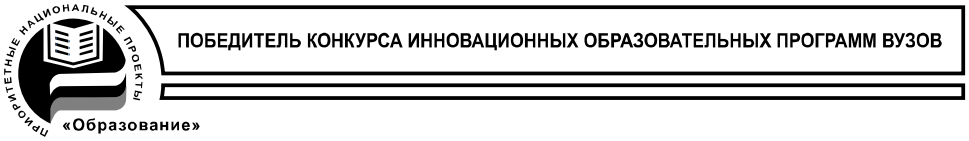
09.10.15
0,,,,,0,0,МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ



В.И. Скорубский

Организация микро-ЭВМ

с архитектурой mcs51/52 и программирование

Пособие к лабораторным работам

Часть I

Санкт-Петербург

2014

1

Скорубский В.И. Организация микро-ЭВМ пособие к лабораторным работам . – СПб: СПбГУ ИТМО, 2014г. – 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 Enverорment -IDE) Keil одноименного подразделения фирмы ARM.

Работы выполняются на двух уровнях – алгоритмическом с использованием языка C51 и уровне Макроассемблера A51. 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]

По проблематике и объему предлагаются несколько частей.- I, II, III

Часть I. В качестве основы для изучения различных вопросов организации и работы компьютеров используется **программная модель** микрокомпьютера MCS51/52, которая является промышленным стандартом и полезна как широко используемая и доступная в приложениях. Приводится краткое описание программной модели на высоком уровне (C51) и уровне микроархитектуры (a51)

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

СПбГУ ИТМО стал победителем конкурса инновационных образовательных программ вузов России на 2007-2008 годы и успешно реализовал инновационную образовательную программу «Инновационная система подготовки специалистов нового поколения в области информационных и оптических технологий», что позволило выйти на качественно новый уровень подготовки выпускников и удовлетворять спрос на специалистов в высокотехнологичных отраслях науки. Реализация этой программы создала основу формирования программы дальнейшего развития вуза до 2020 года, включая внедрение современной модели образования.

Содержание

стр.

3

Введение

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

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

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

Алгоритмическое решение задач и применение в проектировании встроенных систем на MCU поддерживаются высокоуровневыми средствами описания алгоритма на **алгоритмическом языке**, где представлена только программная модель MCU как ЭВМ общего назначения с традиционной организацией.

Для встроенных применений преимущественно используется алгоритмический язык Си, который позволяет сосредоточиться на алгоритмах, где операции, типы данных и средства управления унифицированы. Этот уровень представления MCU определяем как **Программную модель высокого уровня**.

Детали построения и работы MCU представлены системой команд на уровне **Ассемблера**, под которой подразумевается **микроархитектура** ЭВМ, включающая регистровую память, принципы организации доступа к данным в иерархии памяти, операции с данными в фиксированных двоичных форматах. Этот уровень представления MCU определяем **Программной моделью в Ассемблере**. Представление программ в Ассемблере, по-существу, являются **семантикой** алгоритмического языка (классическое название — **операционная семантика**).

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

5

Программная модель MCU является спецификацией при конструировании новой машины в **технологии ASIC**. Огромное число конструктивных решений не позволяет использовать какое-либо обобщенное представление об архитектурах MCU. 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 MCS51/52** (фирма Intel 1980)., считавшаяся промышленным стандартом в приложениях 80-90-х годов

Выбор микроконтроллера MCS51/52 [1] фирмы Intel обусловлен

1) популярностью ,

2) открытой и простой программной моделью . На современном уровне в ПЛИС вся схемотехника размещается как ядро системы на кристалле

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

4) включает схемотехническую поддержку большинства классических принципов общей организации компьютера и внешних интерфейсов

5)наличием эффективных и доступных средств программирования и отладки.

Для изучения программной модели решаются задачи с использованием алгоритмического описания на языке C51 и на машинном языке Ассемблера A51.

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

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

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

В библиотеке KEIL представлены более 80 фирм в 2012 г, и некоторые из них предлагают более 200 различных типов MCU с архитектурой mcs51/52 - 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

Отладка и исследования моделей выполняются в лабораторных работах в системе Кейл с Симулятором и учебными стендами (КИТ)

В первой части рассматриваются:

- организация памяти и типы данных в программных моделях C51 и A51

- декодирование двоично-кодированных форм записи данных при вводе и выводе
- арифметические операции, их применение к целым и дробным числам.
- редактирование символьных данных и управление адресуемой памятью данных
- логика вычислений с битовыми и булевскими типами данных;

Во второй части:

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

При этом используются MCU Sab515 (Siemens,/Infinion), Aduc8xx (ADC) с ядром MCS52 , которые позволяют эмулировать некоторые из распространенных внешних интерфейсов.

Большой опыт использования в промышленности, в приложениях и образовании отражается в разнообразной литературе по архитектуре mcs51. **(**первое подробное описание на русском языке [1] 1990 г)

I. Программные модели mcs51

Программная модель ЭВМ представляет ресурсы памяти, состав средств ввода-вывода, доступные в алгоритмических языках. Применение стандартного языка Си (стандарт ANSI) к конкретным MCU имеет ограничения, поэтому используются модификации C51 с макрорасширениями, учитывающие свойства конкретной ЭВМ.

1.1.Организация памяти и форматы данных в C51

Си получил широкое распространение во встроенных применениях, в первую очередь, благодаря присутствию в нем как общезначимых аппаратно-зависимых понятий, так и возможности создания на функциональном уровне аппаратно-зависимых расширений. Такая модификация Си доступна для всех существующих моделей MCU (Avr, Pic, TMS, Intel, ...).

Для mcs51 фирмой Keil предлагается аппаратное расширение стандарта Си в C51 с ассемблером A51. Компиляция C51 в A51 позволяет контролировать операционную семантику операторов языка C51 и используемые ресурсы mcs51.

7

Диаграмма MCU(рис.1.1.) представляет доступную иерархию памяти и средства ввода-вывода.

Ссылаясь на архитектуру mcs51, в дальнейшем будем иметь ввиду **базовую архитектуру(ядро) [** Википедия **].**

Традиционно используется конструктивное разделение памяти на **внутреннюю** (в кристалле MCU) и **расширенную (внешнюю)**, где применяются дополнительные схемы памяти на печатной плате. В дальнейшем память в различных моделях MCU может быть частично или полностью интегрирована в одном кристалле MCU.

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

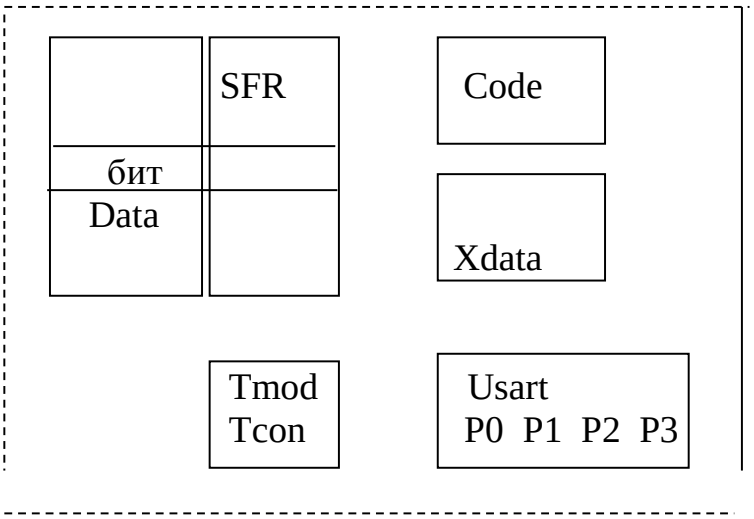


Рис. 1.1. Программная модель ЭВМ в C51.

В отличие от классической Неймановской модели (архитектуры) разделены адресные пространства памяти программ **Code** и данных **(Data , Xdata**).
В настройках компилятора выбирается одна из трех моделей организации памяти данных **Small, Compact, Large** для конкретных MCU.

1) Data – память данных типа Ram, режимы использования чтение и запись, объем 128 байт, быстрая по сравнению с другими типами памяти.
Часть ячеек памяти имеет побитовый доступ – поле **бит.**

2) Регистровая память **Sfr**(8-разрядные регистры специальных функций) – 128 байт адресное пространство. В C51 и A51 **резервированы** идентификаторы 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.(Приложение 1)

Все занятые регистры SFR ядра mcs51 определены в файле **reg51.h** и по умолчанию доступны в **A51.** В модификациях ядра mcs51 (SAB, ADUC,AT и др) регистровый файл **имя.h** определяется в библиотеке .**INC**.
Остальные регистры в SFR свободны и определяются для доступа в С51 как рабочие.

3) Расширенная память данных

xdata — 64 Кбайт адресное пространство, доступ к данным – чтение и запись

char xdata var; // доступ к переменной 16-битовым адресом
char pdata var; //доступ к переменной в странице из 256 байт

4) Code -постоянная (энергонезависимая) память программ– 64 Кбайт адресное пространство, доступ – чтение данных, чтение и исполнение команд Компилятор формирует исполняемый код в этой памяти, запись-загрузка выполняется специальными средствами и не возможна при исполнении программ.

При включении питания и сбросах программа стартует с адреса PC=0000, для установки начального состояния регистров совместно с прикладной программой по запросу подключается стандартный **Startup** –файл

Численные данные (количество) – основная форма представления информации в моделях самого разнообразного типа. ЭВМ предназначена для вычислений и работы с различными формами двоичного кодирования при хранении численных данных.

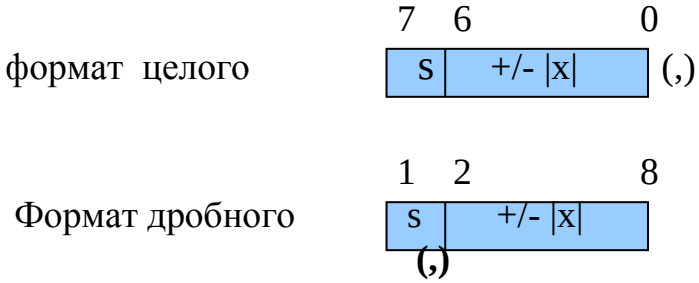
Тип данных определяет множество допустимых значений и применимые к ним операции преобразования. Кодирование различных типов данных двоичными словами, размещение и доступ к ним в памяти ЭВМ определяются **форматом**.

В описании данных предполагаются функциональное назначение (тип данных) и **машинное** представление (формат) в памяти ЭВМ и при выполнении операций .

1.Числа с фиксированной запятой.

char x=0x0e5; знаковое число, код символа, единица измерения объема памяти 8бит.

Отрицательное число в дополнительном коде (дополнение до двух ($2–|x|)|$ для дробных или для целых $(2^n-|x|) =(2–|x|)*2^{n-1}$, n=8-битовый формат с фиксированной точкой, $|x|<=1.0$ для дробных.



S- старший знаковый бит двоичного числа (0-положительное и 1-отрицательное число) можно рассматривать как целую часть дробного числа.
|x| - семиразрядный модуль числа
По умолчанию , в С51 все форматы с фиксированной точкой – целые. В А51 можем соответствующие форматы трактовать как дробные так, как удобно и проще их использовать в арифметических вычислениях.
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 С51 к знаковым числам применимы элементарные знаковые операции (+, -, *, /, %,<<,>>), возможное при этом переполнение формата не контролируется.
int x=0x0fabc; знаковое 16-битовое число ; 16-битовый формат с фиксированной точкой, отрицательное число в дополнительном коде
long x; 32 битовое знаковое число
unsigned char y=200; //беззнаковое двоичное целое число
Беззнаковые числа рассматриваются в тех же форматах как 8-разрядные положительные целые или дробные , операции (+, -, *, ,/, %,<<,>>)) тоже беззнаковые.

1)Арифметика 8 битовая двоичная (Знаковая/беззнаковая)
Формат числа – двоичное число в дополнительном коде. Операции сложения и вычитания знаковые и беззнаковые различаются только формированием признака переполнения OV[ ].

PSW==C.AC.F0.RS1.RS0.OV.-.P содержит признаки результата арифметических операций – **C**(перенос, заем), **AC** – полуперенос, **OV**(знаковое переполнение), **P**(бит четности)**, F0**(бит пользователя), **RS1-RS0** – номер активного регистрового банка.
add a, {Ri,@rj,#d,ad} ; a + {..} → a, Признаки C,OV,P в PSW, в скобках {.. } обозначены режимы адресации второго операнда

addc a, {Ri,@rj,#d,ad} ; a + {..}+ C → a
subb a, {Ri,@rj,#d,ad} ; a - {..}- C → a
add a,P2 ; a +P2→ a **P2-**регистр порта P2

2) 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 **v=(b#0), 0→C, P**
div ab ; a/b →a, b=rest(a/b) признаки **ov,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**

2. Двоичные коды эквивалентны беззнаковым числам

К двоичным кодам применимы поразрядные логические операции **(~,&,|,^)** и, если имеет смысл, то и арифметика.

char str[]="библиотека " строкой символов, кодируются в виде последовательности байтов ASCII-кода, в памяти (по умолчанию в C51) строка завершается кодом 0. В A51 необходимо явно указывать **нуль** в конце строки.
Набор стандартных функций **(cmp, cat, cpy,len,clr,spn,str….)** со строками в библиотеке **string.h**
Little Endian (LE)- доступ при чтении и записи в памяти
Char xx[]="abcde", где xx[0]='a'. xx[1]='b', …xx[4]='e'

3. Числа с плавающей точкой
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 применимы операции **(+,-,*,/)** и функции стандартной библиотеки **math.h**
В C51 доступно свободное преобразование чисел различных форматов и типов. что может иметь практический смысл, но и не безопасно. В языках с синтаксисом C++ часто не разрешены неконтролируемые преобразования типов (Java, C#)
4.Структура определяет древовидную иерархию данных, например, описание модуля MCU. В структуре могут быть определены элементы различного типа данных (char,int.long,float,..)

1

```
struct  mcu {              //имя типа структуры
            char name[8];
             int  format;
             int  pin;
             }MM;       //переменная типа mcu

    MM.name ="80c51BH";
```

5.Битовый тип данных Bit в С51.

Тип данных, доступный в С51 для ЭВМ с архитектурой MCS51.
Применимы логические операции с битами (&,|,~,^), эквивалентные битовым командам А51**.**

Биты упорядочены в поле , состоящем из 128 бит (16 байтов) в памяти **Data** и 128 бит в регистрах SFR

bit x1,x2; //определение битовых переменных в битовом поле Data (00-7f)
char **bdata** mem ; //ячейка в Data с битовой адресацией
sbit y1= mem^0; //0-ой бит ячейки mem

Часть регистров **sfr (ACC. PSW, P0,..)** с адресами кратными 8 – бит адресуемые: x1=ACC^1

Биты доступны по именам, определяемым явно или по умолчанию в соответствующих регистрах

sbit y1=P1^2; //второй бит порта P1

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

В С51 могут быть определены свободные по умолчанию регистры SFR

SFR sf=0x0FE;

и пары смежных байтов как 16-битные регистры,

Sfr16 TT=0xA1;

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

```
SFR   m=0xF8;
sbit   m5=m^5;
```

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

Применяются логические операции 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;
```

В С51, по существу, предполагается **прямой (безадресный) доступ**, идентификатор переменной обозначает значение безотносительно к режиму доступа.

В С51 (как и в стандарте языка С) используются средства косвенного адресного доступа к данным, что позволяет реализовать разнообразные методы доступа к сложным структурам в базах данных.

Пример интерпретации предиката (aa<bb)

!s.bb и !s.aa - двоичные числа в формате байта, где старший бит обозначает инверсию знака. Представим их как дробные числа, в которых старший бит целая единица.
Тогда

отрицательные **числа !s.bb=1- 0.|bb|= и !s.aa= 1-0.|aa|,** где |bb| и |aa| **7**-битовые модули

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

Следовательно,

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

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

3))для чисел с разными знаками

a-b = (1+ 0.|aa|) – (1- 0.|bb|)= 0.|aa| + 0.|bb| и перенос из разряда единицы в целой части С=0

a-b = (1 -0.|aa|) – (1+ 0.|bb|)= - 0.|aa| - 0.|bb| и перенос из разряда единицы в целой части С=0

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

Эквивалентная программа в А51

```
CLR     C
MOV     A,bb
XRL     A,#0x80         ; s.bb^1.00 =!s.bb
MOV     R0,A
MOV     A,aa
XRL     A,#0x80         ; s.aa^1.00=!s.aa
SUBB    A,R0            ;  !s.aa – !s.bb
JNC     M1              ;  if( aa<bb) goto M1
```


1

```
     MOV      A,cc
     CJNE     A,dd, M2        ;   if(cc!=dd) goto M2
 M1:     MOV      A,bb
     JZ      M3                  ;   if (bb==0) goto  M3
M2:    MOV      R7,#0x01
       SJMP     M4              ;goto M4
 M3:    MOV      R7,#0x00   ; S=0
M4:    MOV      S,R7          ; S=1
```

7.Адресный указатель pointer

В с51 необходимо различать, в какой области памяти размещается операнд, тип памяти, формат данных и в какой памяти размещается указатель

По умолчанию, указатель и операнд размещается в памяти Data

char * xx[] =”abcdef”; //адресация в Data

Обращение косвенное к байтам ***xx[0]=0x55;**

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

Указатель данных в памяти Code

char code *aa[]=”abcdef”; //указатель-адрес начала константы в **Data**

char xdata *aa[]; //указатель-адрес данных из **xdata**, указатель в **Dat**a

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

char xdata *xdata aa[]; //указатель-адрес данных из **xdata**, указатель в **xdata**

В С51 при компиляции с учетом разделения памяти по типам программа существенно усложняет по сравнению с косвенным обращением, принятом в А51.

Адресный указатель-целое число с точностью до байта.

К указателю применимы целые арифметические операции **(+, -, *)**.

Формируемый адрес при обращении к многобайтовым форматам должен учитывать физическое размещение этих форматов в адресуемой памяти

Примеры доступа к данным

- char xx[5]="abcde"; //размещение строки
- int *m; указатель
- char *zz=&xx[5]; присвоение значения по адресу
- int y= (int)&xx[0]; значение целого из двух байтов массива
- *m=0x55; значение константы по адресу
- y= xx[2]; чтение байта косвенное
- xx[5]=*m; чтение байта прямое
- y=(int)(&x[0]+1); чтение целого смещенное прямое
- y= *(int *)xx[3]; косвенное чтение с преобразованием адреса из неявного в прямой

1.2. Организация памяти в A51.

Программная модель в A51 определяет организацию памяти с **режимами адресации**, элементарные преобразования данных и управление программой в системе команд mcs51.

Система команд приведена в Help Keil [1] и Приложении 1.

ЭВМ выполняет 111 базовых команд. Время выполнения команд определяется циклом. Для упрощения расчетов при моделировании реального времени в Кейл выбирается частота синхронизации 12 Мгц, единица отсчета времени - **цикл** из 12 тактов(1мкс), команды выполняются за 1-3 цикла.

Симулятор Кейл позволяет выполнять измерения времени исполнения программ.

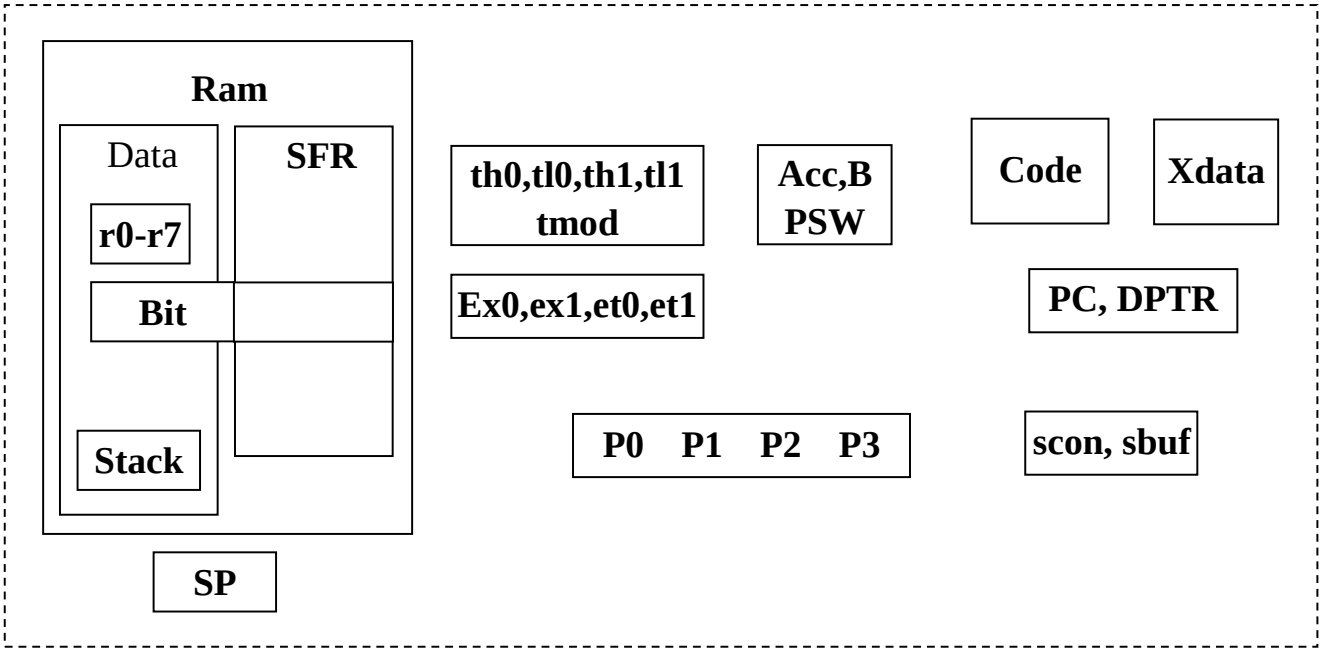


Рис.1.2. Организация памяти mcs51 в A51.

Все типы данных, определяемые в C51, имеют конкретное упорядоченное размещение по адресам в физической памяти mcs51. Доступ (режимы адресации) зависят от типа данных и физической реализации памяти.

Таким образом, доступ к данным в A51 адресный в отличие от прямого в C51, идентификатор переменной определяет адрес размещения в конкретном типе памяти.

LittleEndian (LE)- байты формата (long x=0x87654321) в записи числа в многобайтовом формате хранения в памяти имеют (по умолчанию) такой же порядок размещения

1

(87 65 43 21).
адреса data 8 9 0xa 0xb

Диаграмма доступа к данным в командах **mov** приведена на рис. 3.2. При ограниченных режимах адресации можно выбирать наиболее короткие пути передачи данных при вводе-выводе, при обращении к расширенной памяти.

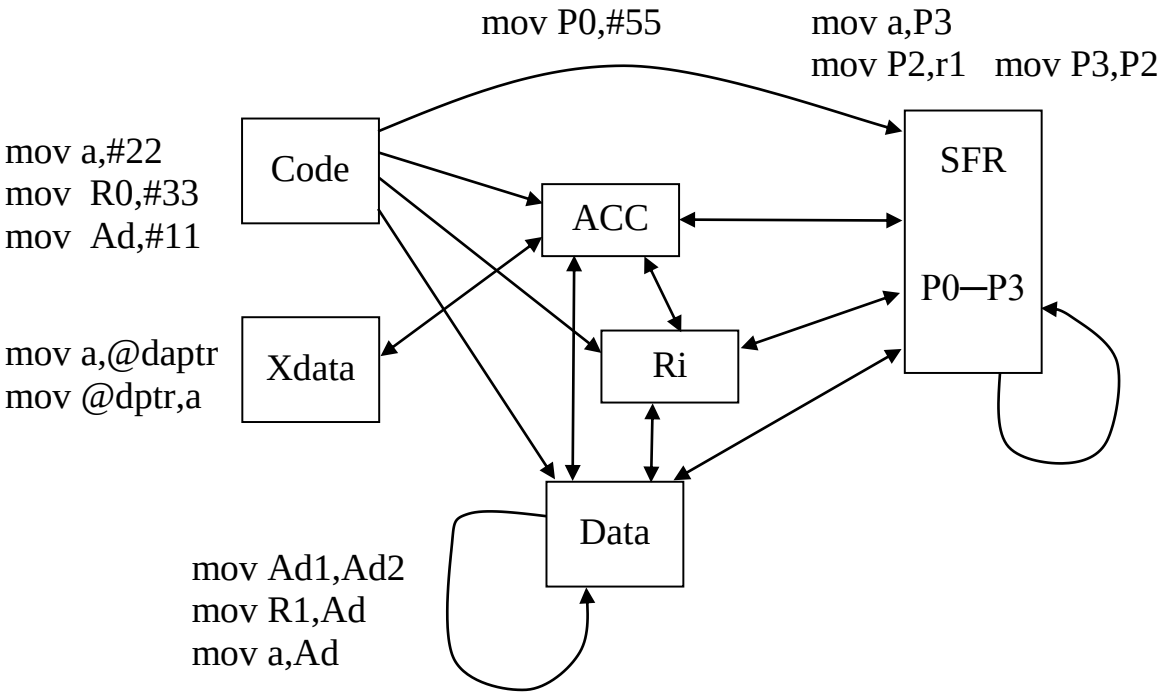


Рис.1.2. Диаграмма доступа к данным в командах mov

Информация с устройств ввода сохраняется обычно в расширенной памяти ЭВМ и загружается (повторно вводится) для обработки в различные виды регистровой и оперативной памяти.

1.Неявно (прямо) доступные регистры

Большинство регистров доступны по именам как в С51, так и в А51,
P1 – это не прямое обращение, как в С51, а **косвенное через адресацию** – к значению.

a(Acc) – основной регистр-аккумулятор, применяемый во всех арифметических и логических операциях с неявным доступом (**a**) – **mov a,r0**.
ACC- обращение по адресу в SFR - **mov Acc,r0**.

B – рабочий регистр, неявно доступен в командах умножения **mul ab** и деления **div ab** или по адресу в SFR – **mov b,r0.**

Регистр состояния **PSW** неявно доступный и модифицируется в арифметических командах, прямо адресуемый в **SFR.**

PC- 16-разрядный программный счетчик или регистр адреса команды неявно доступный рабочий регистр. При включении питания автоматически сбрасывается. PC адресного доступа не имеет, но может контролироваться косвенно и модифицируется неявно.

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

2.О**перативная память данных Ram с прямой адресацией** включает сегменты **Data** и **SFR**.

Сегмент **Data** включает **Stack**, регистровые банки памяти **Ri** , битовую память **Bit[00..7F] (рис**. 1.3.a)

Память **idata** (косвенный доступ через регистры общего назначения r0,r1) имеет с **data** общее адресное пространство 128 байт.

На рис.1.3.b приведена иерархия памяти модели mcs52 (бджайшая модификация mcs51 имеет расширенное адресное пространство Idata 256 байт)

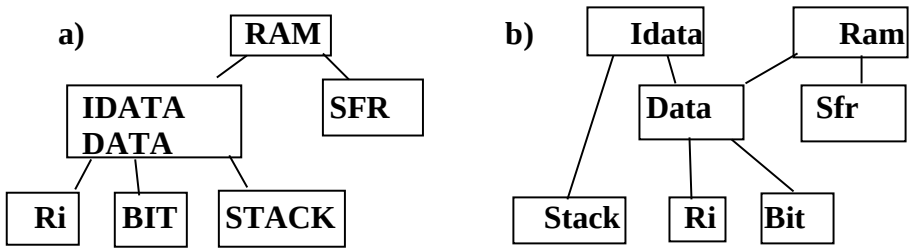


Рис.1.3 **Иерархия внутренней памяти данных a)Mcs51, b)Mcs52**

Все представленные типы памяти различаются способом доступа к данным (прямой или косвенный) и типом данных (бит, байт), но могут относиться к одним и тем же ячейкам Ram.

Сегмент Data.

1)А**бсолютный сегмент (прямая адресация)** для размещения данных,

```
        Dseg at 0x30
  X: ds 2  ;переменная 2 байта
  Y:  ds  1
```


1

Можно также использовать автоматическое размещение данных программой Link по имени сегмента **my segment data**

```
rseg my
x: ds 2
```

Прямая адресация к Data

```
Dseg at 0x40   ;абсолютный сегмент данных
Dm:  ds  4     ;адрес первого (0) байта блока данных из 4-х байт

Mov a, Dm+2        ; Ram[Dm+2]→acc
Mov  Dm, Dm+3   ;  Ram[Dm+3]→ Ram[Dm]
```

Для отдельных байтов можно определять имена, используя псевдокоманды

```
Ss  equ  0x22      (в памяти Data)
Ee   data  0x100   (в памяти Data)
```

2)Регистры общего назначения Ri={ R0,R1,..R7 } (регистровая адресация) – активный регистровый банк.

Доступны 4 банка, совмещенные с начальными ячейками памяти **Data.** Активный банк выбирается в регистре PSW. Адрес соответствующей ячейки Data определяется смещением относительно банка (**RS1.RS2).Ri (**например, в 3-ем банке регистр R2 имеет адрес 0x1A)

Доступ к этим ячейкам регистровый (0-7)

```
mov  a,R0   ; Ram[R0]→ Acc
mov R1,a    ;  Acc → Ram[R1]
mov  00, r1  ;  r1 →r0
mov  r3, 01  ; r1 →r3
```

В С51 при определении функций может быть выбран любой банк рабочих регистров, что позволяет автоматически переключением банков сохранить **контекст** из общезначимых регистров приостанавливаемой или прерываемой программы

```
void func(char x) using 2
```

Здесь назначен банк 2 рабочих регистров при обращении к функции, что позволяет сохранить регистры текущего банка и при выходе восстановить рабочие регистры переключением банка. Состояния других регистров – ACC, SP, B, PSW в ассемблере необходимо сохранять в стеке индивидуально.

В А51 можно задавать размещение операндов из С51 в конкретных регистрах, сохраняя при этом смысл соответствующих идентификаторов в ассемблере (адрес памяти).

```
i  equ  r2
j   equ  r3
```

3)**Косвенная регистровая адресация**

```
Data at 0xA0   ;абсолютный сегмент данных
Dm:  ds  4     ;адрес начала блока данных из 4-х байт
    Mov R0,#Dm  //регистр косвенной адресации
    Mov a,@R0   //косвенная адресация
    Inc R0
    Mov Dm,@R0  ; Ram[R0]→Ram[Dm]
```

Косвенная адресация имеет преимущества – на 1 байт короче команда и возможность переадресации с использованием арифметики

В косвенной адресации модели mcs52 доступен в два раза больший объем памяти.

В С51 можно явно задать регистровую косвенную адресацию **char idata x**;

3) Регистры *SFR* *с прямой адресацией в* ***Ram (80-FFh)***, 128 байт – управляющие и системные регистры. Имена регистров определены неявно в файлах reg51.h, reg515.h или могут быть явно определены псевдокомандой

P5 equ 0x0E0

P5 data 0xF8 (определения в файле reg515.inc в ASM)

К SFR mcs51 относятся указатель стека **SP**, таймеры **TH0,TL0**, **TH1,TL1**, регистры ACC, B, PSW, **DPTR**=DPH.DPL, регистры портов P0,P1,P2,P3.

В ассемблере и С51 для MCS51/52 имена системных и управляющих SFR-регистров зарезервированы по умолчанию. Свободные регистры SFR доступны с прямой адресацией **mov a,0x85**

```
mov  a, b    ; b → A
mov  P1,a    ;  A → P1
mov P1,P2    ; P2→P1
```

Пару смежных свободных регистров SFR можно определить в С51 как одну переменную в формате INT.

```
sfr16  y=0xA1;  //адреса двух свободных смежных регистров в SFR
sfr  xx=0xA0; //поименован свободный байт в SFR для доступа
```

4)**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.**

1

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 и не входят в сегмент битов с последовательной адресацией.

x4 bit ACC.5 ;битовая переменная, соответствующая 5-ому биту ACC
mov c, 0 ; Data(20h.0)→ C , 20h.0 – нулевой бит ячейки Data
mov ACC.7, c ; c → Acc.7
mov c, x0+2 ; x0- адрес первого бита поля бит
mov x4,c

Битовые операции

anl c,{bit, /bit} /bit – инверсия бита ;
Например, **anl c,/ACC.6**
orl c,{bit , /bit} mov c,bit
setb bit, clr bit, cpl C

5) Сегмент Стека

В C51 стек формируется неявно и автоматически доступен при обращении к подпрограммам и в прерываниях. В подпрограммах принято сохранять используемые переменные и регистры основной программы
В модели **Small** стек организуется в Data. В **Compact** - в Xdata со страничным доступом(**pdata**), в **Large** – в xdata с косвенным доступом.
В real-тайм приложениях - основная программа и прерывания обращаются к одной и той же Функции. Компилятор создает отдельный стек для **реентранной** функции – при завершении этой функции возвращается прежнее состояние указателя стека
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

В A51 распределение памяти под **Стек** задано пользователем .
По умолчанию, указатель стека в Data устанавливается аппаратно SP=07
В mcs51 в модели **Small** используются ячейки памяти Data с косвенным автоинкрементным доступом через регистр-указатель вершины SP (пре-автоинкремент **(+SP)** при записи и пост-автодекремент **(SP-)** при чтении)

Определение стека

Dseg at 0x10 ;сегмент стека
Stack: ds 6
Start: ;начало программы
Mov sp,#Stack-1 ; начальное значение стека с учетом пре-инкремента

К стеку возможно как **явное обращение** , так и неявное – в **прерываниях** и переходах к подпрограммам

Явное обращение к стеку:

push ad ; запись байта в стек
Например**, push Acc** обозначает **Ram[Acc]→Data[+SP],**
pop ad ;чтение байта из стека
Например**, pop Acc** обозначает **Data[SP--] → Ram[Acc]**

Типовой порядок размещения различных сегментов данных в иерархической памяти Data

(R0-R7)*4	0-7
глобальные переменные	0x08-0x17
стек	0x18 - 0x1F
бит-адресуемые ячейки	0x20 – 0x2F
локальные переменные	0x30 -

Dseg at 0x30
X: ds 2 ;переменная 2 байта
Stack: ds 4 ; стек поользователя

Для конкретного проекта сегменты могут иметь другие размеры и абсолютные адреса смещаются с сохранением уплотнения

3. **Сегмент Xdata**

xseg at 0x100 ;абсолютный сегмент
mm: ds 50 ;адрес первого байта массива 50 байт

2

Доступ косвенный через dptr

mov dptr,#mm ;адрес
movx a, @dptr ;Xdata(dptr) → A
movx @dptr,a

Доступ страничный (в C51 **char pdata x;**)

mov r0,#xx
movx a, @r0 ; **Xdata(P2.@r0)** →A, в P2 адрес страницы, @r0 –смещение в странице

Определение адреса байта **xx xdata 0x100**

В ассемблере порядок размещения находится под контролем программиста и может быть изменен, кроме порядка, закрепленного аппаратно – адресация битов, адресация регистров в четырех банках, начальное размещение стека.

4. Сегмент программной памяти Code.

Доступ к данным

mov a,#d ; Code[PC+] → a - непосредственная адресация
movc a,@a+pc ; Code[PC + a] → a **;** адресация относительно текущего PC**,** в ACC размещается индекс

cseg at 0 ; старт при сбросе и включении питания в mcs51
jmp start
cseg at 0x40 ; абсолютный сегмент памяти Code с адреса 0x40
start: jmp first
yy: db "abcde" ; адрес первого байта строки в сегменте

first:

mov dptr,#yy ; сохранение адреса
movc a,@a+dptr ; Code(dptr + a) → a, базовая адресация- база в DPTR, в ACC смещение

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

1.3. Ввод-вывод в С51 через порты

В С51 адресный ввод-вывод представлен цифровыми 8-битовыми портами **P0-P3** через адреса SFR и последовательным программируемым интерфейсом **USART**.

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

Порты P1,P2,P3 в реальных схемах включены как **однонаправленные** и настраиваются на соответствующий режим обмена. Детали их использования в совместном режиме конкретизируются на уровне A51.

Вывод **P2=0x55;**
Ввод **char bb=P1**;
bb+=P1;

Порт представляет двоичный 8-битовый код и приобретает смысл типа при вводе с записью в конкретный формат или при выполнении операций с конкретными типами данных.

```
P1=0x80;     //двоичный код
Char   x=P1;        //x= -128  интерпретируется как знаковое число
   z=x;          //z= -128
   y=P1;         //y=128    в формате int интерпретируется как целое беззнаковое
   y=x*P1;       //y <0     P1=0x80 >0 и char x<0
   y=z*P1;       //y >0   P1=0x80 >0 и unsigned char z>0
PSW=0xC0;
   y=PSW;
}
```

Оператор **Чтение-модификация–запись** применяется с двунаправленным портом P0 и **выходными** портами P1,P2,P3.

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

```
anl  P1,#0xAA
Inc P1
```

Порт P1 должен быть выходным.

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.

При вводе (**char x=P1**) данные считываются с контактов порта и сохраняются в памяти, обычно интерпретируются в приложениях в положительном кодировании двоичными кодами (H~1, L~0).

При выводе (**P2=0x55**) данные из памяти записываются в порт и передаются на внешние контакты соответствующими уровнями

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

Разряды портов P1,P2,P3 могут быть использованы для ввода-вывода, если через них не передаются **альтернативные сигналы Altf,** формируемые для управления или контроля внешних схем по умолчанию (сигналы Rd, Wr, int0, int1,…)

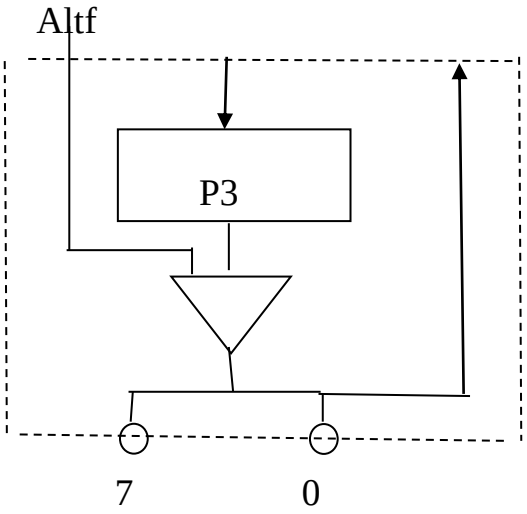


Рис. 1.4.. Структура квази-двунаправленного порта

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

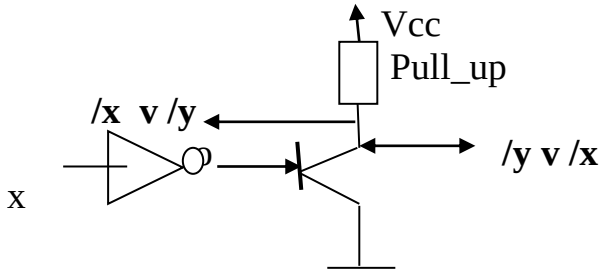


Рис.1.5..Схема драйвера квази-двунаправленного порта

2

Втекающий (**synk**) ток (при низком уровне выходного сигнала) – до 10ма на линию, при максимальном общем потреблении для портов P1-P3 до 100 ма. 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
Altf rd wr t1 t0 int1 int0 txd rxd альтернативные
P2 7.....0
Altf xdata[15..8] альтернативные
P0 7……..0
Altf xdata[7..0]
Data [7..0]

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). Для ввода с контактов необходимо в соответствующих разрядах регистра установить единицы.
В **квази-двунаправленных** портах P1,P2,P3 режим работы порта определяет схема включения порта и тип данных согласуется с типом переменной в Си. :

Системный (внешний) параллельный интерфейс(DA,A,wr,rd.psen) и сетевой последовательный (txd,rxd) в MCS51

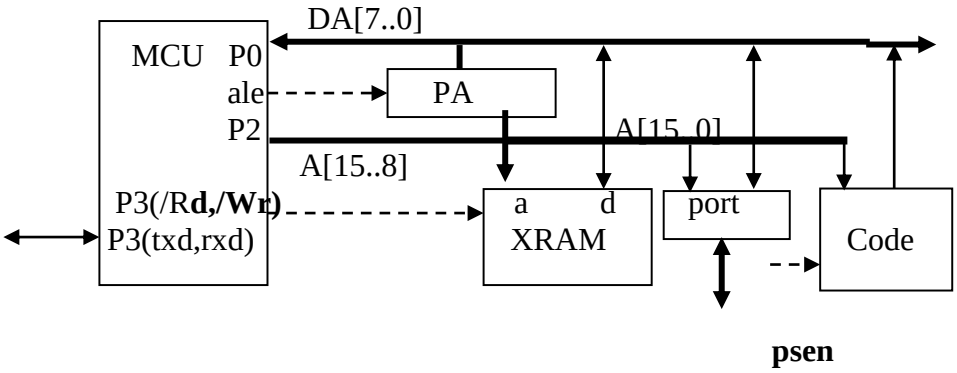


Рис.1.6. Мультиплексированная системная шина

При обращении к внешней памяти данных **Xram и** портам (режим **memory mapped**) 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.
Если используется внешняя программная память Code, то в командах **movc** и при чтении команд формируется сигнал чтения **psen,** при обращении к XRAM –

сигналы чтения /**Rd** и записи /**Wr** формируются как альтернативные через порт P3.

Направление передачи данных через **двунаправленный** порт P0 определяется сигналами записи /**Wr** или чтения /**Rd,** которые являются стробами для внешних устройств и признаками готовности MCU, соответственно, к выводу или вводу через порт. При этом исключается одновременный ввод и вывод, что может привести к конфликту на шине.

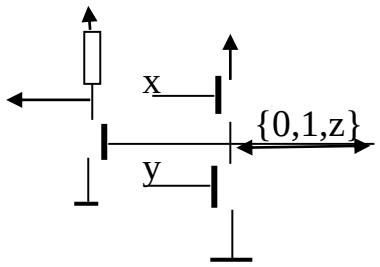


Рис.1.7. Драйвер двунаправленного порта P0

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

1.3. Управление программой.

Операторы управления программой в C51

Goto метка;

```
if (условие)        while(условие)        do                    for (i=0;i<N; i())
     [    ];            {                   {                        {
       else                 [    ];             [   ];                  if(усл) break;
        [   ];           }                    }                        }
                                              while(условие);
```

```
switch (ss) {
                Case  0x55 of :   [      ]; break;
                      ……..
                Case  0x66 of :   [      ]; break;
                   Default: [   ];
                }
```


2

В С51 могут быть определены **рекурсивные (рекуррентные) вычисления,** в которых циклы заменяют рекурсии.
Reentrant (реентранная) функция декларируется как возможная для многократного (повторного) обращения или рекурсивно.
Например, вычисление факториала определяется рекурсивной функцией

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

1) рекурсивная программа

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

Рекурсивное описание задачи – классика в теории алгоритмов, является формальным и в простых случаях вывод формулы является прямым доказательством правильности алгоритма. Однако соответствующая программа сложнее, чем циклическая.

2) Циклическая программа вычислений

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

Команды управления программой

Команды mcs51 формируют состояние программного счетчика PC
Безусловный переход **goto метка** в С51,
в А51 **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 всегда

(+/- 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

1.4. Ассемблирование.

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

Структура ассемблерной программы

1. Выбор программной модели – по умолчанию подразумевается MCS51 если модель изменяется , то

$nomod51 ;отмена стандартных режимов MCS51

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

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

2

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)
```

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
        }
```

II. Курс Лабораторных работ

Ввод-вывод численных данных

Устройства ввода цифровых данных (клавиатуры, цифровые датчики и др.) преобразуют или считывают внешнее естественное представление

численных данных в двоично-десятичной системе кодирования и выполняется преобразование двоично-десятичных кодов в машинные двоичные форматы данных.

Датчики (сенсоры) в управляющих и измерительных вычислительных системах формируют данные в двоичной или двоично-десятичном форматах на входных портах контроллера. Непрерывное информационное поле, таким образом, дискретизируется в численные данные для обработки в ЭВМ .

К вводу также относим загрузку данных в формате с **естественной запятой** **Int x=5678;** с текстами программ на алгоритмических языках

Устройства вывода (например**,** визуализации данных (дисплеи, индикаторы, принтеры и др)) преобразуют промежуточный двоично-десятичный формат чисел в символьное десятичное изображение. При выводе машинные двоичные целые форматы преобразуются программами в двоично-десятичные (**2/10** преобразование) с учетом знака и масштаба.

3

В машинном вводе и выводе преобразования из 2/10 кодирования в двоичное и обратно выполняются с использованием арифметических операций, свойства которых необходимо учитывать в вычислениях.

2.1. Двоичная арифметика

Основные машинные двоичные арифметические операции: сложение, вычитание, умножение и деление (в некоторых ЭВМ, например, ЛИТМО1 использовалась операция извлечения квадратного корня).
Операция **двоичного сложения(вычитания) – элементарная арифметическая операция** выполняется схемой **арифметико-логического устройства (ALU)** ЭВМ.
Операции **умножения и деления** двоичных чисел в MCU могут выполняться программно в соответствии с известными алгоритмами, в которых применяются элементные операции ALU сложения и вычитания.
Предлагается познакомиться с практическим использованием некоторых алгоритмов в программах умножения, деления и квадратного корня.
Реализация в C51 имеет смысл для демонстрации и отладки алгоритма..
Программа в Ассемблере имеет отношение к аппаратной реализации и позволяет реализовать алгоритм схемами ЭВМ
При этом необходимо контролировать некоторую структурную схему выполнения операции, которая определяет форматы используемых регистров и элементарные операции

2.1.1. Сложение и вычитание двоичных чисел

Основные свойства операции беззнакового сложения в фиксированных форматах с фиксированной запятой

unsigned char x,y,z;

z=x+y; ограничение формата вызывает переполнение, если $z>2^8-1$ для целых или $z > 1.0$ для дробных. В языке C51 переполнение автоматически не контролируется и программист должен сам это предусмотреть.

char x,y,z;

отрицательные числа в языке C51 при выполнении операций предполагаются в дополнительных кодах **(2-|x|)** для дробных или **$(2^8-|x|)$** для целых. В 8-разрядных форматах эквивалентно 0-|x| .
Старший бит двоичного кода совпадает со знаком.
Свойства можно наблюдать для дробных чисел, не прибегая к примерам

z=x+y , x,y>=0 возможно положительное переполнение z<0

z=x+y , x>=0,y<0

z=x+y=x –| y|= x -| y| z>0, если x>|y|
z=x+y=x – |y|= 0- (|y|-x) дополнительный код, если x<|y|
z=x+y , x<0,y<0
z=x+y= -|x| –| y|= 0-(| x| +|y|) z>0 и(-)переполнение , если (|x|+|y)|>=1
z=x+y=0-(| x| +|y|) z<0 и результат в доп.коде, если (|x|+|y|)<1

В С51 переполнение не контролируется и реализуется пользователем – формула в предикатах OV=(signx==signy)&&(signx!=signz)

1.1. **Умножение**

1)Умножение положительных двоичных дробных чисел

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 соответствующими целыми произведениями и алгоритмы дробных произведений применимы к целым – возможно при записи целых и дробных чисел в форматах одинаковой размерности различная нумерация разрядов.

Приведение полиномиальной формы представления чисел в позиционной системе счисления к схеме Горнера для вычисления произведения

$$S=A*B=A(B=0.b_1b_2..b_n) = A(b_12^{-1} + b_22^{-2} .. +b_{n-1}2^{-n+1}+b_n2^{-n})=$$
$$= Ab_12^{-1} + Ab_22^{-2}.. +Ab_{n-1}2^{-n+1}+Ab_n2^{-n} =$$
$$=2^{-1} (Ab_1 + 2^{-1}(Ab_2 +\dots +\_2^{-1}(Ab_{n-1} + 2^{-1}(Ab_n +0))..)=>$$
$$S_0=0 => S_1=2^{-1}(S_0+Ab_n) \rightarrow S_2= (S_1+Ab_{n-1})2^{-1}$$

Рекуррентная формула вычисления дробного произведения **со стороны младших разрядов множителя**

$$\textbf{(2.1)}\ S_{i+1}=2^{-1}(S_i+Ab_{n-i}),\ S_0=0,\ i=0,..n-1,\ B=\{b_1,b_2,..,b_{n-i},..,b_n\}$$

Рекурсивная функция **S(i+1)=(S(i) + Ab$_{n-i}$)/2, S(0)=0, b$_n$ -** младшая цифра множителя.

На рис 2.1 приведена схема размещения операндов на регистрах ЭВМ, которая может быть использована для программирования в Ассемблере и в аппаратной реализации умножения

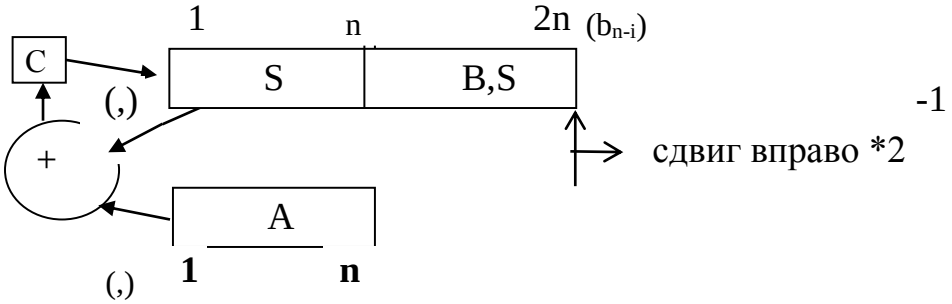


Рис. 2.1. Схема умножения дробных чисел

3

Множитель B размещается в младших разрядах регистра-произведения B.S, младший разряд этого регистра S[2n] при сдвиге вправо сохраняет текущее значение $\mathbf{b_{n-i}}$

Суммирование выполняется в старших (**n+1)**-разрядах формата частичных произведений (Очевидно, в 2.1 Ab_{n-i} =A или 0 для двоичной цифры b_{n-i}={0,1}. Схема может быть использована также и при вычислении 2n-разрядного целого произведения S=AB в форматах с фиксированной точкой после младшего разряда. Двоичный код целого произведения в машинном исполнении на регистрах совпадает с дробным

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

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

Максимальное произведение дробных n-разрядных чисел меньше любого сомножителя и не превышает 2n-разрядный формат. Например, при одном минимальном сомножителе $\mathbf{2^{-n}}$ и другом максимальном $\mathbf{(1-2^{-n})\ 2^{-n}=2^{-n}-\ 2^{-2n} < 2^{-n}}$ При этом для дробных можно ограничиться вычислением и сохранением только старших (**n)** разрядов произведения.(погрешность $\mathbf{\Delta= 2^{(-n)}}$**),** если в дальнейшем не используется деление дробных.

Однако в целом произведении расширение формата произведения до 2n-разрядов может трактоваться как переполнение – в соответствующей машинной команде **mul ab** формируется этот признак OV в **PSW**.

2)умножение двоичных положительных целых чисел

Произведение любых n-разрядных положительных целых чисел больше любого из сомножителей и , в частном случае, равно одному из них

$S=(2^n-1)\ (2^n-1)=2^{2n}\ -2^{n+1}+1$ максимальное целое произведение в 2n-разрядном формате.

А) Умножение со стороны старших разрядов

Приведение полиномиальной формы представления чисел в позиционной системе счисления к схеме Горнера для вычисления произведения

$$\mathbf{S=A*B=A(B=b_{n-1}..b_0) = A(b_{n-1}2^{n-1} + \ldots + b_1 2^1 + b_0 2^0)=}$$
$$\mathbf{=(0+Ab_{n-1})*2+ Ab_{n-2})*2+\ldots + \_Ab_0 =>}$$
$$\mathbf{S_0=0 => S_1=2S_0+Ab_{n-1} \rightarrow S_2= 2S_1+Ab_{n-2} \rightarrow \ldots \rightarrow 2S_{n-1} +Ab_0}$$

Рекуррентная формула вычисления целого произведения со **стороны старших разрядов множителя**

$$\mathbf{(2.2)\ \ S_{i+1}=2S_i+Ab_{n-I}\ ,\ S_0 =0.\ i=n-1,..,0\ \ или\ B =\{\ b_{n-1},b_{n-2},.,\ b_{n-i}\ ,..,b_0\}}$$

Максимальное произведение $\mathbf{(2^n-1)\ (2^n-1)< (2^{2n}-1)}$ размещается в доступном 2n-разрядном формате.

Схема умножения по формуле (2.2) приведена на рис. 2.2.

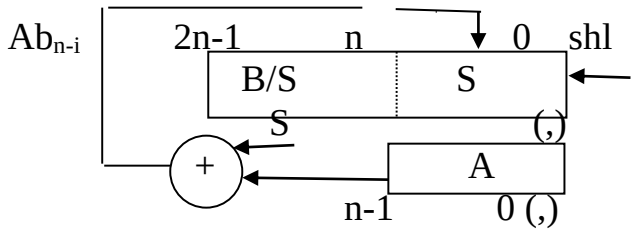


Рис.2.2. Схема умножения целых чисел.

В схеме используются 2n-разрядный регистр хранения произведения, в старших n разрядах размещается множитель B, младшие n разрядов равны 0.
(n)- разрядный регистр множимого A.

B) Способ вычисления произведения целых чисел со стороны младших разрядов

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

$= Ab_{n-1}2^{n-1} + \dots + Ab_1 2^1 + Ab_0 2^0 = b_{n-1}A2^{n-1} + \dots + b_1 A2^1 + b_0 A2^0 =$

$= b_{n-1}(A2^n)\, 2^{-1} + (b_{n-2}(A2^n))\, 2^{-2} \dots + (b_0(A2^n))2^{-n} \rightarrow R=(A2^n) \rightarrow$

$\rightarrow (b_{n-1}R)2^{-1} + (b_{n-2}R)2^{-2} \dots + (b_0R)2^{-n} \rightarrow$ B-дробное после изменения нумерации разрядов $b_{n-1}=b_1$, $b_{n-2}=b_2$, .., $b_0=b_n \rightarrow$

$\rightarrow 2^{-1}(Rb_1 + 2^{-1}(Rb_2 + \dots + 2^{-1}(Rb_{n-1} + 2^{-1}(Rb_n + 0))..) =>$

$S_0=0 => S_1=2^{-1}(S_0+Rb_n) \rightarrow S_2=(S_1+Rb_{n-1})2^{-1} \dots \rightarrow S_n=(S_{n-1} + Rb_1)2^{-1}$

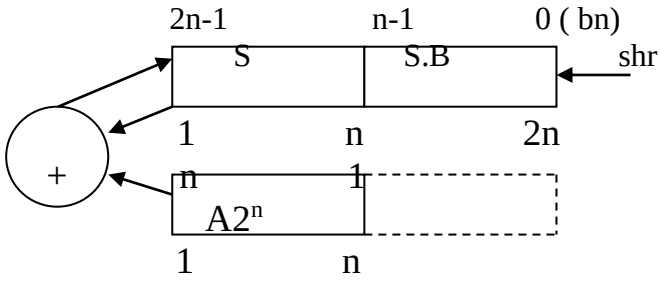
Метод совпадает с методом умножения дробных чисел и является наиболее экономным в схемной реализации.

Рекурсивная функция вычисления целого произведения совпадает с (2.1)

$S(i+1)=(S(i) + Rb_{n-i})/2,$ где R= $(A2^n)$ –целое в масштабе 2^n

и $B = \{ b_1, b_2, .., b_i, .., b_n \}$, $S(0)=0$, b_n - младшая цифра дробного множителя с двоичным кодом $B = \{ b_{n-1}, .., b_{n-i}, .., b_0 \}$, совпадающим с кодом целого числа в масштабе 2^{-n}

Схема умножения



3

Программа умножения n-разрядных двоичных чисел в С51 по формуле 2.1 с учетом схемы размещения операндов Рис.2.1.

```
  Char B,i  //B- множитель, i-счетчик шагов,
  Int  S,Aa; //S-произведение, Aa= A2^n -множимое в согласованном формате

Main(){
  unsigned int Aa,S=0;
  char i;
  Aa=P1<<8,P2=5; //S совмещено с множителем B
  for(i=0;i<8;i++)
     S= (S&1)? (S+Aa)>>1 : S>>1;
          }
```

Объем программы при компиляции в Кейл 127 байт.

Замечания по программе с учетом реальной схемотехники:

1) Алгоритм используется только для положительных чисел, форматы операндов и результаты должны быть **unsigned**
2) При суммировании в схеме возникает неконтролируемое в С51 переполнение – перенос С, теряемый при сдвиге. Ограничиваем при исполнении и вводе диапазон операндов - в целых Aa $<2^7$
3) С учетом этих замечаний выполнить тестирование программы с максимальными возможными в форматах операндами и при A=B=5;
4) Количество шагов умножения зависит от разрядности множителя и от перестановки сомножителей, очевидно, произведение не зависит. По этой причине целесообразно перед началом выполнения операции проверить соотношение между сомножителями и выбрать множителем наименьший из них. При умножении целых (стандартная операция в ЭВМ) можно существенно сократить время вычислений. Операция завершается простыми сдвигами произведения при обращении множителя при сдвигах в нуль.

Программу в А51 выполнить по схеме рис.2.1, используя в комментариях программу в С51, но идентификаторы переменных приходится менять, если при компиляции они окажутся резервированными

```
Dseg at 0    ;абсолютный сегмент данных
 ;/Bb- множитель, i-счетчик шагов, Aa-множимое
; //S-произведение, Aa-множимое
  S: ds 2
 Aa: ds 1
Bb: ds 1
```

```
Ii equ r1
Cseg at 0  ;абсолютный сегмент Code
//S=0; Aa=5,S=5, ii=0;
 Clr a
Mov S,a
Mov ii,a  ;r0
Mov S+1,Bb
;S= (S&1)? (S+Aa)>>1 : S>>1;
cikl:
     ; (S&1)?
   clr c
   mov a,S+1
   jb  ACC.0,  nula
   ; (S+Aa)
    mov a,S
    add a,Aa
    mov S,a
 nula:    ;S>>1
    mov a,S
    rrc a
    mov S,a
    mov a,S+1
    rrc a
    mov  S+1,a
; for(i=0;i<8;i++)
   Inc ii
   Mov a,ii
   cjne ii,#8,cikl
   Nop
 End
```

Объем программы при компиляции в Кейл 35 байт

Упрощение программы с использованием регистровой памяти, в некоторых случаях регистры в мнемонике обозначаются явно (например, r2 – регистр при размещении (S+1) адресе ячейки Data

```
;Bb- множитель, i-счетчик шагов, A-множимое
 ii equ r0      ;  ds 1
; S-произведение, Aa-множимое
  S equ r1       ;S+1 =r2
 Aa equ r3

Cseg at 0  ;абсолютный сегмент Code
//S=0; Aa=5,S=5, ii=0;
 Clr a
Mov r1,a        ;r1=S
Mov ii,a
Mov r2,04     ;r2=S+1
;S= (S&1)? (S+Aa)>>1 : S>>1;
cikl:
    clr c
    mov a,r2   ;S+1
    jnb ACC.0,  nula
    mov a,S
    add a,Aa
    mov S,a
  nula:    ;S>>1
    mov a,S
    rrc a
    mov S,a
    mov a,r2
    rrc a
    mov r2,a
 ; for(i=0;i<8;i++)
   Inc ii
   Mov a,ii
  cjne ii,#8,cikl
   Nop
  End
```

Объем программы при компиляции в Кейл 25 байт.

При тестировании программы привести для сравнения объемы программ , измерить время выполнения цикла с использованием Wie -->Performens Anilaiser

3)Ускоренное умножение с основанием 2^4

При умножении байтов($A=a_1a_2$)*($B=b_1b_2$) с основанием d=$\mathbf{2^4}$

где a1,a2,b1,b2 – HEX-цифры сомножителей в формуле (2.1), учитывается основание d

(2.3) $\mathbf{S(i+1)=(S(i) + Rb_{n-i})/d,}$ где R=$\mathbf{(A2^n)}$ **и** d=$\mathbf{2^4}$

Для вычисления $\mathbf{Rb_{n-i}}$ можно использовать таблицу умножения $S(a_ib_j)$, тогда потребуется 4 шага или такта в следующей схеме

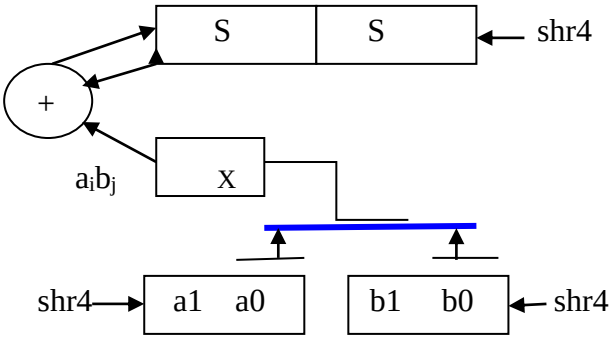


Рис. Схема умножения с основанием d=$\mathbf{2^4}$

Программа функционального моделирования алгоритма в С51

```
#include <reg51.h>
  int  S; //S-произведение,
main(){
  unsigned int S=0;
 char a1,a0,b1,b0;
 a1=P1>>4; a0=P1&0x0f; b1=P2>>4; b0=P2&0x0f;
        S=((a0*b0)+((a1*b0)<<4))<<4; //встроенное умножение
       S=((S+((a0*b1)<<4))>>4)+(a1*b1);
         P0=S; P3=S>>8;
           }
```

Моделирование схемы умножения

```
#include <reg51.h>
  int  S; //S-произведение,
```

```
main(){
  unsigned int S=0;
  char i,j;
  int xdata X[256];
 char a1,a0,b1,b0;
   char s0,s1,s2.s3;
  for(i=0;i<16;i++)  //таблица умножения Нех-цифр
    for(j=0;j<16;j++)
       X[(i<<4)+j]=i*j;
    unsigned char A=P1;
     unsigned char B=P2;

   //схема----------------------------------
        a0=A&0x0f;  b0=B&0x0f;   //0-такт
         S= X[(a0<<4)+b0]<<8;

         a1=A>>4;                        //1-такт
         s1= X[(a1<<4)+b0]<<8;
         S=((S+((s1<<4))<<4;

         b1=B>>4;    a0=A&0x0f;     //2-такт
         s2= X[(a0<<4)+b1]<<8;



        a1=A>>4;                          //3-такт
         s3= X[(a1<<4)+b1]<<8;
        S=((S+(s2<<4))>>4)+s3;

          P0=S; P3=S>>8;
            }
```

4)Знаковое умножение 8-разрядных чисел в С51

Простой способ учета знака - с преобразованием отрицательных сомножителей в прямой код и определением знака произведения суммированием знаковых битов.
Известны методы автоматического учета знаков (например, аппаратный метод Бута)

В mcs51 аппаратно реализовано умножение по общему алгоритму для положительных двоичных чисел, для умножения отрицательных чисел необходимо создать программу на ассемблере, где могут быть учтены форматы кодирования со знаком.
Если сомножители рассматривать как дробные и A<1 и B<1 – модули чисел, то:

1) для положительных сомножителей S= A*B=AB , для максимальных $A=B=1-2^{-n}$ произведение AB < A

2) при разных знаках сомножителей S= A*(2-B)= 2 A – AB. Требуется коррекция результата, может оказаться S>0

3) для отрицательных сомножителей

S=(2-A)*(2-B)=4 -2A -2B +AB= -2(A+B)+AB и S может быть отрицательной. Требуется коррекция результата.

Существуют методы вычисления без использования коррекции. Например, реальная программа из библиотеки C51 для 8-разрядных целых чисел

Программа преобразования числа **char** в формат **int**

```
Mov  r7,x        ;x<0= (2^n - x)=0x9C, x=-63
MOV     A,R7
RLC     A
SUBB    A,ACC
MOV     R6,A     ; 0-C     r6.r7=(-1).x=sx.x =0xff9C  -63 в формате int
```

Вычисления в масштабах с целыми числами

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

$$x,y\text{-модули} = (-2^n)*(2^n-y)*2^n+(2^n-x)*(2^n-y)+(-2^n)*(2^n-x)*2^n$$

$$\text{В формате 2n} = y*2^n + (-y*2^n - y*2^n +xy)+ x*2^n = xy$$

При этом сумма всех единиц в целой части компенсируется и результат прямого умножения положительный **xy.**
Во всех операциях умножения используется команда **mul ab** -16-разрядное произведение 8-разрядных положительных двоичных чисел

1.2.Деление в mcs51

Делимое – произведение дробных чисел и делитель (один из сомножителей) дробные. Следовательно, в корректном делении дробных чисел делимое S<A.

Делимое предполагается в удвоенном формате – запятая фиксирована перед старшим(левым) байтом. Младший байт делимого в команде **div ab** равен нулю, что эквивалентно целому делению n-разрядного делимого в

4

регистре A на n-разрядный делитель в регистре B. Частное формируется в A, в B формируется положительный остаток.
На последнем шаге вычисления (рекуррентная формула 2.1) дробного произведения

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

Тогда 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

(2.4) $\mathbf{S_{i+1}= 2S_i-A}$ **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** $\mathbf{2S_i-A <0}$**,**

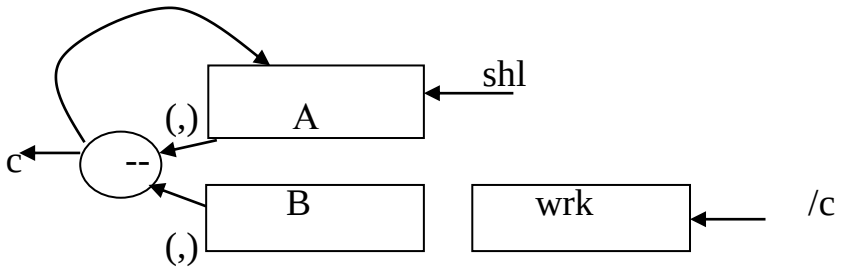


Рис.2.3. Схема деления дробных чисел.

Регистр **B=** совмещает младшие разряды делимого при сдвиге и разряды частного.
Программа деления в C51.

```
#include  <reg51.h>
main()
{ char A=P1 ; //требуется расширенный формат для контроля знака разности
   char B=P2;
  char i=0;
for ( ; i<8; i++ )
{
 P3=A= ((A<<1)-B)>=0) ? (A<<1)-B +1 : A<<1 ;
}
```

В дробных вычислениях требуется пробное вычитание (контроль A<B) и контроль (B!=0).

```
main()
{   char A=P1 ; //делимое <0x7f   положительное
    char B=P2 ;  //делитель <0x7f   положительное
for (char i=0 ; i<9; i++ )
 P3=A= (A <B) ?  A<<1:  ((A-B)<<1) +1 :
   P3>>=1; //остаток
   P0=A; //частное
```

```
while(1);
}
```

Так же как в разделе умножения выполннть программы в С51 и в А51 с размещением данных в Data и регистрах. Выполнить измерения объема памяти данных и программ, измерение времени исполнения.

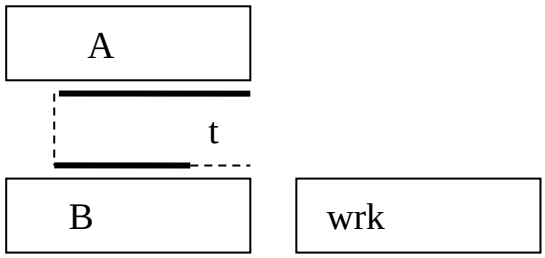
2) Целое деление

В современных компьютерах арифметика с фиксированной точкой целочисленная и необходимо поддерживать целое деление с остатками. Так в С51 есть операция целого деления (/) и отдельно операция определения остатка (%).

В А51 предлагается машинная операция целого деления **div ab=A/B**, которая реализует только целое деление 8/8 →8.8 с остатком.

Можно предположить следующий метод выполнения операции.

Для выполнения прямого деления целых чисел с остатком требуется **нормализация** – приведение делителя к делимому, как в известных ручных операциях с выбором цифры частного со стороны старших разрядов, выполняя сдвиг делителя В влево пока A>B



(8-t) – количество шагов до конца операции.

Положительный остаток формируется в t младших разрядах регистра А,

в регистре wrk поразрядно сдвигом сохраняются биты частного

Программа целого деления с остатком S/A в С51

```
#include <reg51.h>
char A,B,wrk,i;
main()
{
  A=P1;
  B= P2;
          for(i=0;i<8;i++)
            if(A>B)  B<<=1;
                        for ( i=0 ; i<9; i++ )
```


4

```
        P3=A= (A <B) ?  A<<1:  ((A-B)<<1) +1 :
        P3>>=1; //остаток
  while(1);
}
```

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

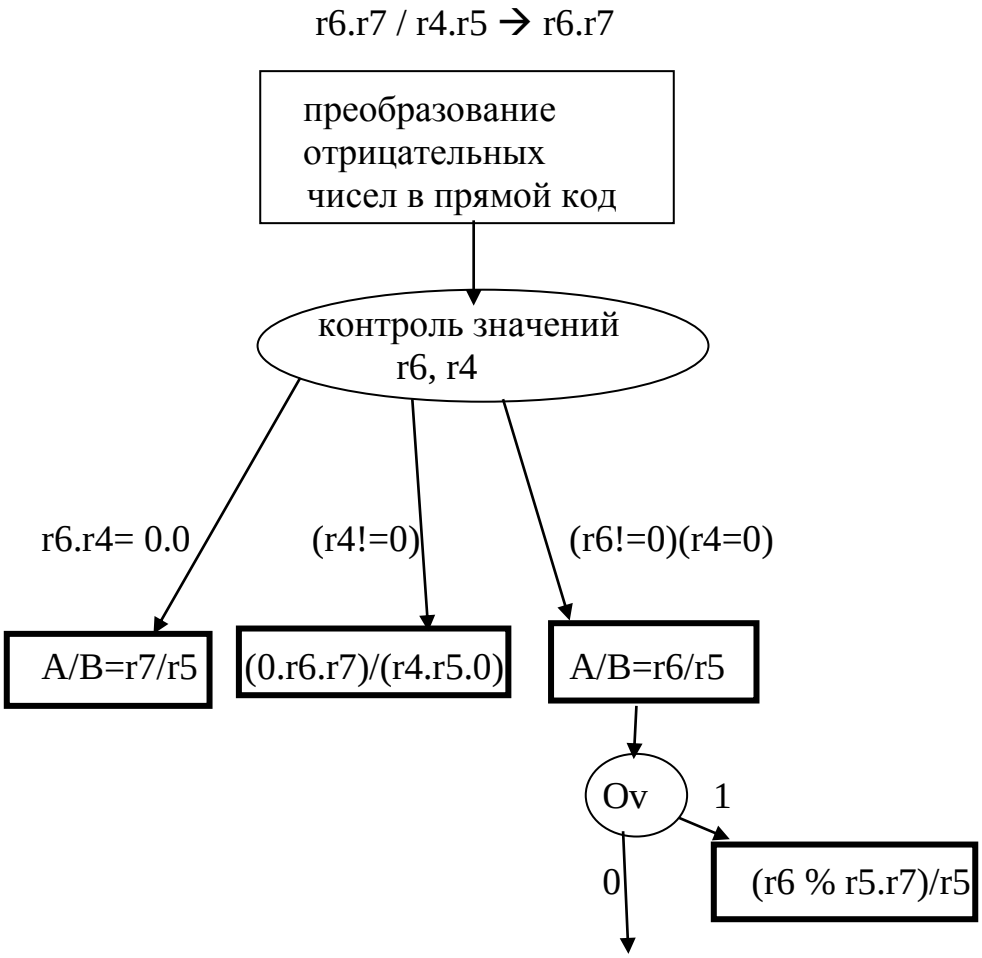


Рис 2.5. Знаковое целое деление в С51

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.

1.3. Извлечение квадратного корня

Обычно как функция включается в стандартную библиотеку с плавающей точкой. Однако известны случаи, где в систему команд ЭВМ включается специальная команда быстрого вычисления квадратного корня
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** $\mathbf{4Q_i.S_i-x_i.01 <0}$

4

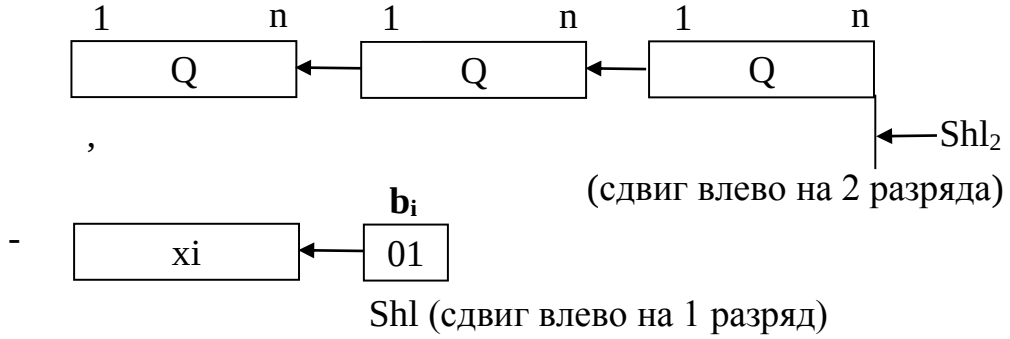


Рис. 2.7. Схема извлечения корня квадратного без восстановления остатка.

2. Ввод беззнаковых целых чисел с цифровых портов

2.1. Преобразования при вводе и выводе целых чисел

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

Информация (количество) в записи (**m)**-разрядного целого десятичного числа

$$\textbf{(2.5)} \qquad N=\sum_{i=0}^{m-1} a_i d^i = a_{m-1}10^{m-1} + a_{m-2}10^{m-2} + \ldots + a_1 10 + a0 =$$

$$= (\ldots((0 + a_{m-1})10 + a_{m-2})10 + \ldots + a_1)10 + a0$$

Рекуррентная формула пересчета десятичногочисла в двоичную систему

$$S_{i+1} = \sum (S_i * 10 + a_{m-i}),\ i=1,2,..,\ S_0=0;$$

где a_i -двоично-десятичная цифра, d=10, m-количество разрядов в записи десятичного числа. Запятая фиксирована в машинном формате после младшего разряда.

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

Рис. 2.6. Формат двоично-десятичного целого числа -входной порт.

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

Десятичные числа с естественной запятой преобладают в машинных вычислениях в экономике, где используется двоично-десятичный формат переменной длины (System Z фирмы IBM), однако в большинстве компьютеров используются машинные форматы с фиксированной точкой (точность – одна **копейка**).

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).

В ЭВМ для вычислений с фиксированной точкой используются форматы **целых двоичных чисел.**

Машинные преобразования целых 10/2 могут быть выполнены пересчетом **количества N** в двоичной системе по обобщенной рекуррентной формуле схемы Горнера [1], где **i=(m-1)…0**

(2.7.) $S_{i+1} = S_i 10 + a_{m-i}$, $\{S_0=0,.., S_m=N\}$ (следует из **2.1.) или**

по формуле с общим членом ряда

$$S_{i+1} = S_i + B_i a_i , B_{i+1}=10B_i , B_0=1, S_0=0, i=0,..m-1$$

Рекуррентная формула (2.2.) может быть переписана в виде **рекурсивной функции [ ]** и выполнена **рекурсивной программой**

$$S(i+1)=S(i)*10 + a_{m-i-1} \quad i=0 - m, S(0)=0;$$

Обратное машинное преобразование 2/10 в двоичной системе может быть выполнено методом “цифра за цифрой”

Неизвестные цифры десятичного числа Ai могут быть найдены делением двоичного числа на основание 10.

Если результат преобразования по формуле (2.2) Sn=Sn-1 + A0, то в остатке A0=Sn%10 и целая часть Sn-1=Sn/10 , последовательно можно определить все цифры со стороны младших разрядов A0A1…..An.

Программа в С51.

При вводе 2-х разрядного десятичного числа $N=a_1a_0$ формула 2.2. имеет вид $N= a_1 *10 + a_0$ При выводе $a_1=N/10$, $a_0= N\%10$

4

Для 8-разрядной ЭВМ mcs51 можно осуществлять ввод и вывод по две десятичные цифры, используя следующие функции Fd2() и F2d().

```
#include <reg51.h>  //файл адресации регистров MCS51
Fd2(char x){
   return (x>>4)*10 + (x&0x0f);
        }
F2d(char x){
   return  ((x/10)<<4) | (x%10); }
main()
{      y=F2d (P1);   //ввод 2/10 числа с порта P1
       P2= F2d (y); //вывод 2/10 числа в порт P2
       while(1);
}
```

2.2. Ввод и вывод дробных численных данных.

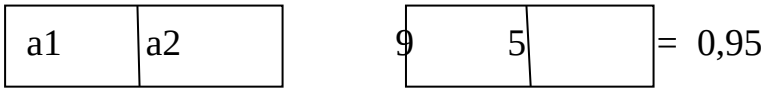
Схемотехника и алгоритмы выполнения арифметических операций ЭВМ с дробными (**n**)-разрядными форматами проще, чем с целыми. Для ЭВМ первого поколения упрощение оборудования было актуально (1955-1964 гг ЭВМ Урал-1 – два формата 15+1, 35+1 бит, 1959-1962 на кафедре ВТ разрабатана ЭВМ ЛИТМО-1 для оптических расчетов со смешанным форматом – целая часть 2 бита и дробная часть 24 бита),

1) **Преобразования 10/2 дробных в целой арифметике при вводе**.
   Информация (**количество N**) в **записи** (**m**)-разрядного дробного десятичного числа **A=0,a1a2a3…am =0,1234….**

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

$$= (\ldots((0+ a_{-1})10^{-1} + a_{-2})10^{-1} + \ldots + a_m)10^{-1}$$

где a_i -двоично-десятичная цифра, d=10, m-количество разрядов в записи десятичного числа. Запятая фиксирована в машинном формате перед старшим разрядом.



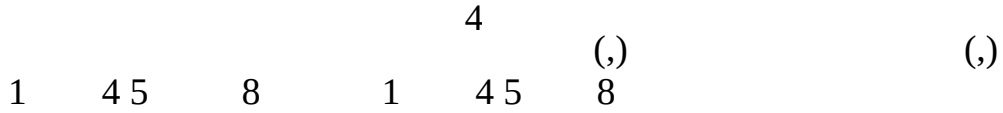
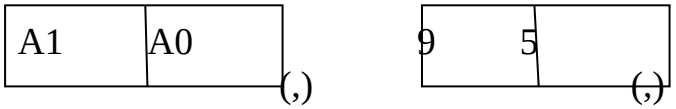


Рис. 2.8. Формат двоично-десятичного дробного числа .

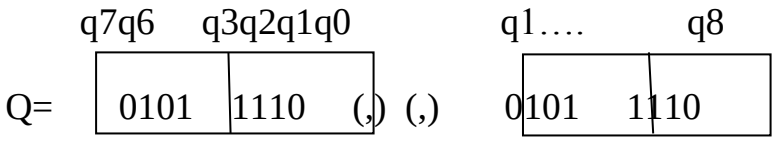
Десятичное дробное число в m-разрядном формате можно рассматривать как целое **$A_{2/10}$ = a1a2..am,0 или AmAm-1…A0 с масштабом 10^m** .



Тогда, применяя рассмотренный выше метод преобразования целых (2.3), получим (**n)**-разрядное двоичное целое число B_2

Для получения дробного двоичного n-разрядного необходимо разделить целое на масштаб Q=**B_2 /10^m в** двоичной системе. Однако в целочисленной арифметике это не возможно и требуется сначала выполнить масштабирование с двоичным масштабом Q= (**B_2*2^n)/10^m = Q_2*2^n** , чтобы результат деления оказался дробным двоичным в масштабе **2^n**

Двоичный код целого числа **Q** в n-разрядном формате совпадает с дробным двоичным **в том же формате**.



Здесь изменяется только порядок отсчета разрядов двоичного кода и q7 в целом формате равен q1 в дробном.

Программа ввода дробного числа

```
#include <reg51.h>
unsigned int B2,y;
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
  while(1);
  }
```

Преобразование десятичной дроби в двоичную в общем случае при округлении усечением имеет погрешность 2^{-8}

4

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[.**

Вывод дробных чисел

Если известно двоичное **$Q=N=S_n=0,a1a2…am$**, то неизвестная двоично-десятичная цифра a_1 = N*10 - целая часть двоичного произведения

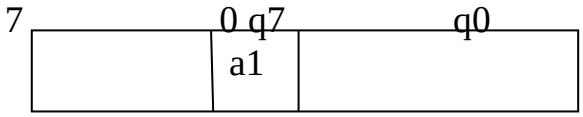
Q_{n-1}=a1, a2a3..am

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

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

Умножая двоичное целое в масштабе 2^n на основание 10, получим **a_1** в целой части (в старших (**n)** разрядах целого произведения. (Умножение 10* (**B_2*2^n**) сохраняет масштаб **2^n** и положение запятой, если сохраняем размещение результата и его младшего разряда - в данном случае перед **n-**ым разрядом).

```
y=B2 *10;
 P2=((y&0xf00)>>4);  //A1- двоично-десятичная цифра
 P2 | =(((y&0xff)*10)&0xf00)>>8;  //A2-вторая  двоично-десятичная цифра
```



(,)

Последовательно можно определить все цифры со стороны старших разрядов .

3. Вычисления функций

Распространенные функции вычисляются по формулам разложения в ряд Тейлора, который в большинстве случаев сходится в диапазоне дробного аргумента **0-1.0**.

3.1. Вычисление функции с плавающей точкой.

FP- машинный формат позволяет использовать полулогарифмическую запись числа при вычислениях и , следовательно, возможность обработки чисел на ЭВМ в широком диапазоне с автоматическим изменением масштаба, с постояннной отнсительной погрещностью и переменной абсолютной .

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

1) сложные алгоритмы преобразования – высокая сложность аппаратуры и большие затраты времени при вычислениях

постоянная относительная погрешность, зависящая только от разрядности мантиссы $\partial = \mathbf{2^{(-n)}}$ **(**двоичная (n)-разрядная мантисса, округление усечением).

С фиксированной точкой программа проще и время вычислений сокращается, что и предполагается показать в лабораторных работах при вычислении функций .

Пример.

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

С использованием Логического Анализатора получить временные диаграммы и 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

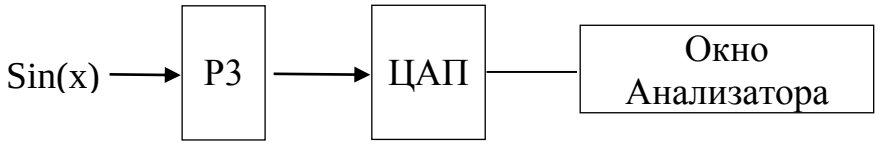


Рис.2.9. Схема работы Анализатора

При выводе через порт P3 учитывается, что значение должно быть целым, положительным и не более $\mathbf{2^8}$.

5

```
#include <reg51.h>
#include <math.h>
float x,y;
 char i;
```

```
main()
{
```

/* цикл формирования значений в массиве
вычисления с плавающей точкой – в том числе и масштабные преобразование и перевод в целые
*/

```
i=0;
for(x=0; x<6.28 ;x+=0.0628){
        P2=0;
        y= sin(x); P2=1;  //метка для измерения времени
        P3=y*100+100;     //масштабное преобразование переводит
```

дробное в целое с точностью 2 знака после запятой и смещает в положительную область значений

```
  }
```

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

Реальное время вычислений контролируется Симулятором и синхронизировано частотой работы компьютера.

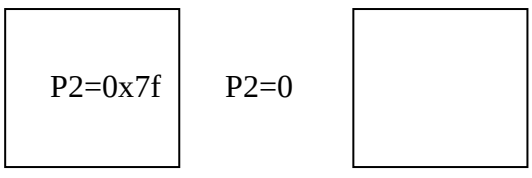
В опциях Project.options.Target частоту MCU выбрать 12.0 Мгц

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

В начале программы формируется метка времени P2^=0x7f.

Командой **LA P2** в командном режиме значение P2 передается при выполнении программы в окно Анализатора. **Время** вычислений контролируется Симулятором и синхронизировано частотой работы компьютера. В опциях Project.options.Target выбираем частоту синхронизации **12.0 Мгц**

Временная диаграмма в окне Анализатора.



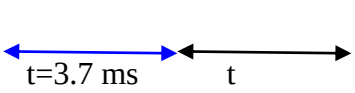
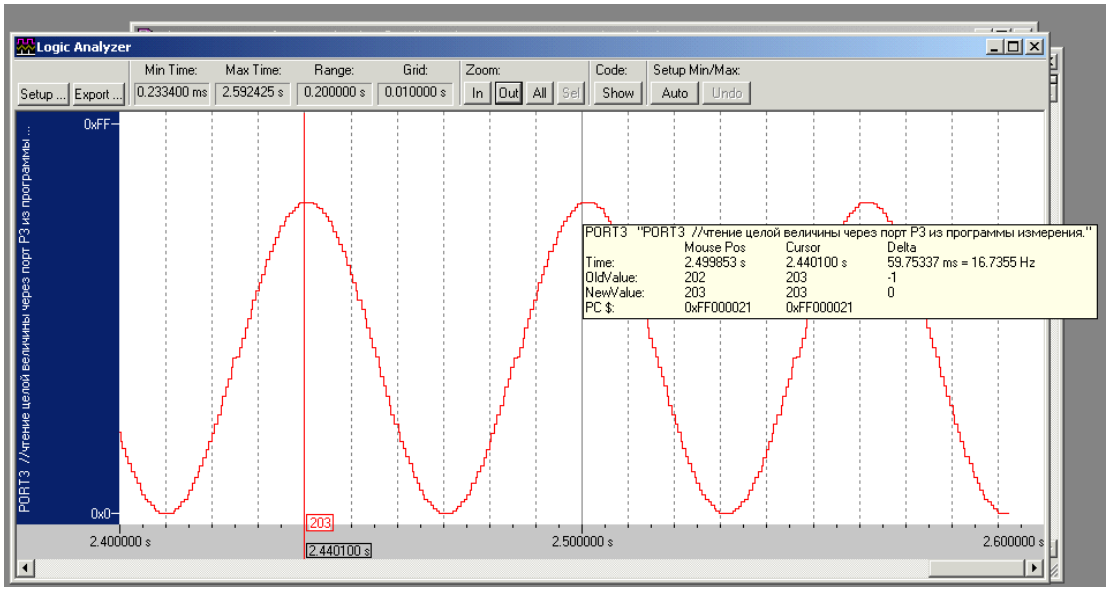


График функции в окне Анализатора.

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



2.8. График функции в окне Анализатора.

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

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

Рассматриваем применение вычислений функций с фиксированной точкой как необходимые для ЭВМ, в которых 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.

5

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

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

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

Вычисления выполняются **по схеме Горнера[1]** с числом членов ряда 3-4 или по формуле с общим членом ряда.

Преобразование по схеме Горнера

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

рекуррентная формула

$S_{i+1} = 1-x^2/a_i * S_i$

$S_0=1$; i=0,1,2

$S3=xS_2$

Выберем аргумент в диапазоне дробных чисел **0- 0.99 радиан и преобразуем в целые в масштабе 100**.

Для сохранения масштаба результата используем следующие преобразования

$\sin x \sim$ **$x - x^3/m^2/3! + x^5/m^4/5! - x^7/m^6/7!$**

x – значение аргумента в диапазоне [0 – m], sinx – дробное в масштабе **m**

Величину масштаба ограничивает заданная точность вычислений и размер формата данных. Например, для 8-разрядных вычислений и точности 2 знака после запятой

$$10^2 \le m < 2^8$$

1) Вычисления с десятичным масштабом $m=10^2$

Вычисления с целыми по схеме Горнера

$\sin x \sim$ **$x(m - x^2/m/6(m - x^2/m/20(m - x^2/m/42)/m))/m)/m$**

x – значение аргумента в диапазоне [0 – 100]

Си занимает 1.2 мс и объем программы 219 байт

A=(x*x)/m; A1=(A*A)/m

Sin~ x- A/6 +A1/20 - ((A1*A)>>8)/20

x – значение аргумента в диапазоне [0 – 255]

```
#include <reg51.h>
unsigned int x,y, sin;
main()
```

```
                                        {
 while(1)
  for(x=0; x<100; x++)
    { A=(x*x)/m; A1=(A*x)/m;
        sin= x – A/6 ;
        sin + = A1/20 ;
         P3=sin -= ((A1*A/m)/42;
     }
 }
```

2) Вычисления с двоичным масштабом

Для упрощения вычислений можно выбрать **двоичный масштаб m=2^8** , что позволяет исключить деление и заменить его сдвигами

A=(x*x)>>8; A1=(A*A)>>8

Sin~ x- A/6 +A1/20 - ((A1*A)>>8)/20

x – значение аргумента в диапазоне [0 – 255]

```
#include <reg51.h>
unsigned int x,y, sin;
main()
{
 while(1)
  for(x=0; x<=0x255; x++)
    { A=(x*x)>>8; A1=(A*x)>>8;
        sin= x – A/6 ;
        sin + = A1/20 ;
         P3=sin -= ((A1*A)>>8)/42;
     }
 }
```

При вычитании в формате байта используем значение единицы 0xff. Если вычитаемое не равно нулю, можно добавить единицу для повышения точности. Объем программы 176 байт, среднее время вычисления 0.15мс

В Ассемблере все деления простые целые выполняются командой **div ab**

3) Исключение деления

С двоичным масштабом можно использовать упрощение вычисления в ассемблере заменой деления умножением

5

```
unsigned char x,y,si;

 unsigned char sin(unsigned char y)
 { si=(((y*13)>>8)*(0x0ff-((y*6)>>8)))>>8;   //256/43=6, 256/20=13
    si=(((y*43)>>8)*(0x0ff-si))>>8;              //256/43
      si=(x*(0x0ff-si))>>8;
         return si;}




main()
{
 while(1)
  for(x=0; x<=0x0ff; x++)
   { y=(x*x)>>8;
           sin(y);
         }
 }
```

Объем программы 132 байта, среднее время вычисления 0.1 мс

4) Преобразование предыдущей программы в A51

Для отладки программы используем размещение переменных в памяти Data. Затем для сокращения объема программы можно заменить переменные в Data регистрами.

```
Dseg at 0x10       ; регистровая память
 x: ds 1              ;x equ r0
 y: ds 1              ;y equ r1
 si: ds1               ;si equ r2
 xx: ds 1              ;xx equ r
stack: ds  2

cseg at 0
 jmp main
cseg at 5
 sin:
    ; si=(((y*13)>>8)*(0xff-((y*6)>>8)))>>8;
                mov a,y
                mov b,#13
                mul ab    ;((y*13)>>8)
                mov xx,b
```

```
                                                    mov a,y
            mov b,#6    ;((y*6)>>8)
            mul ab
            mov a,#0x7f
            subb a,b
            mov b,xx
            mul ab      ;si
            mov si,b
```

```
    ;si=(((y*43)>>8)*(0x7f-si))>>8;
            mov a,y
            mov b,#43
            mul ab    ;((y*43)>>8)
            mov xx,b
            mov a,#0x7f
            subb a,si
            mul ab      ;si
            mov si,b

  ;si=(x*(0xff-si))>>8;
            mov a,#0xff
            subb a,si
            mov b,x
            mul ab      ;si=b
            mov si,b
            ret
main:
           mov sp,#stack-1
 ;for(x=0; x<=0x07f; x++)
            mov x,#0
            ;y=(x*x)>>8;
            ;  sin(y);
     cikl:
            mov a,x
            mov b,x
            mul ab
            mov y,b
            call sin
            mov P3,si
```

```
inc x
cjne x,#0x7f,cikl
jmp main
nop
end
```

Объем программы **64 байта**, среднее время вычисления **0.05 мс**

Варианты Задания по разделу 2.2. Ограничимся для всех функций диапазоном дробных аргументов [0<=x<1] и масштабами m=100 и **$m=2^8$**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4. Иерархия памяти ЭВМ.

На языке С51 используется упрощенная модель памяти, не доступна регистровая память РОН, используемая компилятором по умолчанию, и регистры с неявным доступом – ACC,B,PSW и др.

В Ассемблере нет ограничений на доступ к различным уровням иерархии памяти, что во многом определяет более высокое качество программ. Ассемблер позволяет учесть эффективность кодирования режимов адресации в командах с учетом неявной информации о структуре и диапазонах данных.

Данные представлены и обрабатываются в виде текстовых строк. ASCII-константы хранятся в памяти Code, предлагается преобразование этой строки с сохранением в ASCII в расширенной памяти данных Xdata. В

преобразованиях также используется память Data, регистры с адресным и неявным доступом.

Пример.

1) Прямой доступ к данным

```
#include  <reg51.h>
char x;   //переменная в регистровой памяти данных, имя переменной подразумевает значения
char  code y[ ]= “123”; //символьная константа в программной памяти
char  xdata yy[8]; //результат преобразования в расширенной памяти
main()
```


5

```
{     char i;  //переменная в регистровой памяти
                x=0;
      for (i=0; i<3; i++)      ;перевод в двоичную
                x=x*10+(y[i]&0x0f);
      for(i=7;i>=0; i--)     ; преобразование в символы
                { yy[i]= (x&0x01) ? '1' : '0';
                  x=x>>1;
                }
      while(1); //динамический останов
}
```

Объем программы …..

2) Косвенный доступ к данным по адресу через адрес-указатель

```
#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 на текстовую переменную
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 …..

Задания

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

6

ключ {3,1,2,0,6, …}

Пример

“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

5.Булевские и битовые данные.

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

1) Доступ к битам в C51

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

sbit y1=P1^2;   //биты порта P1 (не смешивать с операцией ^ -
//исключающее ИЛИ
sbit y2=P1^3;
char bdata mem  //ячейки  сегмента DATA с битовой адресацией
int bdata mem1
sbit  y1=mem^0  ;0 бит ячейки mem
```

2) Адресация к битам в Ассемблере

```
y3  bit 11h       ;прямой адрес в поле 0-7f  бит
x1  bit P1.0      ;бит порта
z2  bit acc.1     ; бит аккумулятора
   bseg at  10 ;абсолютный сегмент битов с10-го адреса
   x3:  dbit  2   ; поле из двух бит

    mem equ 21h ;бит-адресуемая ячейка Data
   y1:  bit   mem.0  ;0 бит ячейки mem
   y2:  bit    mem.1
   mov c,x2+3     ; обращение к битам
   anl c,y1
   mov z1,c
```

Пример.

z1=y1&!y2|x1 // вывести вектор значений функции в порт P1

Программа в с51

```
#include <reg51.h>
char bdata mem  //бит-адресуемая переменная
sbit x1=mem^0;  //биты двоичного набора
sbit y1=mem^1;
sbit y2=mem^2;
sbit z=P1^0;
main()
{
      for(mem=0;mem<8;mem++)
            {P1<<=1 ; z= y1&!y2|x1;}
 }
```

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. $z=(/x_1 \vee y_1)\ (x_2y_2 \vee /y_1)$

6

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. $z=(x_1 \vee /x_2 \vee /x_3x_4)(/x_1 \vee /x_4)$

Литература.

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

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

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

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)
movx a,{@rj,@dptr} a←xram{..}
movx {@rj,@dptr},a xram{..}←a
push ad Data(+sp)←Data(ad)

	orl a,{ri,@rj,#d,ad}	a v {…}→ a pop
ad	Data(sp-)←Data(ad0	
orl ad,{#d,a}	a v {…}→ Data[ad]	xch a,{ri,@rj,ad} a←→{.....}
xrl a,{ri,@rj,#d,ad}		xchd a,@rj a(3-0)←→@rj(3-0)
xrl ad,{#d,a}		swap a a(3-0)←→a(7-4)
clr a		-
cpl a	не(a)	
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	

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)

обозначение бита smod,sm0,....

6

/bit - инверсия бита
rel - <метка>=смещение PC в доп коде

Приложение 2.

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 на разных этапах.

(New project → выбрать Device → имя сохранить в своем каталоге)
Manage component →включить файл в проект)

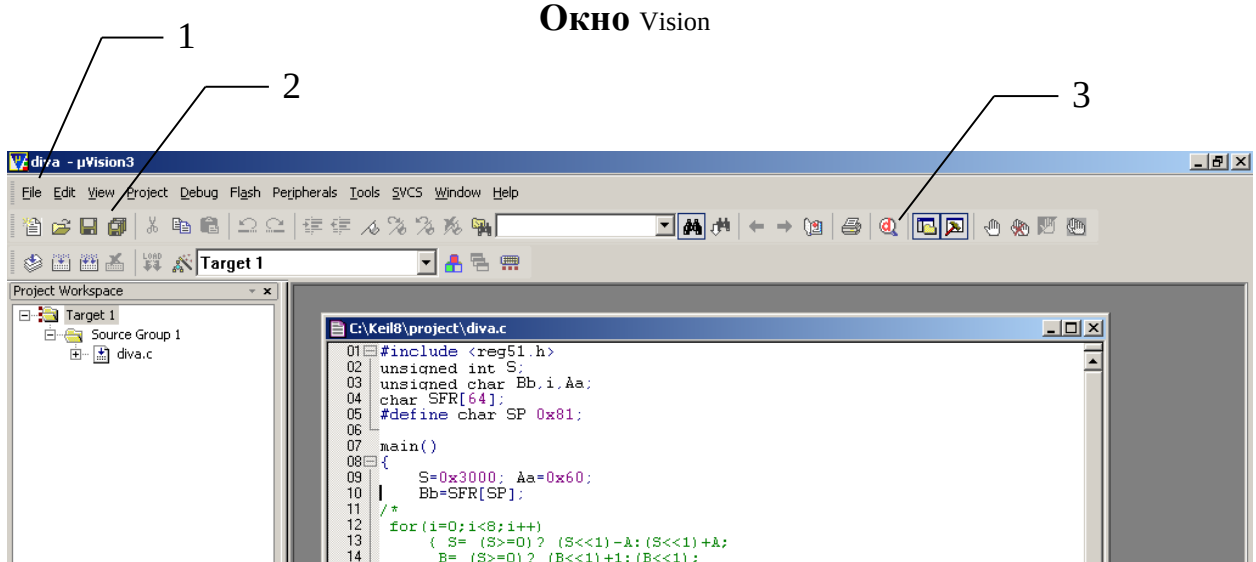
3)Синтаксический контроль.(**Compile**)

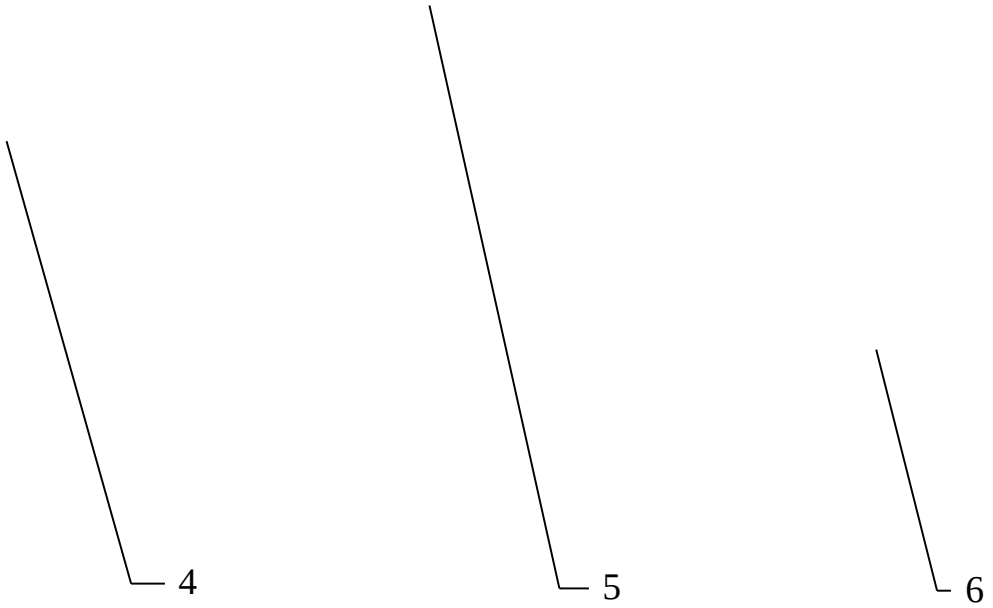
4) Компиляция программы в объектный код (HEX-файл и LIST-листинг) (**Build)**.

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**

File

New - редактирование текстовых файлов

6

Open -
Close -
Save - все остальные имеют стандартное назначение

Edit - имеют стандартное назначение

Project

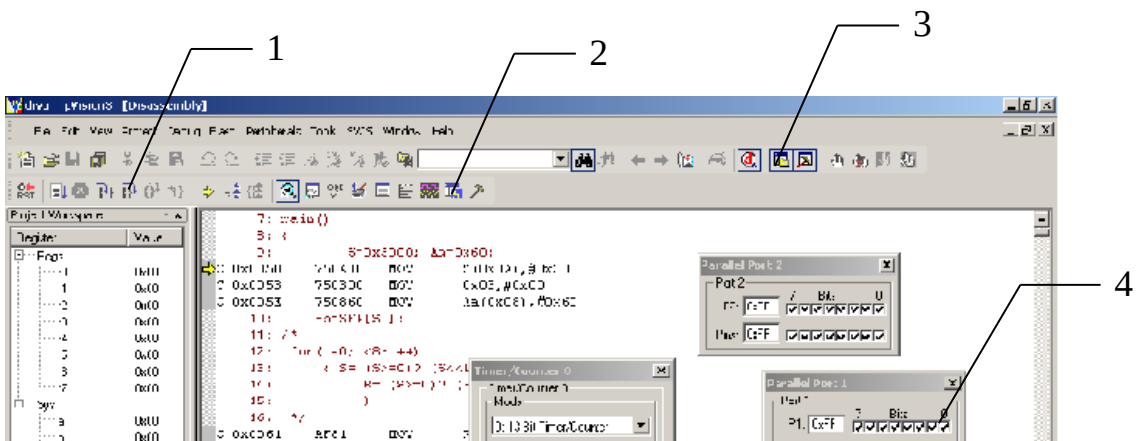
New → µVision project (создать проект)
Import
Open
Close

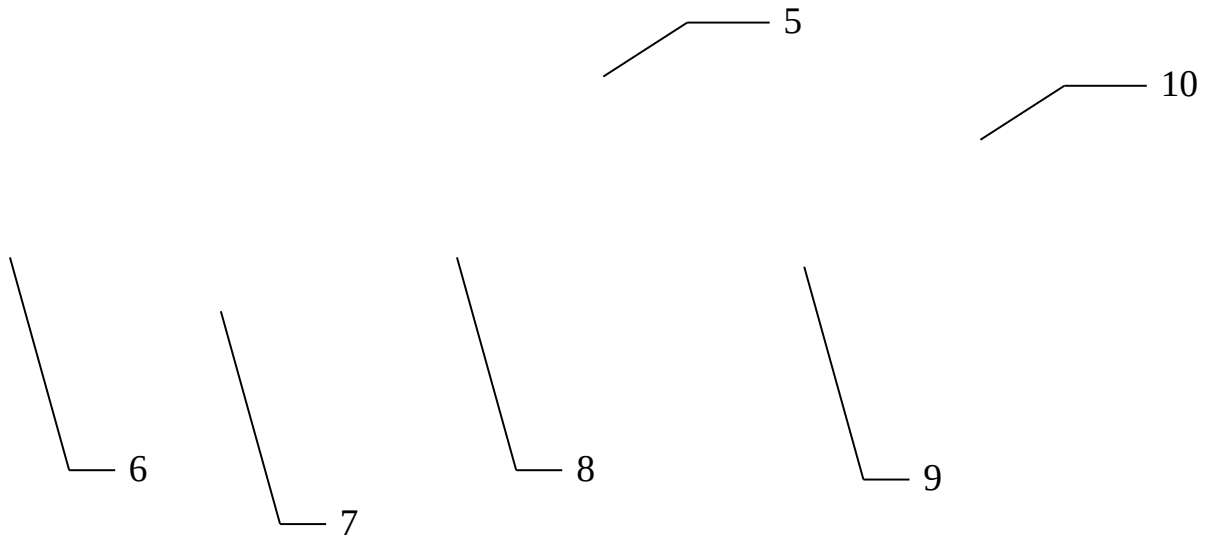
Manage → компоненты, окрестности (в проект включить файл)
Select Device → библиотека элементов
Options → настройки параметров компиляции и загрузки
Device – выбор модуля
Target - выбор частоты MCU
Out - вывод HEX-кода
List – вывод листинга .lst
C51 – размещение таблицы векторов
L51 – размещение программы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

Build - синтаксический разбор и линкирование
Translate –синтаксический контроль

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.

В меню Анализатора выбираем форму оценивания
Grafic Perfomens Analizer

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

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

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.

6

Приложение 3.

Вопросы по курсу лабораторных работ к зачету и экзамену.

1.Структура и возможности системы Кейл.
1. Программная модель MCS51 в С51.
2. Программная модель MCS51 в Ассемблере.
3. Структура памяти – адресация.
4. Иерерхия памяти Ram
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. Преобразование 2/10, 10/2 дробных чисел при вводе-выводе .
10. Символьные преобразования 10/16.
11. Символьные преобразования 16/10.
12. Программа умножения в С51.
13. Программа деления в С51.
14. Вычисление функций с дробными числами – масштабирование
15. Макроассемблирование, применение.
16. Битовые данные, адресация

СПбГУ ИТМО стал победителем конкурса инновационных образовательных программ вузов России на 2007–2008 годы и успешно реализовал инновационную образовательную программу «Инновационная система подготовки специалистов нового поколения в области информационных и оптических технологий», что позволило выйти на качественно новый уровень подготовки выпускников и удовлетворять возрастающий спрос на специалистов в информационной, оптической и других высокотехнологичных отраслях науки. Реализация этой программы создала основу формирования программы дальнейшего развития вуза до 2015 года, включая внедрение современной модели образования.

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

О кафедре

Кафедра ВТ СПбГУ ИТМО создана в 1937 году и является одной из старейших и авторитетнейших научно-педагогических школ России.

Первоначально кафедра называлась кафедрой математических и счетно-решающих приборов и устройств и занималась разработкой электромеханических вычислительных устройств и приборов управления. Свое нынешнее название кафедра получила в 1963 году.

Кафедра вычислительной техники является одной из крупнейших в университете, на которой работают высококвалифицированные специалисты, в том числе 8 профессоров и 15 доцентов, обучающие около 500 студентов и 30 аспирантов.

Кафедра имеет 4 компьютерных класса, объединяющих более 70 компьютеров в локальную вычислительную сеть кафедры и обеспечивающих доступ студентов ко всем информационным ресурсам кафедры и выход в Интернет. Кроме того, на кафедре имеются учебные и научно-исследовательские лаборатории по вычислительной технике, в которых работают студенты кафедры.

Чему мы учим

7

Традиционно на кафедре ВТ основной упор в подготовке специалистов делается на фундаментальную базовую подготовку в рамках общепрофессиональных и специальных дисциплин, охватывающих наиболее важные разделы вычислительной техники: функциональная схемотехника и микропроцессорная техника, алгоритмизация и программирование, информационные системы и базы данных, мультимедиатехнологии, вычислительные сети и средства телекоммуникации, защита информации и информационная безопасность. В то же время, кафедра предоставляет студентам старших курсов возможность специализироваться в более узких профессиональных областях в соответствии с их интересами.

Специализации на выбор

Кафедра ВТ ИТМО предлагает в рамках инженерной и магистерской подготовки студентам на выбор по 3 специализации.

1. Специализация в области информационно-управляющих систем направлена на подготовку специалистов, умеющих проектировать и разрабатывать управляющие системы реального времени на основе средств микропроцессорной техники. При этом студентам, обучающимся по этой специализации, предоставляется уникальная возможность участвовать в конкретных разработках реального оборудования, изучая все этапы проектирования и производства, вплоть до получения конечного продукта. Дня этого на кафедре организована специальная учебно-производственная лаборатория, оснащенная самым современным оборудованием. Следует отметить, что в последнее время, в связи с подъемом отечественной промышленности, специалисты в области разработки и проектирования информационно-управляющих систем становятся все более востребованными, причем не только в России, но и за рубежом.

2. Кафедра вычислительной техники - одна из первых, начавшая в свое время подготовку специалистов в области открытых информационно-вычислительных систем. Сегодня студентам, специализирующимся а этой области, предоставляется уникальная возможность изучать и осваивать одно из самых мощных средств создания больших информационных систем - систему управления базами данных Oracle. При этом повышенные требования, предъявляемые к вычислительным ресурсам, с помощью которых реализуются базы данных в среде Oracle, удовлетворяются за счет организации на кафедре специализированного компьютерного класса, оснащенного мощными компьютерами фирмы SUN, связанными в локальную сеть кафедры. В то же время, студенты, специализирующиеся в данной области, получают хорошую базовую подготовку в области информационных систем, что позволяет им по завершению обучения успешно разрабатывать базы данных и знаний не только в среде Oracle, но и на основе любых других систем управления базами данных.

3. И, конечно же, кафедра не могла остаться в стороне от бурного натиска вычислительных сетей и средств телекоммуникаций в сфере компьютерных технологий. Наличие высокопрофессиональных кадров в данной области и соответствующей технической базы на кафедре (две локальные вычислительные сети, объединяющие около 80 компьютеров и предоставляющие возможность работы в разных операционных средах - Windows, Unix, Solaris), позволило организовать подготовку специалистов по данному направлению, включая изучение вопросов компьютерной безопасности, администрирования, оптимизации и проектирования вычислительных сетей.

7