

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**  
**ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**  
**ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ**



ПОБЕДИТЕЛЬ КОНКУРСА ИННОВАЦИОННЫХ ОБРАЗОВАТЕЛЬНЫХ ПРОГРАММ ВУЗОВ

**П.С. Довгий, В.И. Скорубский**  
**ОРГАНИЗАЦИЯ ЭВМ**  
**Пособие к лабораторным работам**



**Санкт-Петербург**

**2009**

*Довгий П.С., Скорубский В.И.* Организация ЭВМ: пособие к лабораторным работам. – СПб: СПбГУ ИТМО, 2009. – 56 с.

Пособие содержит описание и примеры выполнения лабораторных работ по курсу Организация ЭВМ. В качестве основной технологической базы используется доступная в Интернет Демо-версия Интегрированной системы проектирования (IDE) Keil одноименного подразделения фирмы ARM.

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

Приводится краткое описание программной модели на высоком уровне и уровне микроархитектуры, описание системы команд в Ассемблере.

Работы выполняются на двух уровнях – алгоритмическом с использованием языка С51 и Ассемблерном – в Макроассемблере А51. Используются эффективные и наглядные средства отладки и демонстрации в системе Keil на всех уровнях, в частности, графика Логического Анализатора и интерпретатор внешних событий в виде Сигнальных функций.

Пособие предназначено для студентов по курсу «Организация ЭВМ» для специальностей 230100 «Информатика и вычислительная техника», 230101 «Вычислительные машины, комплексы, системы и сети», 210202 «Проектирование, программирование и эксплуатация ИВС», 230104 «Системы автоматизации проектирования».

Рекомендовано Советом факультета Компьютерных технологий и управления  
\_\_\_\_\_ 2009 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 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.

©Санкт-Петербургский государственный университет информационных технологий, механики и оптики, 2009

## Содержание

	стр.
Введение .....	4
1. Архитектура MCS51 .....	5
1.1. Программная модель в С51 .....	5
1.2. Программная модель на уровне Ассемблера .....	7
1.2.1. Структура памяти, команды обмена данными .....	8
1.2.2. Арифметические и логические операции .....	10
1.2.3. Команды управления программой .....	12
2. Архитектура ЭВМ на программном уровне .....	12
2.1. Ввод-вывод чисел с фиксированной точкой .....	13
2.2. Иерархия памяти ЭВМ .....	15
2.3. Двоичная арифметика .....	19
2.4. Вычисления функций .....	22
2.4.1. Вычисление с плавающей точкой .....	22
2.4.2. Вычисление функций с фиксированной точкой .....	24
2.5. Битовые данные .....	27
2.6. Система прерывания .....	28
2.6.1. Внешние прерывания .....	29
2.6.2. Внутренние прерывания при переполнении таймеров .....	31
3. Программное управление вводом-выводом в ЭВМ .....	36
3.1. Пульт ввода-вывода оператора .....	36
3.2. Работа ADC преобразователя в ЭВМ SAB515/535 .....	40
3.3. Последовательный интерфейс .....	43
Литература .....	47
Приложение 1. Система команд MCS51 – мнемкоды .....	48
Приложение 2. Интегрированная система программирования и отладки Keil .....	51
Приложение 3. Вопросы по курсу лабораторных работ к зачету и экзамену .....	54

## Введение.

Цикл лабораторных работ связан с изучением программной модели микроЭВМ (микроконтроллеры, micro-computer, micro-controller unit – **MCU**) и средств автоматизации программирования.

Основное отличие MCU от микропроцессоров общего назначения является их прикладная ориентация и применение во встроенных системах контроля, управления и измерений. В микроконтроллерах (MCU) интегрированы и согласованы на уровне стандарта разнообразные интерфейсы и средства прямого управления периферией.

Выбор микроконтроллера MCS51 [1] фирмы Intel для исследований обуславливается популярностью, открытой архитектурой, многообразием расширений и модификаций, сохраняющих ядро MCS51.

Для изучения программной модели решаются разнообразные вычислительные и логические задачи с использованием алгоритмического описания на языке Си и на машинном языке Ассемблере.

Задачи исследования программной модели разделяются по типу:

- прямое программное управление вводом и выводом с использованием параллельных портов;
- машинные арифметические операции с фиксированной точкой;
- вычисления функций в различных форматах данных – с плавающей точкой и фиксированной;
- символьное редактирование данных с использованием нескольких уровней памяти;
- битовые вычисления;
- система прерывания – измерения реального времени с использованием таймеров и внешних прерываний;
- управление клавиатурой в локальных пультах;
- обработка данных при вводе с использованием аналого-цифрового преобразования;
- последовательный канал передачи данных.

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

IDE являются необходимым сопровождением технологии проектирования встроенных вычислительных систем и включают средства компиляции и отладки алгоритмического описания на уровне Ассемблера **A51** и языка **C51**.

## 1. Архитектура MCS51.

Архитектура может быть представлена различными моделями ЭВМ.

**Высокоуровневая программная модель** – совокупность ресурсов памяти, состав средств ввода-вывода, доступные в C51.

**Низкоуровневая программная модель (микроархитектура)** представляет организацию памяти, ввод-вывод и систему команд в Ассемблере А51. Подробное описание системы команд приведено в Help Keil [1] и Приложении 1.

CISC архитектура MCS51 выполняет 102 команды, частота 11.056 МГц, цикл исполнения команды 12 тактов, питание 5 В.

Более сотни клонов нескольких десятков фирм с ядром MCS51 представлены в библиотеке KEIL и отличаются разнообразием периферии, организацией и ресурсами памяти, наличием специальных средств управления питанием, частотой, сбросом.

### 1.1. Программная модель в C51.

Диаграмма MCU представляет иерархию памяти и средства ввода-вывода, которые доступны на программном уровне в виде операторов и резервированных переменных языка C51. Язык C51 является расширением стандарта языка Си с учетом особенностей его реализации в MCS51.

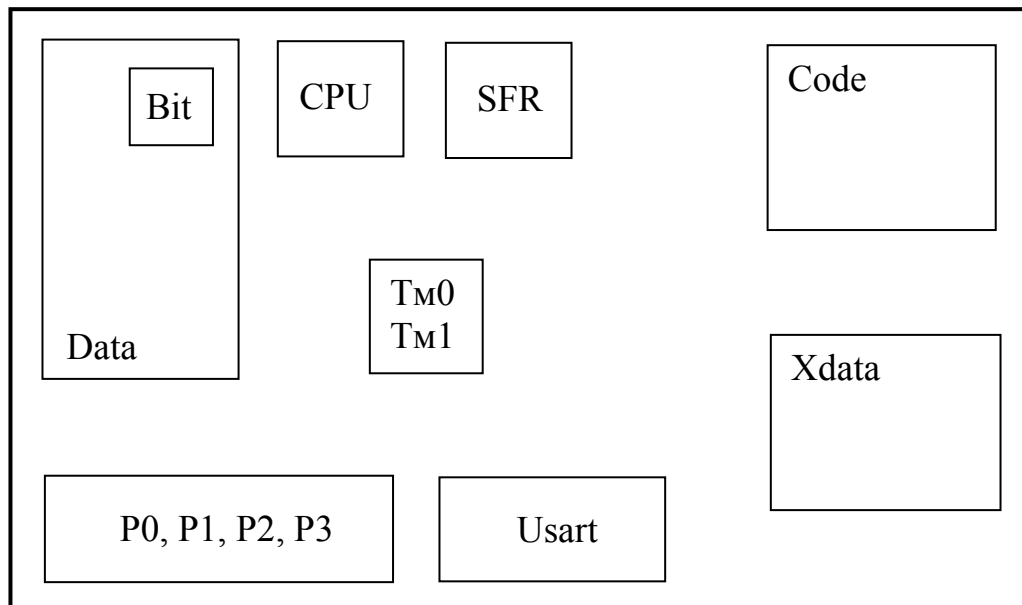


Рис. 1.1. Программная модель ЭВМ в C51.

Традиционно используется конструктивное разделение памяти на внутреннюю (в кристалле MCU) и внешнюю или **расширенную** (используются дополнительные схемы памяти на печатной плате), **оперативную** (быструю) и

неоперативную (большого объема). Все типы памяти отличаются объемом, способом доступа и типами хранимых в них данных.

### Оперативная память данных

**Data** - 128 байт память RAM

**char x1,x2,aa[5];** //переменные в Data, имя – значение

**char \*xx;** //указатель-адрес переменной

**Bit** - 128 бит или 16 байтов в памяти **Data**

**bit x1,x2;** //определение битовых переменных в Data

**char bdata mem** //ячейка в Data с битовой адресацией

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

**SFR** - блок регистров специальных функций –128 байт

Регистры доступны в C51 по именам (P0,P1,P2,..SP, ..TH0,TL0..)

Часть регистров бит-адресуемые, некоторые биты доступны по

именам, а остальные по адресам

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

### Постоянная память

**Code** – 64 Кбайт адресное пространство, доступ – чтение и исполнение команд

**char code aa[]="abcdef";** //текстовая константа

**char code \*aa[]="abcdef";** //указатель-адрес константы

### Расширенная память данных

**Xdata** — запись и чтение. 64 Кбайт адресное пространство.

**char xdata aa[100];**

**char xdata \*aa[];** //указатель-адрес динамического массива

**Ввод-вывод** представлен цифровыми 8-битовыми портами **P0-P3** и последовательным интерфейсом **USART**.

Порты содержат регистр данных, входные и выходные буферные схемы, подключаемые к внешним контактам MCU. При вводе (**char x=P1**) данные из порта сохраняются в памяти и интерпретируются в положительном кодировании двоичными кодами (H~1, L~0). При выводе (**P2=0x55**) данные из памяти записываются в порт и передаются на внешние контакты в положительном кодировании.

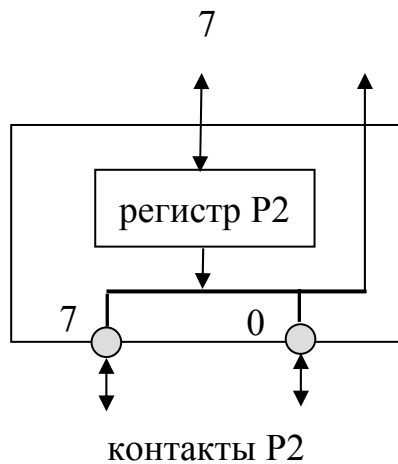


Рис. 1.2. Схема порта.

Порт P0 **двунаправленный** и может в реальной схеме работать в двух направлениях - ввод или вывод в разные моменты времени.

Порты P1,P2,P3 считаются **квазидвунаправленными**, т.е. могут работать в двух направлениях, но в реальных схемах предполагается, что они включены как однонаправленные. Детали их использования в совместном режиме конкретизируются на уровне Ассемблера.

### 1.2. Программная модель на уровне Ассемблера.

Диаграмма обозначает программно-доступные на уровне системы команд или Ассемблера ресурсы компьютера.

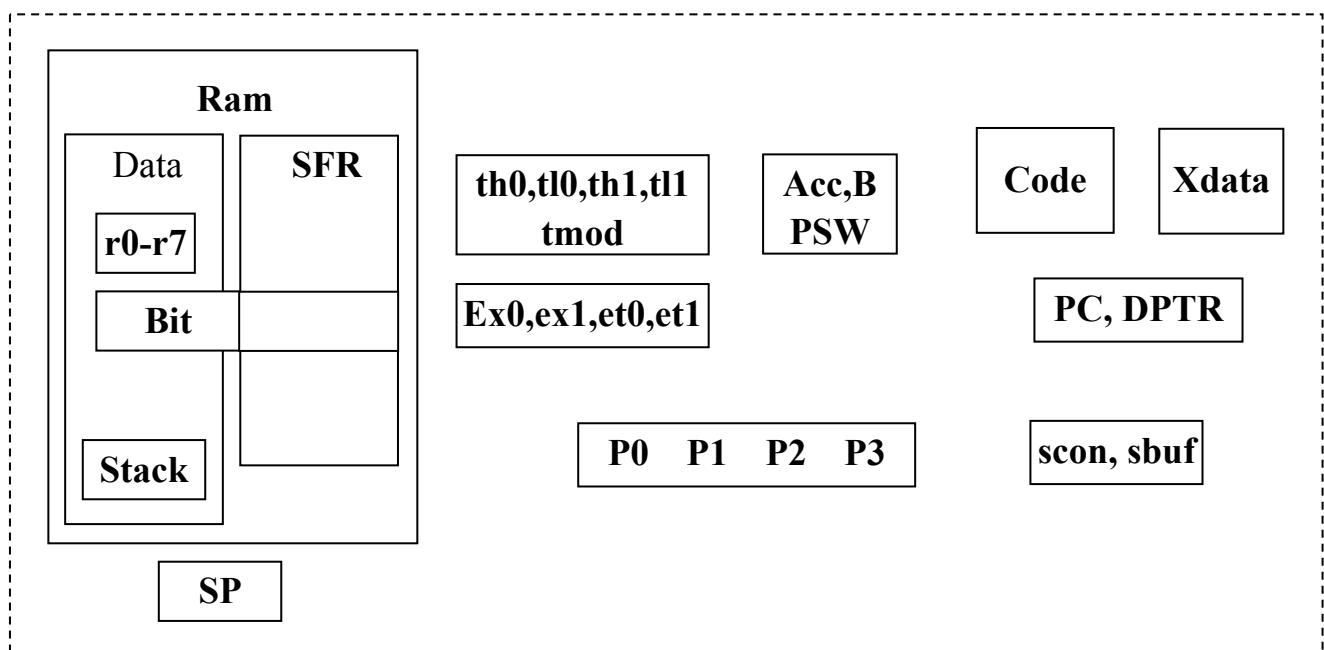


Рис.1.3. Программная модель на уровне Ассемблера.

### 1.2.1. Структура Памяти, команды обмена данными.

#### 1) Регистры неявно доступные

**a(Acc)** – основной регистр-аккумулятор, применяемый во всех арифметических и логических операциях с неявным доступом (**a**). В некоторых командах регистр доступен по адресу **Acc**.

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

Регистр состояния **PSW=C.AC.F0.RS1.RS0.OV.-P** содержит признаки результата арифметических операций – C(перенос, заем), AC – полуперенос, OV(переполнение), P(бит четности), F0(бит пользователя), RS1-RS0 – номер активного регистрового банка.

**PC**- 16-разрядный программный счетчик или регистр адреса команды. При включении питания автоматически сбрасывается. Таким образом, в MCS51 начальный запуск программы с адреса 0000.

**DPTR** – 16-разрядный адресный регистр (Data Pointer) доступа к внешней памяти Code, Xdata.

2) **Регистры общего назначения**  $R_i = \{ R_0, R_1, \dots, R_7 \}$  – активный регистровый банк, доступны 4 банка, совмещенные с начальными ячейками памяти **Data**, активный банк выбирается в регистре PSW.

```
mov a,R0 ; Data(R0)→ A
mov R1,a ; A → Data(R1)
```

3) **Регистры SFR с прямой адресацией (80-FFh)**, 128 байт – управляющие и системные регистры.

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

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

4) Иерархическая **оперативная память данных Data**, включает также регистровую память  $R_i$ , битовую и стековую.

128 байт, прямой адрес **ad=0-7fH**

**dseg at 0x20** ; абсолютный сегмент памяти в Data с адреса 0x20

**mem ds 1** ; переменная 1 байт

**mov a, 55h** ; Data(55h)→ Acc прямая адресация в Data

**mov mem, a** ; a → Data(mem). Mem- адрес ячейки Data

**mov a,@R0** ; Data(R0)→Acc косвенная адресация в R0

**mov @R0,mem** ; Data(mem)→Data(R0)

**mov mem,0x30** ; Data(0x30)→Data(mem)



**a) Bit** – 128 бит, прямой адрес бита 0-7fH, память совмещена с ячейками 0x20-0x2f в Data

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

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

**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**

**b) Stack** - в памяти Data с косвенным доступом через регистр-указатель вершины SP, пре-автоинкремент (+SP) при записи и пост-автодекремент (SP-) при чтении

**push ad**

Например, **push Acc** обозначает Acc → Data(+SP),

**pop ad**

Например, **pop Acc** обозначает Data(SP--) → Acc

При включении и сбросе MCU устанавливается SP=07.

**5) Постоянная память программ и констант Code.**

**mov a, #d** ; Code(PC+) → a - непосредственная адресация

**movc a, @a+pc** ; Code(PC + a) → a ; адресация относительно текущего PC, в ACC индекс

**cseg at 0x40** ; абсолютный сегмент памяти Code с адреса 0x40

**yy: db "abcde"** ; адрес первого байта строки в сегменте

**mov dptr, #yy** ; сохранение адреса

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

**6) Расширенная память данных Xdata**

**xseg at 0x100** ; абсолютный сегмент

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

**mov dptr, #mm** ; адрес в dptr

**movx a, @dptr** ; Xdata(dptr) → A

**movx @dptr, a**

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

Схема доступа к данным в командах **mov** приведена на рис. 1.4. При ограниченных режимах адресации можно выбирать наиболее короткие пути передачи данных при вводе-выводе, при обращении к внешней памяти.

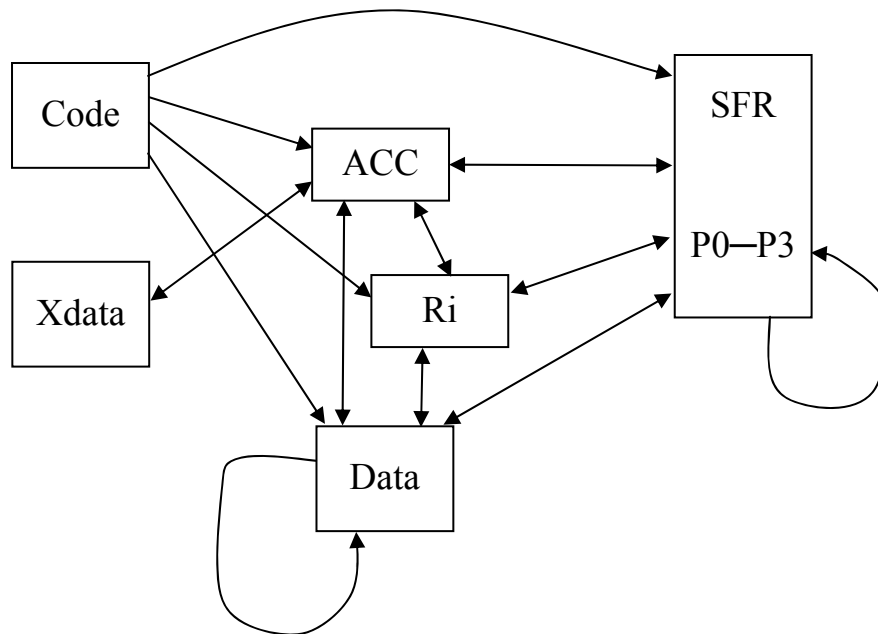


Рис.1.4. Схема адресации в MCS51.

### 1.2.2. Арифметические и логические операции.

#### а) Арифметика 8 битовая двоичная Знаковая/беззнаковая

**add a, {Ri,@rj,#d,ad}** ;  $a + \{..\} \rightarrow a$ , Признаки C,OV,P в PSW, в скобках {..} обозначены режимы адресации второго операнда

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

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

**add a,P2** ;  $a + P2 \rightarrow a$  P2-регистр порта P2

#### б) Беззнаковая арифметика

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

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

**mul ab,**  $a * b \rightarrow b.a$ , признаки  $v=(b \neq 0)$ ,  $0 \rightarrow C$ , P

**div ab,**  $a / b \rightarrow a$ ,  $b = \text{rest}(a/b)$  признаки ov,p

**rrc a,**  $RR(c.a) \rightarrow (a.C)$  признаки C,P

**rlc a,**  $RL(a.C) \rightarrow (C.a)$  признаки C,P

**clr a,**  $0 \rightarrow a$

### с) Десятичная арифметика

Для ускорения ввода и вывода при разделении с фазой обработки данных и в расчетах с данными переменной длины используется десятичная арифметика в форматах с естественной запятой.

В MCS51 работа с десятичными данными поддерживается специальными командами:

**DA a** – десятичная коррекция результатов двоичного сложения или вычитания 2/10 чисел

**swap a** – обмен тетрадами в Асс

**xchd a, @rj** - обмен тетрадами

### д) Логика поразрядная 8 битовая

**andl a, {Ri,@rj,#d,ad}**  $a \& \{..\} \rightarrow a$  признаки р, 0→с,

**andl ad, {#d, a}** ;

**orl a, {Ri,@rj,#d,ad}**  $a \vee \{..\} \rightarrow a$  признаки р, 0→с,

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

**xrl {Ri,@rj,#d,ad}**  $a \# \{..\} \rightarrow a$  признаки р, 0→с

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

**cpl a** ; **not a**

**rr a** ; циклический сдвиг Асс вправо (признак С не изменяется)

**rl a** ; циклический сдвиг Асс влево (признак С не изменяется)

### е) Битовые операции

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

Например, **anl c, /ACC.6**

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

**setb bit,**

**clr bit,**

**cpl C**

### ф) Работа с портами

Передача данных через двунаправленный порт P0 определяется сигналами **записи** или **чтения**, которые являются стробами для внешних устройств и признаками готовности MCU, соответственно, к выводу или вводу через порт. Таким образом, исключается одновременный ввод и вывод.

В квазидвунаправленных портах P1,P2,P3 состояние порта задается программой:

Ввод **char bb=P1;** //чтение с контактов порта **mov bb,P1**

Вывод **P2= bb;** //запись в регистр порта командой **mov P2,bb**

Регистр порта – независимый и может быть использован как буферный для временного хранения данных. С другой стороны, состояние регистра по низкому уровню L не совместимо с высоким уровнем H на контакте. Для ввода с контактов необходимо в соответствующих разрядах регистра установить единицы. Также обращение к порту трактуется различным способом, что необходимо учитывать при работе с битами порта как с входными и независимо – с регистром порта.

```
P1++;           //компилируется как inc P1 ;инкремент регистра P1
P1=P1&0x55; // компилируется как anl P1,#0x55 ; операция с регистром P1
P1=(P1&0x55)+P1; //компилируется как mov a,P1 ; ввод с контактов
                                     anl a,#0x55
                                     add a, P1 ; сложение с контактами и
ВВОД
                                     mov P1,a ; запись в регистр P1
```

### 1.2.3. Команды управления программой.

Команды формируют состояние программного счетчика PC

```
jmp метка ; метка → PC безусловный переход
call метка ; PC → Stack(+SP), метка → PC переход к
подпрограмме
ret ; Stack(SP-) → PC возврат из подпрограммы
jc/jnc метка ,
jz/jnz метка, переход, если ACC (=0)/(!=0)
jb/jnb bit, метка ;
        пример jb ACC.0,start переход по значению бита
djnz {ri,ad}, метка ; [{}]-1, if ({}#0), то метка → PC]
cjne (ri,@rj,ad) ,#d, метка ; if ({}#d) метка → PC;
```

Обзор команд ЭВМ приведен в Приложении 1.

## 2. Архитектура ЭВМ на программном уровне.

Формулируются задания с примером для демонстрации исходных данных и результатов. Требуется предложить алгоритмическое решение и сформулировать его на языке C51. Выполнить компиляцию и отладку в системе Keil.

Привести программу решения этой задачи на языке Ассемблера А51, выполнить отладку в системе Keil.

## 2.1. Ввод-вывод чисел с фиксированной точкой.

Назначение ввода-вывода – обмен данными между памятью ЭВМ и периферией ввода-вывода, подключаемой к контактам портов.

На уровне программной модели в операциях ввода-вывода также выполняются преобразования двоичных кодов данных, которые приводят их к внутреннему представлению в памяти в виде 8-разрядных двоичных чисел.

### 1) Ввод десятичных данных и преобразование в двоичное число.

Машинные преобразования выполняются в двоичной системе пересчетом из 2/10 системы в двоичную.

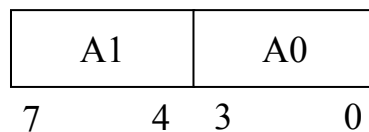


Рис. 2.1. Формат 2/10 числа в порте.

$(A1.A0)_{2/10}$  – двоично-десятичное число, A1 и A0 – старшая и младшая тетрады,  $B_2$  двоичное число после пересчета содержит тоже самое число целых единиц – информация, содержащаяся в любой записи этого числа. Формула пересчета

$$(A1.A0)_{2/10} = (A1 * (10)_2 + A0)_2 = B_2$$

### *Программа в C51*

```
#include <reg51.h> //файл определений регистров MCS51
main()
{   P2= (P1>>4)*10 + (P1&0x0f); //ввод 2/10 числа с порта P1, выделение
    со сдвигом старшей тетрады, вывод двоичного B2 в P2
}
```

В Projects/options/listing задать Assembly Code формирование листинга компиляции в Ассемблере.

C51 COMPILER V8.08 EXP  
PAGE 1

06/10/2009 12:30:19

06/10/2009 12:30:19 PAGE 2

ASSEMBLY LISTING OF GENERATED OBJECT CODE

```
; 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
0008 A4        MUL   AB
0009 FF        MOV   R7,A
000a E590      MOV   A,P1
000c 540F      ANL   A,#0FH
000d 2F        ADD   A,R7
000e F5A0      MOV   P2,A

```

```

; FUNCTION main (END)

```

### *Программа, подготовленная в Ассемблере a51*

```

cseg at 0 ; абсолютный сегмент кода размещается с адреса 0
; в программной памяти
mov a,P1    ; ввод байта данных с порта P1 в аккумулятор
anl a,#0f0h ; выделение старшей тетрады A1
swap a      ; тетрада перемещается в младшие разряды
mov b,#10   ; 10 в регистре множителя
mul ab      ; a=A1*10, b=0
mov b,a     ; сохранить младшие разряды произведения
mov a,P1    ; ввод байта данных с порта P1 в аккумулятор
anl a,#0fh  ; выделение младшей тетрады A0
add a,b     ; a=(A1*(10)2 + A0)2
mov P2,a    ; вывод результата в порт P2
end      ; псевдокоманда ассемблера- завершение текста

```

Сравниваем листинг C51-программы с программой, разработанной в Ассемблере: количество байтов после компиляции C-программы - 17, в ассемблере программа занимает 18 байтов.

Компилятор оптимально выбирает для временного хранения произведения регистр R7. Если использовать эту подсказку, то программа в ассемблере сократится до 16 байт.

## **2) Вывод чисел в порт (Перевод 2→10)**

Вычисления выполняются в двоичной системе делением на основание  $P3=((P2/10)\ll 4) + P2\%10$ ; где  $A1=B/10$ ;  $A0=B\%10$

**Задания:**

Совместить перевод  $10 \rightarrow 2$  и  $2 \rightarrow 10$  в одной программе.

Сравнить листинги .lst программ в C51 и A51 и пояснить различия в двух программах.

**2.2. Иерархия памяти ЭВМ.**

Задана текстовая строка-константа в ASCII 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 доступом.

**Пример.**

Символьное десятичное число преобразовать в символьное двоичное.

“123”  $\rightarrow$  (“01110101“)<sub>2</sub>

Ascii-код десятичной цифры занимает байт – в старшей тетраде добавляется 0x30.

Десятичное число размещается как символьная константа в программной памяти Code, преобразуется в шестнадцатеричное во внутренней регистровой памяти Data и результат записывается во внешнюю память данных Xdata.

**Программа в C51****1) Прямая адресация**

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

main()
{ 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
    { yу[i]= (x&0x01) ? ‘1’ : ‘0’;
      x=x>>1;
    }
}
```

```

    }
    while(1); //динамический останов
}

```

## 2) Программа на C51 – косвенный доступ 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 * уу; //указатель на текстовую переменную

main()
{
    for (i=0; i<3; i++) x=x*10+(*уу++&0x0f);

    for(i=7;i>=0; i--)
        { *уу++= (x&0x01) ? '1' : '0';
          x=x>>1;
        }
    while(1); //динамический останов
}

```

## Ассемблер

**Dseg** at 8 ;сегмент данных в Data

X: ds 1

**Xseg** at 0 ;сегмент данных в Xdata

Yу: ds 8 ;char xdata уу[8]; //результат преобразования во внешней  
памяти

**Cseg** at 0 ;начало программного сегмента, после сброса или включения  
питания

Jmp start ;подразумевается команда старта на начало программы

Y: db "123" ;текстовая константа в ASCII-коде

;for (i=0; i<3; i++) - комментарии из программы на C

start: clr a ;подготовка параметров цикла

Mov x,a

Mov r0,#3

Mov DPL,#уу ;ограничимся младшим байтом адреса – старший DPH=0

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

Mov a,x

Mov b.#10



```

Mul ab
Mov b,a
Clr a
movc a,@a+dptr
Inc dptr
Anl a,#0x0f
Add a,b
Mov x,a
Djnz r0,cikl

Mov r0,#8
Cikl2:      ;yy[i]= (x&0x01) ? '1' : '0';
Mov dpl,r0
Mov a,x
Anl a,#01
Orl a,#0x30
Movx @dptr,a
;x>>1
Mov a,x ; x=x>>1;
Rr a
Mov x,a
Djnz r0,cikl2
Nop
end

```

### Задания

1. Упорядочить текст лексикографически → в порядке возрастания ASCII-кода  
 “This programmer” → “aaghimmootTrrs”
2. Вставить пробелы после символа “r”  
 “This programmer” → ”r” → “This pr ogr ammator”
3. Заменить прописную букву “x” на заглавную в тексте  
 “This programmer” →”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 двоичное число преобразовать в символьное десятичное.

$$"123" \leftarrow ("01110101")_2$$

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

$$"590" \rightarrow "0x24e"$$

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

$$"590" \leftarrow "0x24e"$$

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

$$"-25,023" \rightarrow "e+2 - 0.25023"$$

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

$$"256,54" + "75,56" = "332,10"$$

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

$$"01011010" \rightarrow 00\ 11\ 00\ 11\ 11\ 00\ 11\ 00\ (+)$$

10 10 10 10 10 10 10 10 синхросигнал

$$\rightarrow "10\ 01\ 10\ 01\ 01\ 10\ 01\ 10" \text{ Манчестерский код}$$

Восстановление символьного двоичного кода из Манчестерского

$$"1001100101100110" \text{ Манчестерский код}$$

$$\rightarrow "0\ 1\ 0\ 1\ 1\ 0\ 1\ 0" \text{ двоичный код}$$

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

$$"132" \rightarrow "2\ 3,0, 2"$$

7. Шифрование и дешифрование Гронсфельда

таблица символов {a,b,c,d,e,f, ...}

нумерация 0 1 2 3 4 5 6

ключ {3,1,2,0,6, ...}

Пример

$$"cadda" \leftrightarrow "cdaad"$$

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

$$"0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1" \leftrightarrow "2\ 1\ 0\ 0\ 2\ 1\ 2\ 0"$$

9. 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”

10. 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

### 2.3. Двоичная арифметика.

а) **Умножение** по методу вычисления произведения для дробных чисел с фиксированной точкой перед старшим разрядом

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

Рекуррентная формула вычисления дробного произведения

$$(*) \quad S_{i+1} = 2^{-1}(S_i + Ab_{n-i}), \quad S_0 = 0$$

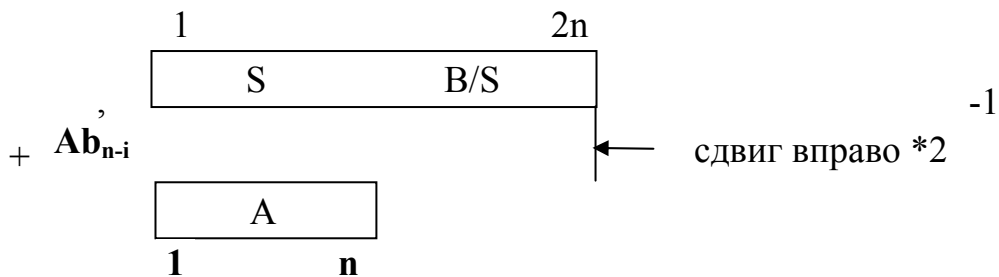


Рис. 2.2. Схема умножения в Ассемблере.

Множитель размещается в младших разрядах произведения, младший разряд B/S регистра — текущее значение  $b_{n-i}$

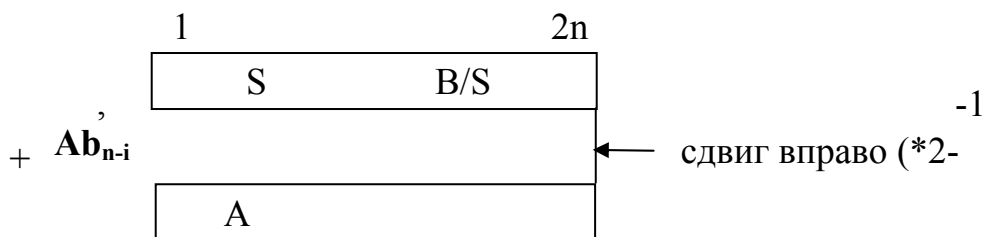


Рис. 2.3. Схема умножения в Си.

Здесь выровнены форматы для согласования форматов операндов при сложении.

При вычислении в Си признак переноса при суммировании не контролируется, поэтому либо диапазон чисел при выполнении операций на один разряд меньше, либо используются расширенные форматы.

В Ассемблере эти ограничения отсутствуют и можно получить правильный результат во всем диапазоне 8-разрядных чисел.

### б) Беззнаковое деление 16-разрядного числа на 8-разрядное.

Алгоритм деления дробных чисел с фиксированной точкой перед старшим разрядом реализовать по рекуррентной формуле – обратной по отношению к формуле умножения : если  $S=A*B$ , то  $B=S/A$ .

Если  $S_n=2^{-1}(S_{n-1}+Ab_1)$ , то  $b_1=1$ , если  $S_{n-1} = 2S_n-A \geq 0$ , иначе  $b_1=0$ .

И так далее на каждом шаге  $b_i=1$ , если  $S_{n-i} = 2S_{n-i-1}-A \geq 0$ , иначе  $b_i=0$ .

Изменяя нумерацию остатков, приходим к следующей рекуррентной формуле деления без восстановления остатка – сначала определяется знак разности, остаток сохраняется, если знак положительный.

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

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

В схеме вычислений на Си форматы операндов выровнены для выполнения вычитания.

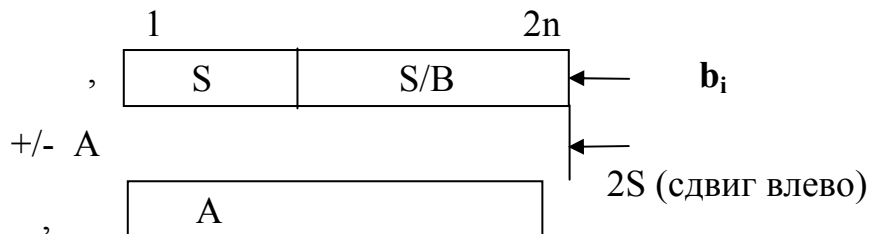


Рис.2.4. Схема деления в Си без восстановления остатка.

Регистр S/B совмещает младшие разряды делимого при сдвиге и разряды частного.

В программе на языке С необходимо контролировать знак остатка в явном виде, при этом сужается диапазон значений делимого и делителя или необходимо использовать расширенные форматы.

```
Main()
{ Int S=P1<<8 ;
  Int A=P2<<8 ;
  For (char i=0 ; i<8; i++ ;)
```

```

P3=S= ((S<<1)-A)>=0) ? (S<<1)-A +1 : S<<1 ;
}

```

В Ассемблере можно контролировать как бит признака  $C$ , так и старшие биты операндов, что позволяет работать без знаков в полных форматах ( $n$ -разрядный делитель и  $2n$ -разрядное делимое) значений операндов.

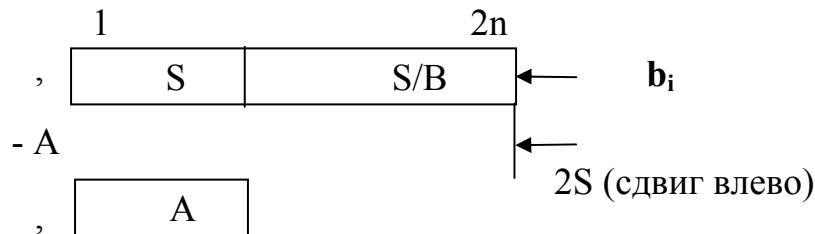


Рис. 2.5. Схема деления в Ассемблере без восстановления остатка.

Признак отрицательного остатка  $C \& \neg S[1]$ , где  $C$  - заем при вычитании и при сдвиге регистра  $S$  в старшем бите  $S[1]$  единица.

#### в) Извлечение квадратного корня из 16-разрядного целого.

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  $S_0 \geq (x_0 + 0.b_1)^2 = (x_0 + 0.1)^2 = x_0^2 + x_0 + 0.01$ ,  $x_1 = 0.b_1$  и  $b_1=1$ -старшая цифра дробного корня, если

$$S_1 = S_0 - 0.01 \geq 0$$

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

Предположим  $x_2 = b_1, b_2$

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

$$Q_2.S_2 = 4Q_1.S_1 - x_1.01 = x_1^2 \geq 0$$

Пусть  $b_2=1$  и  $Q_2.S_2 \geq 0$ , 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  $x_2 = b_1, b_2$  и получим  $x_3 = b_1 b_2, b_3$

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

$$Q_3.S_3 = 4Q_2.S_2 - x_2.01 = x_2^2 \geq 0$$

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

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

### Схема вычисления в Ассемблере

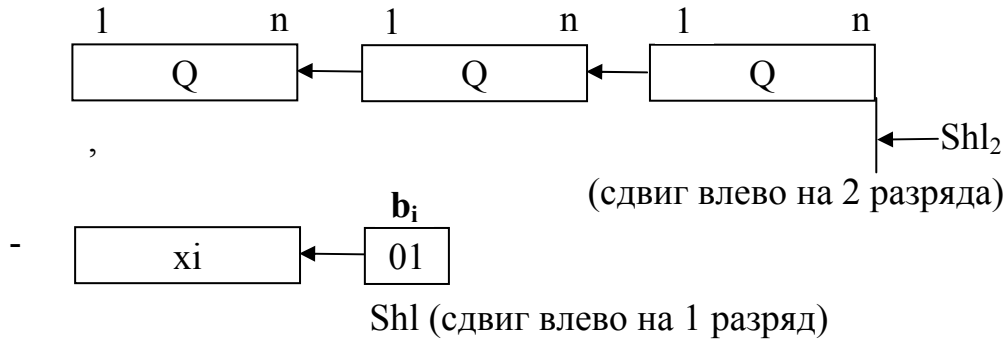


Рис. 2.6. Схема извлечения корня в Ассемблере без восстановления остатка.

#### Задание:

Разработать и отладить программы в С51 и на Ассемблере а51. Сравнить объем программ и время выполнения.

Программу извлечения корня выполнить в С51.

## 2.4. Вычисления функций.

### 2.4.1. Вычисление функции с плавающей точкой.

Выбрать из **таблицы заданий** (ниже) вариант функции для вычислений. В этом разделе подготовить программу вычислений с плавающей точкой с использованием библиотеки **math.h** языка С51.

#### Пример.

Используя библиотечную функцию в С51 вычислить значения  $\sin(x)$  в диапазоне аргумента  $0-360^\circ$ . 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. Цифро-аналоговое преобразование из дискретной формы в аналоговую (график) выполняет Анализатор.

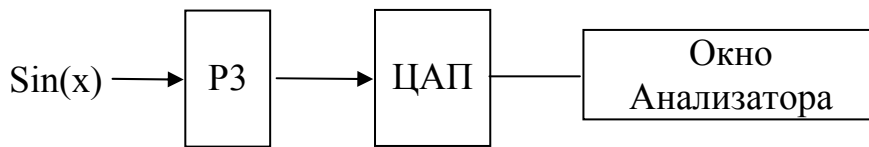


Рис.2.7. Схема работы Анализатора.

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

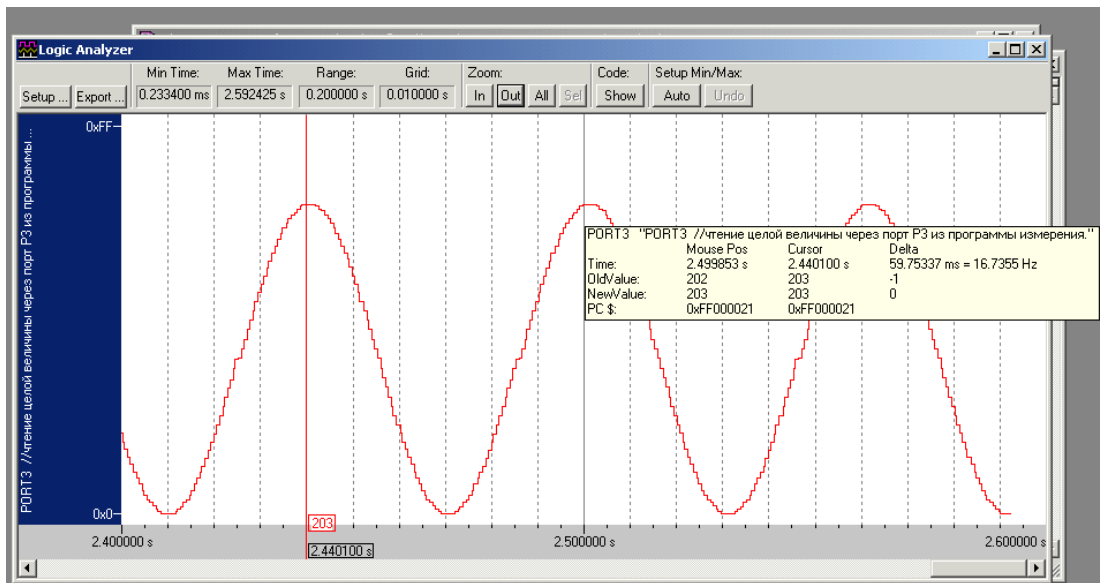
```

#include <reg51.h>
#include <math.h>
float x;
unsigned char xdata y[100],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)
        y[i++]=P3=sin(x)*100+100;    //масштабное преобразование
    переводит дробное в целое с точностью 2 знака после запятой и смещает в
    положительную область значений

    /* цикл вычисления функции и вывод ее значения из таблицы в графике для
       измерений среднего времени вычисления функции*/
    while(1)
    { i=0;
      for(x=0; x<6.28 ;x+=0.0628)
          {sin(x); P3=y[i++];}
      }
}
  
```

Командой **LA P3** значение в Анализатор передается через порт P3.

## График функции в окне Анализатора.



### 2.8. График функции в окне Анализатора.

Объем программы – 1.5 Кбайт, время вычисления одного значения  $59.7 \text{ мс}/100 = 0.59 \text{ мс}$ .

### 2.4.2. Вычисление функции с фиксированной точкой.

Вычисление с фиксированной точкой требует меньше времени, программа вычисления существенно меньше. Это может быть полезно при организации вычислений в ЭВМ с ограниченными ресурсами памяти в реальном времени.

Функция обычно задана разложением в ряд Тейлора, например,

$$\sin x \sim x/1 - x^3/3! + x^5/5! - x^7/7! + \dots \text{ при всех } x$$

Аргумент - дробное число в диапазоне 0- 1.57 радиан.

Система команд ЭВМ поддерживает вычисления с целыми числами. Для применения этих же команд арифметики выполняется Масштабирование дробных значений, которое переводит диапазон дробных в диапазон целых

$$x' = x * m;$$

Для сохранения масштаба результата используем следующие преобразования

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

**Вычисления с целыми в С51** выполнить по схеме Горнера

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

или по формуле с общим членом ряда,

$$S_{i+1} = S_i + A_i * x/m * x/(m(i+1)(i+2))$$



Контролировать **формат данных 8 бит, промежуточный результат 16 бит**. Число членов ряда не более четырех.

**Дальнейшее** сокращение объема памяти и времени вычислений возможно с переходом на программирование в МакроАссемблере или в смешанном программировании, где стандартные обращения и организация памяти в С соединена с быстрыми ассемблерными подпрограммами. Технология программирования может использовать **макрокоманды** для расширения системы команд – добавлением к списку команд макрокоманд, например, типа RR(регистр-регистр), SS(память-память), RS(регистр-память), 16-битовой арифметики – сдвиги влево и вправо и др.

Структура **макроопределения**

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

<тело макроопределения – список ассемблерных команд с параметрами)

**ENDM**

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

**Примеры,**

макрокоманда RR: **movr r0,r1** (эквивалентная в MCS51 команда **mov 00,r1**)

макроопределение **movr MACRO ri,rj**  
                                   **mov a,rj**  
                                   **mov ri,a**  
                                   **ENDM**

Макрокоманда RS **sadd r2, Mem-** имя в памяти Data

макроопределение **sadd MACRO ri, SS**  
                                   **mov a,ri**  
                                   **add a, SS**  
                                   **mov ri,a**  
                                   **ENDM**

Макрокоманды сокращают текст программы и позволяют использовать систему команд, более подходящую к алгоритму решаемой задачи.

**Задания.**

- 1) Для заданных функций разработать программу вычисления функции с плавающей точкой, вывести график, измерить среднее время вычисления одного значения и объем программы.
- 2) Разработать программы вычисления с фиксированной точкой в С51 и в ассемблере. Вычисление функции - в целых 8-разрядных числах в диапазоне аргумента 0 – 1.0 и с использованием макрокоманд и

подпрограмм. Измерить среднее время вычислений и объем программы. Привести график изменения этих параметров для трех способов.

### Варианты Задания по разделу 2.2.

1.  $1/(1-x) \sim 1 + x + x^2 + x^3 + \dots$   $-1 < x < 1$  (сходимость ряда)
2.  $1/(1+x) \sim 1 - x + x^2 - x^3 + \dots$   $-1 < x < 1$
3.  $(1+x)^{0.5} \sim 1 + x/2 - x^2/(2*4) + 1*3*x^3/(2*4*6) - 1*3*5*x^4/(2*4*6*9) - \dots$   
 $1 \leq x \leq 1$
- $a^x \sim 1 + (\ln a)/1*x + (\ln a)^2 x^2/2! + (\ln a)^3 x^3/3! + \dots$  при всех  $x$   
 $a > 1; \ln a(10*a/10) = \ln 10 + \ln(a/10) = \ln 10 + \ln(1-(1-a/10))$
4.  $a=2$
5.  $a=1/2$
6.  $\cos x \sim 1 - x^2/2! + x^4/4! - x^6/6! + \dots$  при всех  $x$
7.  $\operatorname{tg} x \sim x + x^3/3 + 2x^5/15 + 17x^7/315 + 62x^9/2835 - \dots$   $-\pi/2 < x < \pi/2$
8.  $\operatorname{ctg} x \sim 1/x - (x/3 + x^3/45 + 2x^5/945 + 2x^7/4725 + \dots)$   $-\pi < x < \pi$
9.  $\ln(1+x) \sim x - x^2/2 + x^3/3 - x^4/4 + x^5/5 - \dots$   $-1 < x < 1$
10.  $\ln(1-x) \sim -x - x^2/2 - x^3/3 - x^4/4 - x^5/5 - \dots$   $-1 < x < 1$
11.  $\arcsin(x) \sim x + x^3/(2*3) + 1*3*x^5/(2*4*5) + 1*3*5*x^7/(2*4*6*7) + \dots$   $-1 < x \leq 1$
12.  $\operatorname{arctg}(x) \sim x - x^3/3 + x^5/5 - x^7/7 + \dots$   $-1 < x \leq 1$
13.  $(1-x)^{0.5} \sim 1 - x/2 - x^2/(2*4) - 1*3*x^3/(2*4*6) - 1*3*5*x^4/(2*4*6*9) - \dots$   $-1 \leq x \leq 1$
14.  $(1+x)^{1/3} \sim 1 + x/3 - 2x^2/(3*6) + 2*5*x^3/(3*6*9) - \dots$   $-1 \leq x \leq 1$
15.  $(1+x)^{3/2} \sim 1 + 3x/2 + 3x^2/(2*4) - 3x^3/(2*4*6) + 9x^4/(2*4*6*8) - \dots$   $-1 \leq x \leq 1$
16.  $\operatorname{arsh}(x) \sim x - x^3/(2*3) + 1*3*x^5/(2*4*5) - 1*3*5*x^7/(2*4*6*7) + \dots$
17.  $\operatorname{ch}(x) \sim 1 + x^2/2! + x^4/4! + x^6/6! + \dots$
18.  $\operatorname{sh}(x) \sim x/1 + x^3/3! + x^5/5! + x^7/7! + \dots$

$$19. Si(x) \sim x - x^3/(3*3!) + x^5/(5*5!) - x^7/(7*7!)+$$

$$20. Ci(x) \sim 1 - x^2/(2*2!) + x^4/(4*4!) - x^6/(6*6!) +$$

## 2.5. 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 c51

```
#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.  $z=/x_1(x_2 \vee /x_3) \vee x_1x_4$
4.  $z=(x_1 \vee /x_2x_3)/(x_2 \vee x_4)$
5.  $z= /y_1 \vee /y_2(y_1x_1 \vee /x_2)$
6.  $z=(x_1 \vee /x_3x_4)/(x_1 \vee x_2)$
7.  $z=/y_1x_2 \vee y_2(/x_1 \vee /x_2)$
8.  $z=(/x_1 \vee x_2)(x_1x_3 \vee /x_4)$
9.  $z=(x_1y_1 \vee /x_2y_2)/(x_2 \vee y_1)$
10.  $z=(/x_1 \vee y_1) (x_2y_2 \vee /y_1)$
11.  $z=y_1(/y_2 \vee /y_3) \vee /y_1y_4$
12.  $z=(y_1 \vee /y_2y_3)/(y_2 \vee y_4)$
13.  $z=/y_1 y_2 \vee /x_1x_2(y_1 \vee /y_2)$
14.  $z=x_1y_1 \vee /x_2(/y_2 \vee /x_1)$
15.  $z=(x_1 \vee /x_2 \vee /x_3x_4)/(x_1 \vee /x_4)$

### 2.6. Система прерывания.

Система прерывания ЭВМ обеспечивает контроль “случайных” внутренних и внешних событий.

Схема прерывания фиксирует внешние или внутренние сигналы, обозначающие события, и вызывает подпрограмму принятия решений и их обработки.

В частном случае, **внешние сигналы** фиксируют события, связанные с завершением асинхронных процессов в периферии и используются при вводе/выводе данных, если они сигнализируют о ее готовности для ввода-вывода.

Например,

– при вводе с клавиатуры – сигналы нажатия клавиш,

– при вводе с аналого-цифровым преобразованием входного сигнала (АЦП)-сигнал завершения преобразования.

**Внутренние сигналы и события** связаны, например, с завершением или началом временных интервалов, с достижением контрольных точек в программе.

### 2.6.1. Внешние прерывания.

Внешние события в MCU MCS51 представлены сигналами-запросами прерывания на входных портах P3.2=INT0 и P3.3=INT1.

Таблица 2.1.

1	2	3	4	5	6	7
p3.2 /INT0	IT0	IE0	EX0	0	0	03
p3.3 /INT1	IT1	IE1	EX1	2	2	13h

- 1 – входные сигналы прерываний.
- 2 – тип прерываний IT0=1 – выбирается Н/L фронт входного сигнала. Виртуальные внутренние сигналы прерывания или программные прерывания могут быть сформированы программно при выводе в порты P3.2 или P3.1. Тогда изменение состояния контакта схема порта трактуется как внешнее событие.
- 3 – бит регистра запросов прерываний (IE=1 запрос). Бит сбрасывается при входе в прерываний.
- 4 – Маска разрешения прерывания (EX0=1 разрешено).
- 5,6 – Номер (приоритет) прерывания – чем меньше номер, тем выше приоритет.
- 7 – Адрес-вектор в таблице векторов прерываний в памяти Code.

Виртуальные внутренние сигналы прерывания или **программные прерывания** могут быть сформированы программно при выводе в порты P3.2 или P3.1. Изменение состояния контакта порта можно рассматривать как внешнее событие.

#### Работа системы прерываний.

1. Если бит маски и бит разрешения прерываний EA равны единице, на регистре запросов аппаратно фиксируются запросы прерывания.

2. В конце текущей команды выбирается наиболее приоритетный запрос и формируется вызов в памяти Code подпрограммы обработки прерывания через вектор прерывания. Выбранный запрос сбрасывается при входе в прерывание.

3. В подпрограмме сохраняется состояние программы (ACC, PSW, регистры), выполняется некоторая функция, восстанавливается состояние и происходит возврат в прерванную программу.

#### **Управление прерываниями:**

1. Установить маску прерывания и бит разрешения прерываний EA.
2. Выбрать тип прерывания (по спаду H/L или по уровню L) для внешних сигналов INT0, INT1.
3. Сформировать вектор прерывания в таблице векторов по фиксированному адресу – (jmp на программу обработки в Ассемблере).
4. Подготовить программу обработки, которая завершается командой возврата из прерывания reti (в Ассемблере).
5. Основная программа и программа обработки прерываний должны быть смещены в памяти Code на размер таблицы векторов прерывания (в опциях C51 проекта Кейл и в сегментах для Ассемблера).
6. В программе на Ассемблере резервировать Стек для прерываний и подпрограмм (по умолчанию в C51 назначается **SP=07**).
7. В программе на Ассемблере предусматривается сохранение состояния (контекста) программы и восстановление. Для сохранения и восстановления активных регистров r0-r7 могут быть использованы два варианта – сохранение регистров в Стекe и сохранение переключением регистрового банка.

Обращение к подпрограмме в C51

```
Void Ex00(void) interrupt 0 using 1
{
}
```

**Ex00**- имя подпрограммы обработки прерывания.

**Interrupt** – служебное слово-признак функции и 0-номер прерывания.

**Using 1** – регистровый банк 1 используется в подпрограмме.

Указывая номер банка регистров, автоматически сохраняем **контекст** программы из регистров общего назначения текущего банка. При этом в программе обработки прерывания используется новый регистровый банк. При выходе из программы прерывания автоматически восстанавливается регистровый банк и регистровый контекст на момент прерывания.

В языке C51 шаги 3-7 выполняются автоматически, в Ассемблере программируются.

#### **Пример.**

Программа обработки внешнего прерывания INT0 (ввод по прерыванию)

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

```
intt0() interrupt 0 //подпрограмма обработки прерывания, ввод по готовности
{ выполнить ввод с порта 2/10 числа и преобразовать в двоичное }
```

```

main()
{ EX0=1;    //маска внешнего прерывания INT0
  IT0=1;    //тип прерывания Н/Л фронт на входе P3.2=INT0
  EA=1;     //разрешение прерываний
  while(1); //ожидание прерываний в динамическом останове
}

```

Ассемблер.

```

iseg at 30h ;сегмент памяти для стека
Stack ds 4 ; 4 байта для стека

```

```

cseg at 0 ;стартовый адрес
jmp start

```

```

cseg at 3h ; вектор внешних прерываний INT0
jmp ex00

```

```

cseg at 40h
start: mov SP,#Stack-1 ;начало программы
      setb EX0 ;маска
        setb IT0 ; тип прерывания
        setb EA ; разрешение прерывания
Cikl: add a,#0x100
      Jmp cikl

```

```

ex00: push ACC ;сохранить ACC
      push PSW ;сохранение состояния
      mov a,P2 ; подпрограмма принятия решения по прерыванию
      Add a,P1
      Mov P0,a
      Pop PSW
      Pop ACC
      Reti

```

**end**

### 2.6.2.Внутренние прерывания при переполнении таймеров.

Внутренние прерывания могут быть сформированы последовательным каналом ввода-вывода USART и двумя таймерами.

**Таймеры** – счетчики реального времени, задаваемого стабильным кварцевым генератором частоты. В MCS51 принят основной цикл выполнения команды 12 тактов генератора частоты. Команды выполняются за 1-2 цикла.

Один цикл является единицей времени, отсчитываемой таймером. Если установить частоту генератора 12 МГц, то частота отсчета 1 МГц и единица времени 1 мкс.

Контроль продолжительных промежутков, измерение реального времени выполняются с использованием прерываний от таймеров. Записываем в таймер константу (– С), тогда отсчет до прерывания выполняется за С мкс.

### Таблица параметров внутренних прерываний.

Таблица 2.2.

1	2	3	4	5	6	7
		TF0	ET0	1	1	0bh
		T1vR1	ES	4	4	23h
		TF1	ET1	3	3	1bh

Запросы прерываний **TF0**, **TF1** формируются при переполнении таймеров и установкой по программе.

**Управление 16-битовыми таймерами** задается битами управляющего регистра **TMOD**

Таймер 0 представлен двумя регистрами SFR - **TH0.TL0**

Таймер 1 - регистрами **TH1.TL1**

- a) Выбор режима 16-бит счетчика таймера 1 **TMOD[5.4]=01**  
таймера 2 **TMOD[1.0]=01**
- b) разрешение счета таймера 0 - бит **TR0=1**  
разрешение счета таймера 1 - бит **TR1=1**
- c) режим **Gate** таймер считает, пока на входе **INTi** высокий уровень,  
по **INT1** **TMOD.7 =1**  
по **INT0** **TMOD.3 =1**

### Примеры.

1. Счетчик прерываний от таймера в P3.

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

```
intt0() interrupt 1 //счетчик времени
{ P3++; }
```

```
main()
{ TMOD=1; //режим 16-бит счетчик Tm0
  ET0=1; //маска прерывания Tm0
```



```

TR0=1    //разрешение счета
EA=1;    //разрешение прерываний

while(1); //ожидание прерываний
}

```

## 2. Измерение длительности интервала на входе INT0 по Gate0

### **intt0() interrupt 0**

```

{
P2=TH0;
P3=TL0;
TL0=0; TH0=0;
}

```

### **main()**

```

{ TMOD=09;    //режим 16-бит счетчик Tm0, Gate0
TR0=1    //разрешение счета
while(1); //ожидание прерываний
}

```

## **Задание 1.**

### **Часы.**

Таймеры по переполнению через постоянный интервал времени вызывают прерывания. Счетчик событий используется для расчета текущего времени с учетом длительности интервала в десятичной системе в секундах и минутах в портах P1,P2. Точность измерения не менее 0.1 сек на интервале 1 минута.

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

```

unsigned int code tt=0x10000-50000; //-50000 константа пересчета == 1/20 сек
unsigned char th,tl;

```

```

unsigned char sec,min;
unsigned char count=0;

```

```
intt0() interrupt 1 //счетчик
```

```

{
TL0=tl; //загрузить константу пересчета
TH0=th;
count++; счетчик переполнений
if(count==20) {sec++;count=0;}
}

```

```

if(sec==60){min++; sec=0;}
}
main()
{ tl=tt&0xff; //формирование констант для TH0 и TLO
th=tt>>8;
TMOD=1;
ET0=1;
TR0=1;
EA=1;
while(1)
{ P1=sec;
  P2=min; }
}

```

## Задание 2.

### Широтно-импульсная модуляция (ШИМ).

ШИМ применяется в технике для управления двигателями постоянного и переменного тока, управления питанием и в технике связи. Программируемая ШИМ может быть использована в этих приложениях как способ ввода-вывода информации в ЭВМ во встроенных микропроцессорных системах. ШИМ-способ прямого программного управления периферийными устройствами в ЭВМ.

Суть модуляции состоит в том, что информацию можно представить скважностью  $t1/(t1+t2)$  прямоугольных сигналов рис. 2.13.

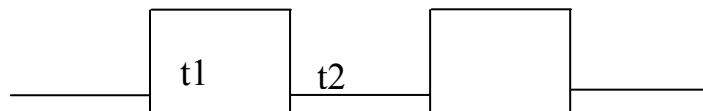


Рис.2.9. Диаграмма ШИМ.

При постоянном периоде  $T=t1+t2$  скважность  $Q(t/T)$  – функция от длительности положительного сигнала  $t$ . Используя масштабирование,  $Q$  можно привести к любому численному диапазону и интерпретировать его как напряжение, дискретное значение непрерывной функции, как битовую последовательность и др. Используя внешний фильтр (интегратор), можно выделить постоянную составляющую или низкочастотную непрерывную составляющую, например, кодируемый ШИМ синусоидальный сигнал. Для этого сигнала можно программировать частоту в ограниченном низкочастотном диапазоне

Ниже приведена схема ШИМ для формирования частоты.

Таймер 0 при переполнении переключает по прерыванию бит INT1 порта P3, формируя прямоугольный сигнал, и загружается константой пересчета.



цикле формирования периодического сигнала применяются операторы задержки **twatch(100)**.

Функции SIGNAL разрабатываются в С-подобном языке, в языке доступны все внешние порты MCU, которые могут быть обозначены виртуальными обозначениями (см. Help Keil).

Для выполнения сигнальных функций:

1. Перейти в режим Загрузки(Debug);
2. В окне COMMAND выполнить команду загрузки сигнального файла  
**>include fnt.inc** (файл fnt.inc должен быть в одном каталоге с объектным).
3. Запустить прикладную программу

### Пример файла **func.inc**.

// Функции формирования входного периодического прямоугольного сигнала в Бите P3.2- INT0)

```
SIGNAL void Signa (unsigned int Time) {
twatch (Time);
while (1) {
PORT3 = 0x3f;
twatch (Time);
PORT3 = 0;
twatch (Time);
} }
Signa(1000) //запуск функции
La PORT3 //вывод сигнала в Анализаторе
```

Сформировать сигнальной функцией на входе P3.2 прямоугольный сигнал и измерить его скважность. При этом методом **Захвата (Capture)** измеряется период и методом **Gate** – длительность положительного сигнала.

### Задание 4.

**Передать через порт двоичный код n=5,6,7,8,...бит в ШИМ**

вариант 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16  
бит 5 6 7 8 9 10 11 12 5 6 7 8 9 10 11 12

Внешний сигнал в ШИМ формируется **функцией SIGNAL** на входе INT1.

## 3. Программное управление вводом-выводом в ЭВМ.

### 3.1. Пульт ввода-вывода оператора.

Ручное управление и ввод, контроль состояния ЭВМ и объекта управления, оперативное программирование осуществляется с **локальных**

**пультов.** Типичный состав пульта – двузначные переключатели (Включение, Выключение, Режимы, ...), клавиатуры, светодиодные и ЖКИ-индикаторы.

### Ввод данных с переключателей и клавиатуры.

#### *Использование внутренних прерываний.*

Метод периодического сканирования клавиатуры с прерыванием при обнаружении нажатия клавиши. Останавливается сканирование и формируется код сканирования.

Каждой клавише ставится в соответствие единственный 8-битовый код. Код используется для идентификации клавиши.

#### Пример раскладки клавиатуры

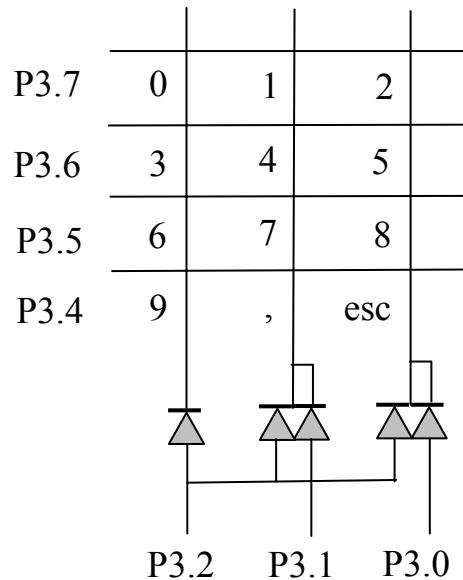


Рис. 3.1. Раскладка клавиатуры.

При сканировании, например, строки P3.7=0 и нажатой клавише 1 низкий уровень(L) через развязывающий диод поступает на вход прерывания INT0=P3.2 и на вход P3.1, где считывается и формируется значение кода сканирования. Все остальные биты порта P3 – единицы.

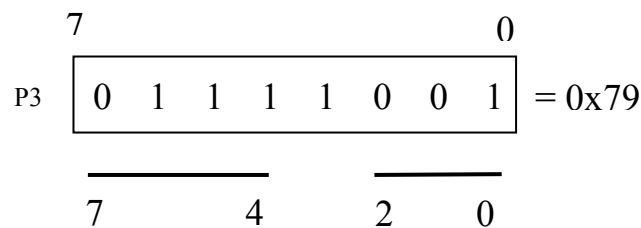


Рис. 3.2. Код сканирования.

Основная программа ввода числа с клавиатуры (фиксируется нажатие клавиши, идентифицируется клавиша и формируется текущее значение числа с переводом 10/2).

#### Алгоритм последовательного сканирования

- 1) Формирование кода сканирования **строка P3[7..4]=7.**
- 2) Контроль нажатия клавиши – **(установить код и сформировать прерывание).**
- 3) Идентифицировать замкнутый переключатель по коду 0x79.
- 4) Ожидание размыкания считыванием кода с порта P3, в младшей тетраде кода (должно быть 0x0F).

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

```
float numb; //целое число при вводе
```

```
char digit; //цифра, вводимая с клавиши
```

```
//предопределение функций – позволяет в любом порядке вводить их в конце  
//программы
```

```
char xdata mas[8]; //формирование строки символов из вводимых цифр
```

```
void Int00(void);
```

```
//ввод десятичного числа цифра за цифрой
```

```
    //идентификация цифры по коду клавиши
```

```
main ()
```

```
{
```

```
EX0=1; //маски прерываний
```

```
IT0=1; //тип прерывания – по фронту
```

```
EA=1; //разрешение прерываний
```

```
while (1); //ожидание прерываний
```

```
}
```

```
void Int00(void) interrupt 0 //фиксировать нажатие клавиши по прерыванию  
INT0
```

```
{
```

```
switch (P3)
```

```
{ case 0x01: digit='0'; break;
```

```
  case 0xfe: digit='1'; break;
```

```
  case 0x02: digit='2'; break;
```

```
  case 0xfd: digit='3'; break;
```

```
  case 0x79: digit='4'; break;
```

```
  case 0xfb: digit='5'; break;
```

```
  case 0x08: digit='6'; break;
```

```

        case 0xf7: digit=7; break;
        case 0x10: digit=8; break;
        case 0xef: digit=9; break;
        case 0xef: digit=','; break;
        case 0xef: digit='e'; break;
        default: digit=0xff;
    }
    if(digit==' ') {mm=m; mas[i++]=digit; digit=0xff}
    if(digit=='e') //завершение ввода числа
        {i=0; digit=0xff;
        numb/=mm; } //масштабирование с учетом запятой
    else {numb=numb*10 +digit&0xf; m*=10;}
    if (digit!=0xff)
        { mas[i++]=digit; //сохранить цифру
        m*=10; //масштаб числа
        numb=numb*10+(digit&0x0f); // перевод 10/2
    }
    while(~INT0 );
}
}

```

**Задание.**

Построить таблицу кодов сканирования.

Разработать программу ввода чисел для заданной раскладки.

Варианты раскладки клавиатуры

- 1)     **1 2 3 4**  
        **5 6 7 8**  
        **9 0 , esc**
  
- 2)     **1 2 3 4 5**  
        **6 7 8 9 0**  
        **- ,    esc**
  
- 3)     **1 2 3**  
        **4 5 6**  
        **7 8 9**  
        **0 , esc**
  
- 4)     **← ↑ →**  
        **F1 ↓ F2**  
        **, -- esc**

### 3.2. Работа ADC преобразователя ЭВМ SAB515/535.

Структура блока измерения с учетом встроенных в SAB515 схем управления аналого-цифровым преобразованием включает следующие программно-доступные регистры:

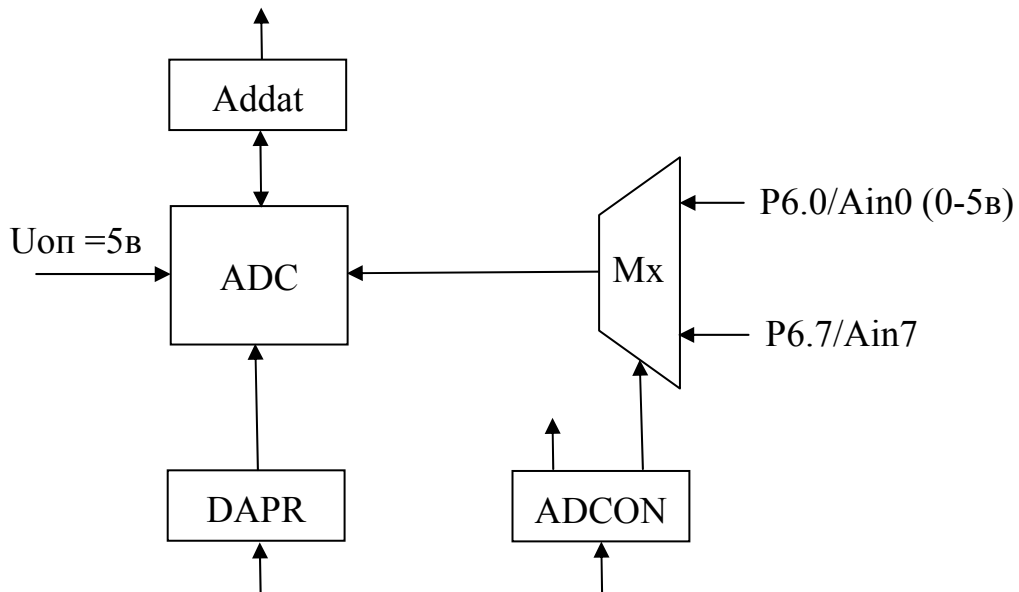


Рис. 3.3. Программируемая схема ADC.

1) 8 адресуемых аналоговых входа Ain0-7 порта P6.

Входной аналоговый сигнал подключается к одному из входов

2) регистр команд ADCON= b.clk.-.bsy.adm.mx[2.0]

**Bsy=1** – признак преобразования в ADC.

**Adm= 0/1** – одиночное/многократное преобразование.

**Mx** - адрес входного канала.

3) регистр масштаба DAPR=0 , опорное напряжение 5 в и диапазон входного сигнала 0-5 в.

4) результат преобразования - регистр ADDAT.

5) 8-разрядный ADC-преобразователь  $S(U_x)$  последовательного приближения. Завершение преобразования – по сигналу готовности **bsy=0**, по прерыванию или при одиночном преобразовании – по времени (задержка ~ 20 мкс).

Значение аналогового сигнала

$$U = U_{оп} / 2^8 * S,$$

где  $U_{оп} / 2^8$  – цена бита для 8-разрядного преобразования;

S-двоичный код – результат преобразования;

$U_{оп}$  – максимальное значение сигнала на входе;

$U = U_{оп}$  для  $S = 2^8 = 0xFF$ .

Программа чтения периодического аналогового сигнала с сохранением и обработкой в MCU:



```

#include <reg515.h> //в опциях Project выбрать микросхему
SAB8C515 фирмы
// Infenion
delay(char t) //задержка преобразования
{ while(t--); }

main()
{ char i;

while(1)
for(i=0; i<100; i++)
{ DAPR=0; //запуск преобразования с опорным напряжением
5в
delay(20); //задержка для завершения преобразования
P3=ADDAT; //чтение результата и подтверждение его в P3
для Анализатора
}
}

```

Сигнальная функция для работы с Логическим анализатором (файл adc.inc)

```

SIGNAL void Signa (unsigned int Time) { //Сигнальная функция
float x;

twatch (Time);
while (1) {

//sin(x) –аналоговая float величина , с учетом опорного (и максимального по
умолчанию) напряжения 0xff~5в
AIN0 =__sin(x)*2 + 2,0; //масштабирование и смещение в положительной
области на 2 вольта
twatch (Time); //задержка для последовательного исполнения
симулятором двух программ
}
}
}
Signa(1000) //запуск сигнальной функции -
LA AIN0 // контроль аналогового сигнала на входе
LA PORT3 //чтение целой величины –результата преобразования через порт
P3 из программы измерения.

```

**Include adc.inc - командный режим**

**Signal kill signa – командный режим****Задания.**

– измерить частоту, амплитуду и смещение аналогового сигнала по результатам измерения ADC.

```
#include <reg515.h>
```

```
    unsigned int volt,tm; //значение в мВ
    unsigned char max=0,min=0;
```

```
delay(char t)
{ while(t--); }
```

```
    Adc() //функция преобразования
    {
        DAPR=0; //запуск преобразования
        delay(20); //задержка для завершения преобразования
        P3=ADDAT; //чтение результата и подтверждение его в P3 для
Анализатора
    }
```

```
main()
{ char i;
  TMOD=1;
  ADCON=0; //выбирается вход 0 и режим одиночного преобразования
//найти max и min
  while(INT0) // ожидание завершения = нажатие клавиши INT0
      {
          adc();
          max= (P3>max)? P3 : max;
          min= (P3<min)? P3 : min;
      }
//измерить период : начало измерения – ожидание max
  while(P3<max) adc();
```

```
    {TH0=0;TL0=0;TR0=1;} //запуск таймера по max для измерения 1/2
периода
```

```
    while(P3>min) adc(); //ожидание min
        TR0=0;
        tm=(TH0<<8) + TL0; //захват по min
```

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

### 3.3. Последовательный интерфейс.

Последовательные каналы (интерфейсы) с минимальным числом линий связи (2-3) широко применяются для подключения к ЭВМ программируемого удаленного оборудования.

Интерфейсы различаются как синхронные и асинхронные.

**1. Синхронные** Интерфейсы для внутрисхемного (внутриплатного) применения в режиме Ведущий(Master)-Ведомый(Slave). В явном виде используется линия жесткой **побитовой синхронизации** и дуплексная (двунаправленная) линия данных. Синхросигнал формирует Ведущий.

**2. Асинхронный интерфейс** используется как внешний, так и внутренний – межмодульный и мультимашинный.

Для связи с удаленным периферийным оборудованием или HOST-машиной, требуются помехоустойчивость, согласование уровней и электрическая развязка по питанию. Согласование обеспечивается специальными схемами преобразования сигналов в **стандарте RS232** (1962 г EIA) – физический интерфейс NRZ (точка-точка), расстояние до 15 м, скорость до 19.2 Кбод (в настоящее время- 115 Кбод). Уровни сигналов от -12 до -5в – значение единицы и от +5 до +12в – значение нуля, минимальное число линий 3 (Tx,Rx,Gnd).

**Асинхронный протокол** обмена данными по внешним каналам передачи данных 8-9 бит включает 8 бит двоичного кода данных и, в частном случае, бит контроля четности, Для выбора байта в последовательности битов используется *старт-бит* низкого уровня и завершающий *стоп-бит* высокого уровня.

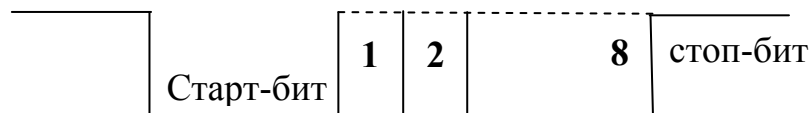


Рис. 3.4. Протокол передачи байта в RS232.

**Синхронизация битов** обеспечивается:

- многократным 1:16 и более сканированием бита,
- одинаковой скоростью передачи на стороне датчика и приема на стороне приемника.

При этом допускается некоторое рассогласование синхронизации по фазе, что и характеризует такой интерфейс как асинхронный.

RS232 (COM-порт) используется в качестве внешнего интерфейса ПЭВМ для подключения периферии.

В USART **MCS51** совмещены синхронный и асинхронный интерфейсы, предусматриваются 4 режима:

**0** – **синхронный** режим для межсхемного использования полудуплексный двухпроводный с двумя линиями – двунаправленная линия данных и однонаправленная линия синхронизации, скорость передачи 1 Мбод.

**1** – **асинхронный 8-битовый** дуплексный (линия Td- передачи и линия Rd-приема) с программируемой таймером Tm1 скоростью обмена (стандартный ряд скоростей – 480 бод, 960, ..4800, 9600, 19.200, – до 115 кГц). 8-разрядный таймер позволяет получить необходимую скорость до 19.2 Кбод с точностью не менее 3% при частоте 11.059 МГц.

**Завершение обмена** контролируется битами готовности TI (завершение передачи), RI (завершение приема) или по прерыванию при установке этих битов.

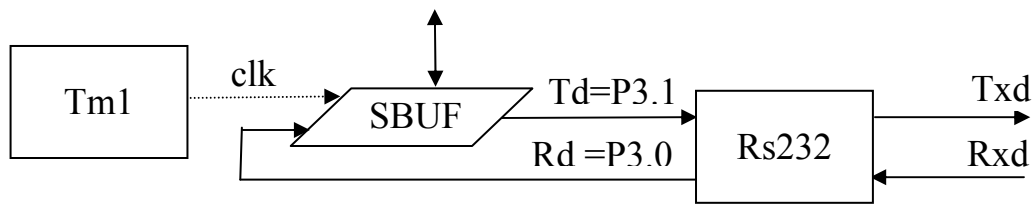


Рис.3.5. Схема работы модуля UART с RS232.

Модуль содержит регистр данных **SBUF** и управляющий регистр **SCON= SM[0.1].SM2.REN.TB8.RB8.Ti.Ri**

Линии передачи Td и приема Rd могут быть использованы для соединения модулей в мультимашинной системе. Для организации внешнего интерфейса в стандарте Rs232 используется внешняя схема преобразования уровней.

При **инициализации USART** в регистре **SCON** разрешается чтение **REN=1**, выбирается режим **SM[0-1]={0 – синхронный(S), 1- асинхронный(A), 3-асинхронный мультимашинный}**.

**Скорость** передачи задается таймером Tm1 (режим 2 – меандр, 8 – разрядный счетчик, частота задается константой в TH1).

Передача начинается записью байта в регистр **SBUF=x**, биты кода передаются в **линию Txd** младшими разрядами вперед, завершается передача установкой признака **Ti=1**;

Если разрешен прием (**REN=1**), то поступающая на **вход Rxd** последовательность бит сдвигается в регистр **SBUF** и устанавливается признак завершения приема **Ri=1**.

**Типовая настройка** – скорость обмена 9600 бод, 8-бит формат, 1 – стоп бит, нет бита контроля четности, управление обменом – программное, режим асинхронный **SM=01**.

Стандартные функции из библиотеки **studio.h** языка C51 обмениваются данными через USART.

Функция **char getkey()** – ввести символ

Функция **printf**("text %d\n", x) – форматированный вывод в последовательный канал, результат отображается в окне **Serial** системы Keil.

### Ввод по прерыванию

```
#include <reg51.h>
#include <stdio.h> //библиотека ввода-вывода C51

Char i,aa[5];
Int x;

Seria() interrupt 4
{
    s=getkey();
    aa[i++]=s;
    x=x*10+(s&0x0f); //двоичное число
    If (i==4) { printf("x= %s3d\n", x); // форматированный вывод в USART
                I=0;
            }
}

Main()
{
    SCON=0x50; //режим асинхронный 8 бит, gen=1
    TMOD=0x20; //таймер 1 – режим 2
    TH1=0xfd; //константа автозагрузки – частота 9600 бод
    TR1=1;
    ES=1; //маска прерывания
    EA=1;
    TI=1; //начальная установка готовности передачи
    While(1);
}
```

**Прикладная программа** выбирает режим и разрешает обмен.

**Сигнальная функция** формируют внешнюю передачу через последовательный канал.

В сигнальной функции используется переменная **SIN** для обозначения входа RxD последовательного канала

SIN=0x55;

По умолчанию также предусматривается вывод значения оператором **printf(..)** в окне SERIAL, выбираемого меню VIEW.

Ниже приведен файл serial.inc **ввода числа**

```

signal void serial(void){
char s;
for(s=0; s<5;s++){
  twatch(1000);
  SIN=s+0x30; //ввод через usart ASCII-кода цифры
  }
}

```

**serial()** //исполнение функции

Функция serial() формирует символ на входе RxD, в окне Watch отображается введенный текст.

**Задание:**

- организовать терминальный режим управления с клавиатуры;
- ввести десятичное число с естественной запятой и преобразовать в двоичное с плавающей точкой
- результат вывести в окно дисплея.

Если использовать оператор ASSIGN COM1, то обмен осуществляется через физический порт COM1 PC – можно связаться с внешней программой терминала.

**Литература.**

1. Help в Keil (C51, Макроассемблер, Система команд MCS51).
2. Сташин В.В. Урусов А.В. Мологонцева О.Ф. Проектирование цифровых устройств на однокристалльных микроконтроллерах, М: Энергоатомиздат, 1990.

## Система команд MCS51 – мнемокоды.

## Арифметика и логика

add a, {ri, @rj, #d, ad}  $a \leftarrow a + \{ \dots \}$ , призн с, v, p  
 addc a, { ..... }  $a \leftarrow a + \{ \dots \} + c$ , ....  
 subb a, { ..... }  $a \leftarrow a - \{ \dots \} - c$ , ....  
 inc {ri, @rj, ad, dptr, a} { ..... } + 1  
 dec {ri, @rj, ad, a}  
 mul ab  $ba \leftarrow a * b$  v=(a\*b>255) 0→с, p  
 div ab  $a \leftarrow a / b$ , b←rest (b==0)→ov, 0→с  
 anl a, {ri, @rj, #d, ad}  $a \& \{ \dots \} \rightarrow a$  0→с, p  
 anl ad, {#d, a}  
 orl a, {ri, @rj, #d, ad}  $a \vee \{ \dots \} \rightarrow a$   
 orl ad, {#d, a}  
 xrl a, {ri, @rj, #d, ad}  
 xrl ad, {#d, a}  
 clr a  
 cpl a не(a)  
 rl a rol(a) p  
 rlc a rolc(a), с, p  
 rr a ror(a) p  
 rrc a rorc(a), с, p  
 da a коррекция

## Пересылки

mov a, {ri, @rj, #d, ad}  $a \leftarrow \{ \dots \}$   
 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  $a \leftarrow \text{Code}(dptr+a)$   
 movc a, @a+pc  $a \leftarrow \text{Code}(pc+a)$   
 movx a, { @rj, @dptr }  $a \leftarrow \text{xram} \{ \dots \}$   
 movx { @rj, @dptr }, a  $\text{xram} \{ \dots \} \leftarrow a$   
 push ad  $\text{Data}(+sp) \leftarrow \text{Data}(ad)$   
 pop ad  $\text{Data}(sp-) \rightarrow \text{Data}(ad)$   
 xch a, {ri, @rj, ad}  $a \leftrightarrow \{ \dots \}$   
 xchd a, @rj  $a(3-0) \leftrightarrow @rj(3-0)$   
 swap a  $a(3-0) \leftrightarrow a(7-4)$



## Команды булевого 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
sjmp rel      PC+rel
jmp @a+dptr   PC←a+dptr
jz rel        PC+rel,если (a=0)
jnz rel       .... ,если (a<>0)
jc rel        ....
jnc rel       ....
jb bit,rel    .... ,если bit=1
jnb bit,rel   .... ,если bit=0
jbc bit,rel   ... ,если bit=1,bit<-0
djnz {ri,ad},rel  {}-1;pc+rel,если {}<>0
cjne {ri,@rj},#d,rel rel,если {}<>#d
lcall a16     стек←pc, PC←a16
acall a11     ...., PC(10-0)←a11
ret           PC←стек
reti         PC←стек,tf←0
nop          пропуск

```

	Обозначения битов SFR								Адрес
	7	6	5	4	3	2	1	0	
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	tel 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

$r_i = \{r_0, r_1, \dots, r_7\}$   $r_j = \{r_0, r_1\}$  **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 специального регистра

bit - адрес бита в поле битов 00-7f или в специальном регистра-

адрес образуется из собственного адреса регистра, к которому добавляется номер бита;

,разряд специального регистра асс.5, psw.0, ... ,(80i - адреса битов 80,...87 регистра 80)

обозначение бита smod,sm0,....

/bit - инверсия бита

rel - относительный адрес <метка> в доп. коде

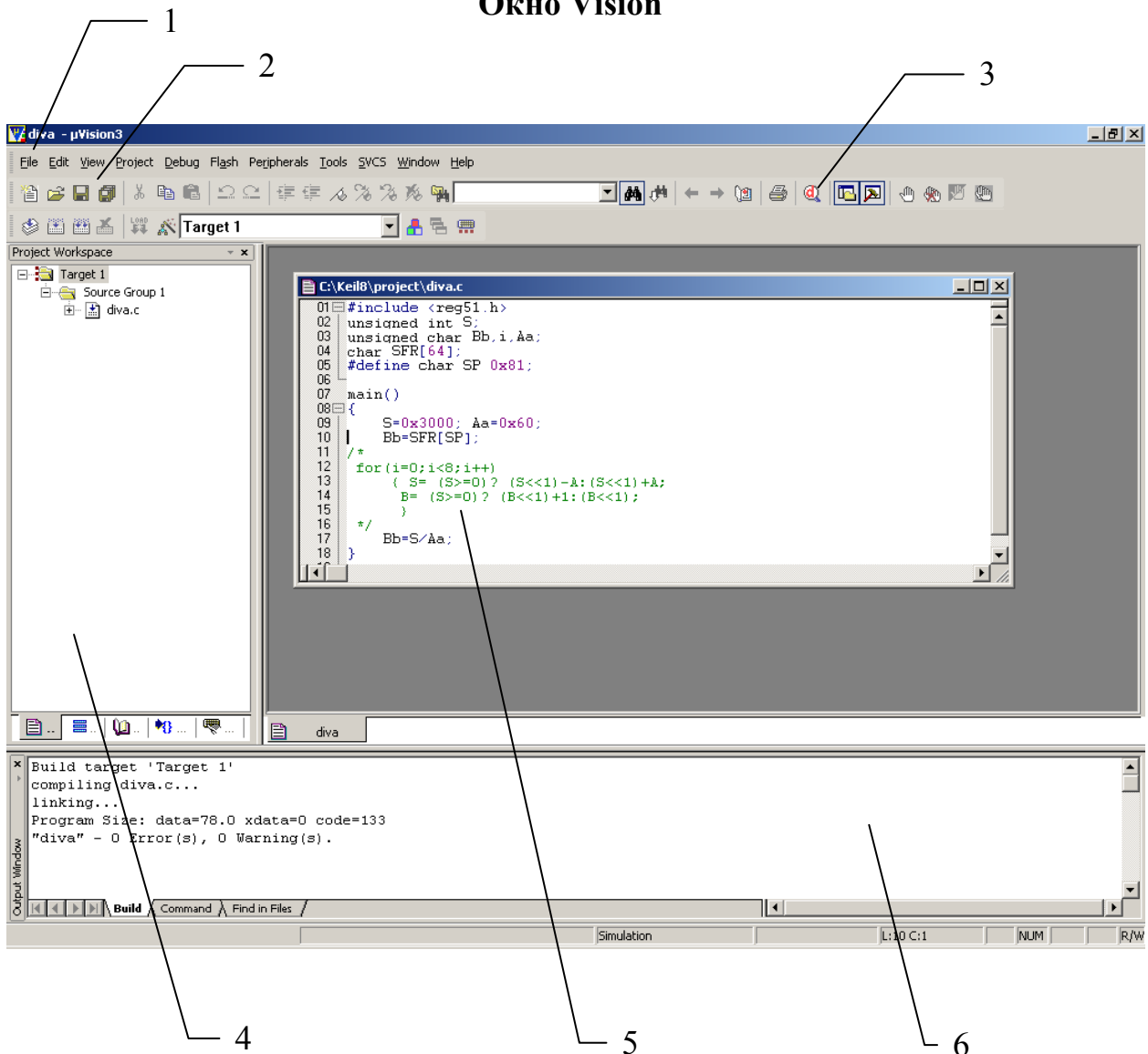
## Интегрированная система программирования и отладки Keil.

### Назначение Интегрированной среды IDE:

- Программирование задач на языке Ассемблера MCS51, C51.
- Создание проекта для работы с программой на разных этапах.
- Синтаксический контроль.
- Компиляция программы в объектный код (HEX-файл и LIST-листинг).
- Загрузка и симуляция выполнения программы с контролем состояния памяти и периферии.

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.

### Стандартное меню **Vision**

**File Edit View Project Debug Flash Peritherial Tools SVCS  
Window Help**

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

#### **File**

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

**Edit** - имеют стандартное назначение

#### **Project**

**New** → **µVision project**  
**Import**  
**Open**  
**Close**

**Manage** → компоненты, окрестности  
**Select Device** → библиотека элементов  
**Options** → настройки параметров компиляции и загрузки

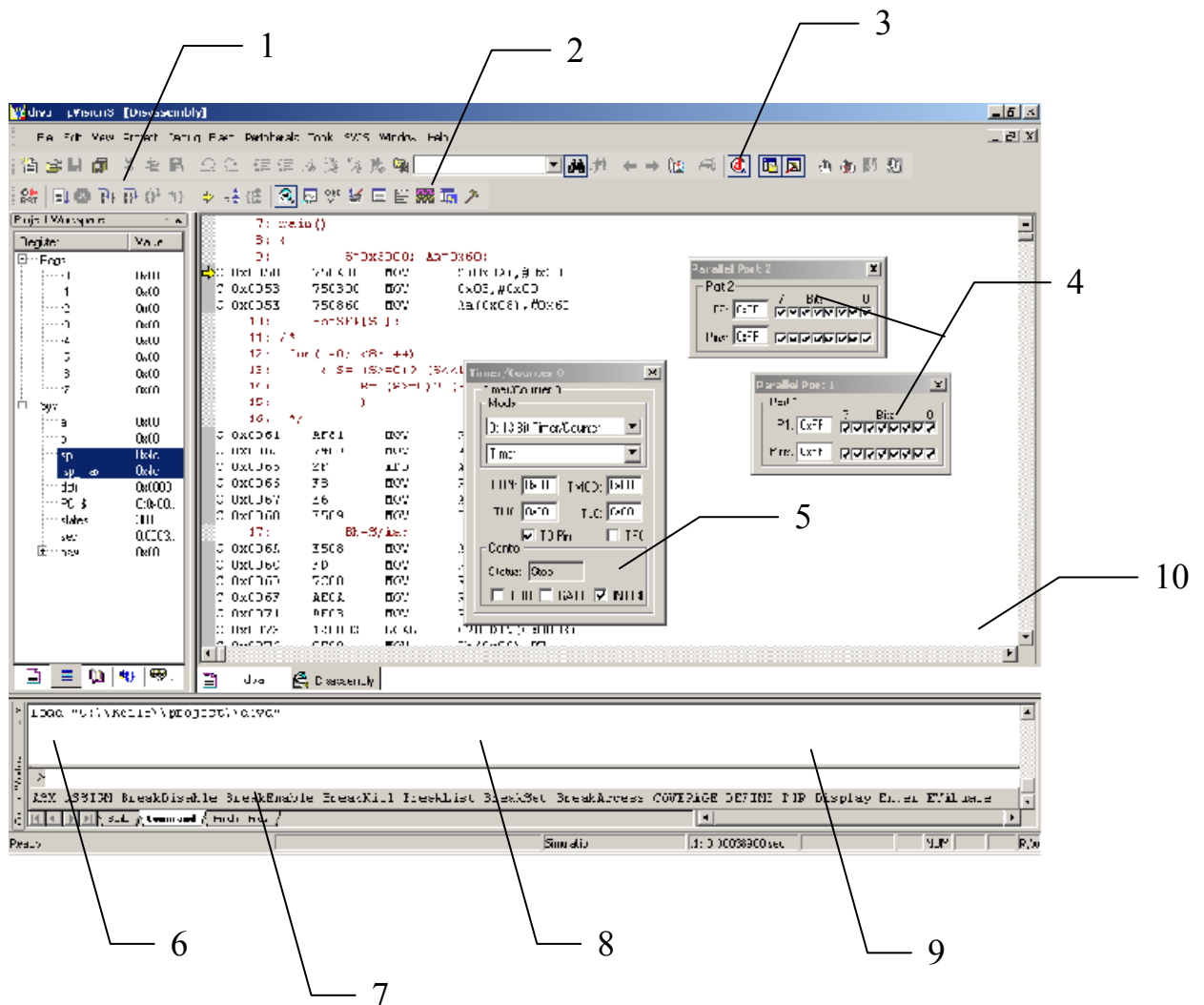
- Device** – выбор модуля
- Target** - выбор частоты MCU
- Out** - вывод HEX-кода
- List** – вывод листинга .lst
- C51** – размещение таблицы векторов
- L51** – размещение программы Code
- размещение данных в памяти Xdata

**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.
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.

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

1. Программная модель MCS51 в C51.
2. Программная модель MCS51 в Ассемблере.
3. Структура памяти – адресация.
4. Арифметические и логические операции.
5. Команды управления программой.
6. Преобразование 2/10, 10/2 при вводе-выводе через порты.
7. Символьные преобразования 10/16.
8. Символьные преобразования 16/10.
9. Программа умножения в C51.
10. Программа деления в C51.
11. Вычисление функций с дробными числами – масштабирование
12. Макроассемблирование, применение.
13. Битовые данные.
14. Система прерывания MCS51.
15. Внешние прерывания, применение.
16. Внутренние прерывания от таймеров, применение.
17. Методы измерения временных параметров Захватом и в режиме Gate
18. Широтно-импульсная модуляция.
19. Измерения реального времени – часы.
20. Ввод данных с клавиатуры.
21. Принцип работы ADC SAB515.
22. Последовательный интерфейс UART MCS51.
23. Структура и возможности системы Кейл.



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

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, мультимедиа-технологии, вычислительные сети и средства телеком-

муникации, 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. Для 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 в 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.