a<>0)
jc rel	....,если C
jnc rel	....,если неC
jb bit,rel	PC+3+rel,если bit=1
jnb bit,rel	....,если bit=0
jbc bit,rel	...,если bit=1,bit<-0
djnz {ri,ad},rel	{}-1; PC+1/2+/-rel[6.0],если {}<>0
cjne {ri,@rj},#d,rel	rel,если {}<>#d
lcall a16	стек□pc, PC←a16
acall a11	..., PC(10-0)←a11[10.0]
ret	PC←стек
reti	PC←стек,tf□0
nop	пропуск

обозначения битов SFR

	7	6	5	4	3	2	1	0	адрес
acc	.	.	.	.	.	.	.	.	e0i
b	.	.	.	.	.	.	.	.	F0i
psw	c	ac	f0	rs1	rs0	ov	.	p	d0i
sp									81
dph									83
dpl									82
ie	ea	.	.	es	et1	ex1	et0	ex0	a8i
p0	.	.	.	.	.	.	.	.	80i
p1	.	.	.	.	.	.	.	.	90i
p2	.	.	.	.	.	.	.	.	a0i
p3	rd	wr	t1	t0	int1	int0	txd	rxd	b0i
ip	.	.	.	ps	pt1	px1	pt0	px0	b8i
tmod	gate1	c/t1	m1	m0	gate0	c/t0	m1	m0	89
tcon	tf1	tr1	tf0	tr0	te1	it1	ie0	it0	88i
th0									8c
tl0									8a
sbuf									99
th1									8d
tl1									8b
pcon	smod	.	.	.	gf1	gf0	pd	idl	87
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 аккумуляторе

f0- признак пользователя, rs1.rs0 - банк регистров

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

ad - адрес Data, имя регистра SFR

bit - адрес бита в поле битов 00-7f или в специальном регистре- 80-ff ,адрес образуется из собственного адреса регистра, к которому добавляется номер бита;

,разряд регистра acc.5, psw.0, ... ,(80i - адеса битов 80,...87 регистра 80)

7

обозначение бита smod,sm0,....
/bit - инверсия бита
rel - <метка>=смещение PC в доп коде

Приложение 2

Ассемблирование.

При разработке ассемблерной программы может быть использована программа в C51 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

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

PC=0: Jmp Start – запуск программы с адреса 0000
Таблица векторов прерываний
Таблица констант
Подпрограммы
Программа
End

Макроассемблер

A51 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.

```
rsadd   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)
```


7

В файле с расширением **.M51** приведено распределение памяти для проекта
Пример размещения данных в С51

```
#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

Распределение памяти Code

```
    0000      Jmp ?start
_____________________________
     Вектора прерываний.
        3      Jmp  int0
        b      Jmp  tm0
       13      Jmp  int1
       1b      Jmp  tm1
       23      Jmp  usart
_____________________________
            Станд библ С51
_____________________________
            Прикл библ
_____________________________
    int0:    обработчики
    tm0:      прерываний
    ……
```

```
_____________________________
 ?start      Startup51
             Jmp main
_____________________________
  Main:   основная программа
```

Распределение памяти Data с 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 переменные	0x08-0x17
стек	0x18 - 0x1F

бит-адресуемые ячейки	0x20 – 0x2F
локальные переменные	0x30 -

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.c (включить файл в проект)

Manage component →включить файл в проект)

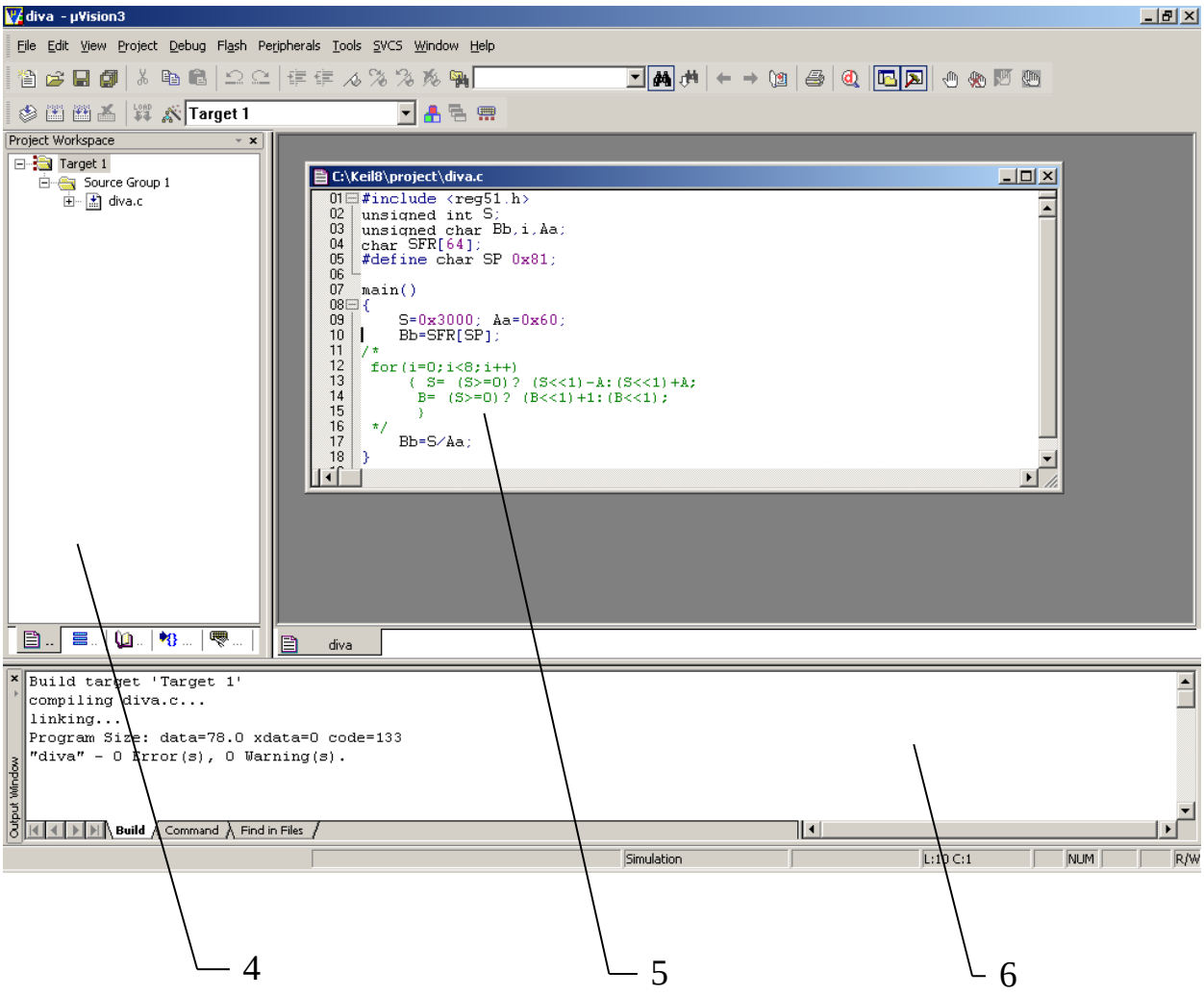
3)Синтаксический контроль.(**Compile**)

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

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

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

Окно Vision



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

Меню Vision

File Edit View Project Debug Flash Peritherial Tools SVCS Window Help

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

7

File

New - редактирование текстовых файлов
Open -
Close -
Save - все остальные имеют стандартное назначение

Edit - имеют стандартное назначение

Project

New → μVision project (создать проект)
Import
Open
Close

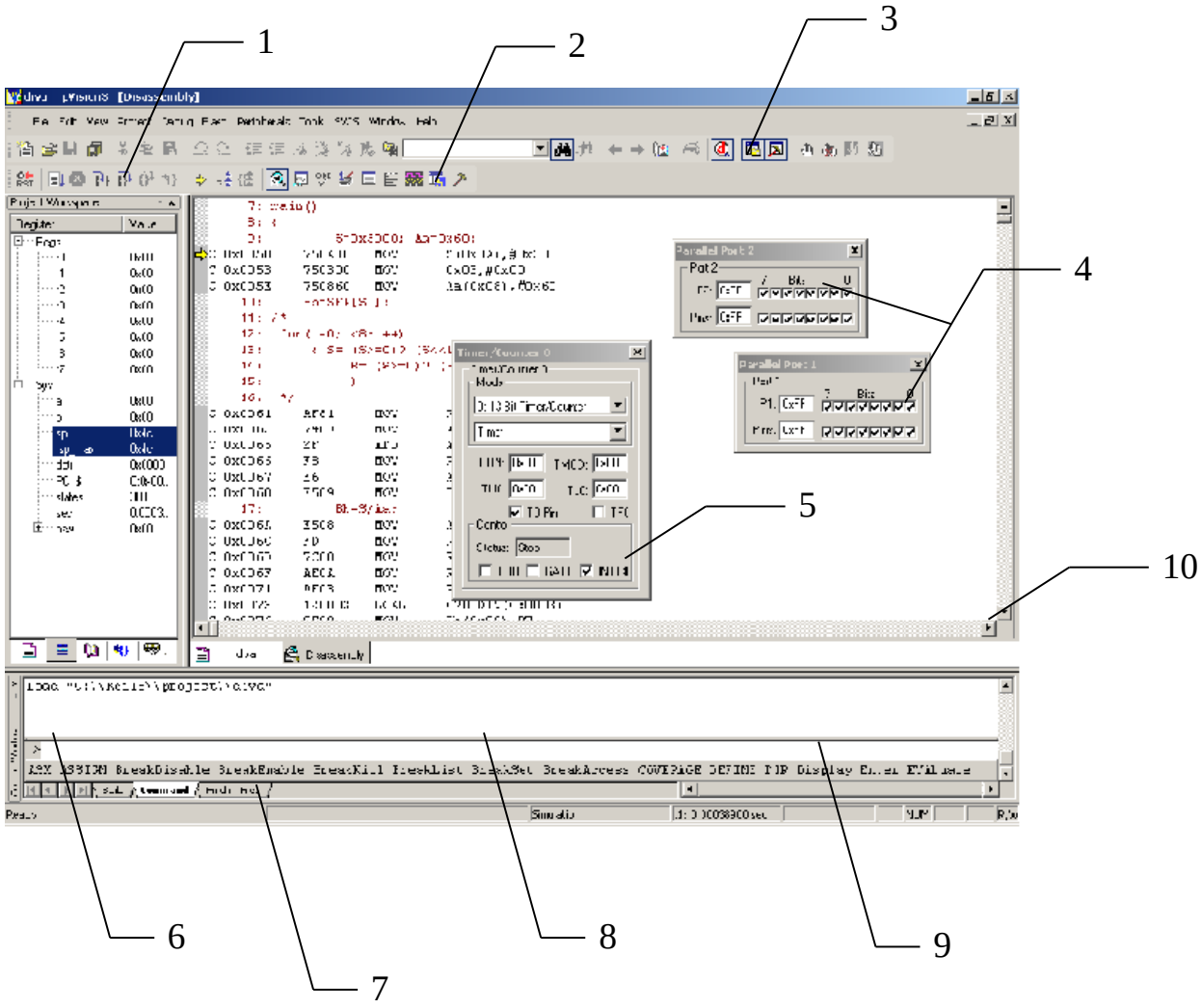
Manage → компоненты, окрестности (в проект включить файл)
Select Device → библиотека элементов
Options → настройки параметров компиляции и загрузки
Device – выбор модуля
Target - выбор частоты MCU
Out - вывод HEX-кода
List – вывод листинга .lst
C51 – размещение таблицы векторов
L51 – размещение программы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

Build - синтаксический разбор и линкирование
Translate –синтаксический контроль

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

Peritherial –активизируется после загрузки Project и
Содержит ссылки на периферию конкретной выбранной в проекте машины.

Окно Загрузчика (Debug)



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

6. Сообщения загрузчика.
7. Командная строка.
8, 9. Размещение окон Watch, Memory – выбираются в меню View.
10. Загруженный исполняемый файл в смешанной форме.

В меню Анализатора выбираем форму оценивания

Grafic **Perfomens Analizer**

В **Grafic** оценивание выполняется средствами, близкими к оцениванию осциллографом в виртуальном времени

7

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