**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,**

**МЕХАНИКИ И ОПТИКИ**

Кафедра вычислительной техники

КУРСОВАЯ РАБОТА

по дисциплине

«Организация ЭВМ и систем»

на тему: «Проектирование ЭВМ»

Вариант 11

Выполнил:

студент группы 3125

Мячков Эдуард

Проверил:

к.п.н., доцент каф. ИТГС

Тропченко А.А.

Санкт-Петербург

**Целью** курсового проектаявляется разработка микропрограммного управления и схемы ЭВМ с архитектурой и системой команд MCS51.

В рамках курсового проекта требуется:

1. привести обзор микроархитектуры на уровне Ассемблера
2. привести задание и спецификацию (описание) команд задания из Keйл.Help, включая кодирование и принцип исполнения
3. разработать функциональные микропрограммы для команд в системе Микро51 на языке Borland C++, используя общую структурную схему ЭВМ
4. разработать тест для заданной системы команд и выполнить моделирование в Микро51
5. разработать структурную схему для выполнения одной из команд тестовой программы:
   * реализовать ее в MaxPlus2 в Графическом редакторе, скомпилировать
   * привести текстовое описание выполнения команды по схеме
   * моделировать и оценить параметры схемы (сложность, процент затрат на реализацию всех типов памяти задержки, максимальная частота)
   * В Микро51 дополнить микропрограмму средствами кодирования микрокоманд и формирования файлов загрузки ПЗУ
6. реализовать структурную схему в MaxPlus2 в Текстовом редакторе на языке AHDL, скомпилировать, моделировать и оценить параметры схемы (сложность, процент затрат на реализацию всех типов памяти задержки, максимальная частота)
7. **Программная модель (микроархитектура) на уровне Ассемблера.**

Диаграмма как изображение микроархитектуры обозначает программно-доступные на уровне системы команд (Ассемблера) ресурсы компьютера.

Рисунок1

Рис. 1.1 Программная модель

* 1. *Структура памяти, команды обмена данными*

Интегрированная в микросхеме память имеет иерархическую организацию, в которой уровни памяти различаются типами хранимых данных, режимами адресации , назначением, объемом и быстродействием.

1. Основные регистры:

**а(Асс)** – основной регистр-аккумулятор, применяемый во всех арифметических и логических операциях с неявным (безадресным) доступом.

**B** - рабочий регистр ALU

**PSW=C.AC.F0.RS1.RS0.OV.-.P -** регистр состояния, которыйсодержитпризнаки результата арифметических операций: -С(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

1. Память Ram – 256 байт разделена на два блока Data и SFR

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

К SFR относятся указатель стека SP, таймеры *ТH0,ТL0, ТH1,ТL1*, теневые регистры *ACC, B, PSW, DPTR=DPH.DPL*, регистры портов *P0,P1,P2,P3.*

* *Оперативная память данных Data* – структура иерархическая по назначению и доступу.

1. **Ri={ R0,R1,..R7 }** - активный банк регистров общего назначения 8 байт, доступны 4 банка, совмещенные с начальными ячейками памяти Data , активный банк выбирается в регистре PSW. Регистры Ri имеют короткие адреса, что позволяет их разместить в первом байте кода команды

**mov a,R0** ; Data(R0) Асс

**mov R1,a** ; Асс  Data(R1)

1. **Bit** - 128 бит, прямой адрес бита 0-7fH, память совмещена с ячейками 20-2f Data, еще 128 бит c с адресами 80h-ffh относятся к SFR

**mov c, 0 ;** Data(20h.0) С , 20h.0 – нулевой бит ячейки Data

**mov ACC.7, c ; c  Acc.7,**

**mov c, x0 ; x0-** имя бита

1. **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 Data(22h)Idata(+SP)**,** SFR(P1)Idata(+SP)

**pop ad**

Например**, pop Acc**  обозначает Data(SP--)  Acc**,**  Idata(SP--)  SFR(Acc)

При включении и сбросе MCU устанавливается SP=07. При переполнении стека следующий адрес вершины SP=0.

1. Постоянная память программ и констант Code 64 кб адресное пространство,

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

**movc a,@a+pc ;** Code(PC + Acc)  Acc **;** адресация относительно текущего РС**,** в АСС индекс

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

1. Расширенная память данных Xdata – запись и чтение данных при исполнении программ. Объем адресного пространства 64 Кбайта,

**movx a, @dptr**, Xdata(dptr)  Асс

**movx @dptr,a**

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

* 1. *Арифметические операции*

1. Знаковая арифметика:

**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 Р2

1. 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

1. Десятичная арифметика:

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

* 1. *Логические поразрядные операции:*

**anl a, {Ri,@rj,#d,ad}** a & {..} a признаки p, 0c,

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

пример **anl P3,#0f0h** операция **чтения-модификации-записи** регистра P3

**orl a,** {Ri,@rj,#d,ad} a v {..} a признаки p, 0c,

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

**xrl** {Ri,@rj,#d,ad} a # {..} a признаки p, 0c

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

**pl a ; not a**

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

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

* 1. *Битовые логические операции:*

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

Например, **anl c,/ACC.6**

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

**setb bit,**

**clr bit,**

**cpl C**

* 1. *Команды управления программой*:

К ним относятся команды ветвления, формирущие 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 АСС (=0)/(!=0)

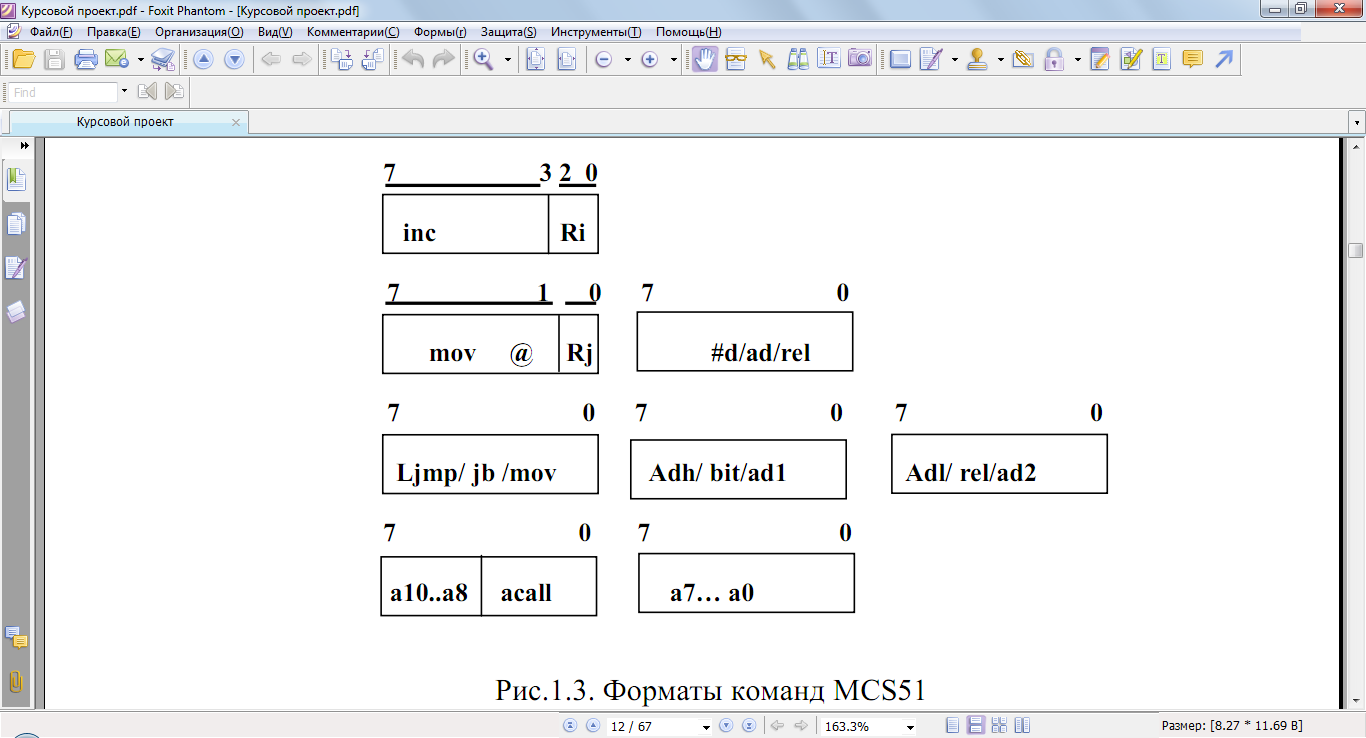
**jb/jnb bit, метка ;**

примерjb 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;

* 1. *Форматы команд* – однобайтовые, двухбайтовые и трехбайтовые (форматы и кодирование команд подробнее - в Keil/Help):



1. **Описание заданных команд ЭВМ**

|  |
| --- |
| **ANL (логическое И) direct, A** |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | C | .AC | .F0 | .RS1 | .RS0 | .OV.-. |  | P | |

|  |  |
| --- | --- |
| Bytes | 2 |
| Cycles | 1 |
| Encoding | |  |  |  | | --- | --- | --- | | 01010010 |  | direct | |
| Operation | ANL  (direct) = (direct) AND A |
| Example | ANL 40h, A |

|  |
| --- |
| **ANL direct, #immediate** |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | C | .AC | .F0 | .RS1 | .RS0 | .OV.-. |  | P | |

|  |  |
| --- | --- |
| Bytes | 3 |
| Cycles | 2 |
| Encoding | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 01010011 |  | direct |  | immediate | |
| Operation | ANL  (direct) = (direct) AND immediate |
| Example | ANL 30h, #77h |

|  |
| --- |
| **RRC (****сдвиг содержимого аккумулятора вправо через флаг переноса) A** |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | C | .AC | .F0 | .RS1. | RS0. | OV.-. |  | P | |

|  |  |
| --- | --- |
| Bytes | 1 |
| Cycles | 1 |
| Encoding | |  | | --- | | 00010011 | |
| Operation | RRC  An = An+1 where n = 0 to 6  A7 = C  C = A0 |
| Example | RRC A |

|  |
| --- |
| **MOV direct, Rn** |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | C | .AC | .F0 | .RS1. | RS0. | OV.-. |  | P | |

|  |  |
| --- | --- |
| Bytes | 2 |
| Cycles | 2 |
| Encoding | |  |  |  | | --- | --- | --- | | 10001nnn |  | direct | |
| Operation | MOV  (direct) = Rn |
| Example | MOV P2, R5 |

|  |
| --- |
| **MOV direct, #immediate** |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | C | .AC | .F0 | .RS1 | .RS0 | .OV.-. |  | P | |

|  |  |
| --- | --- |
| Bytes | 3 |
| Cycles | 2 |
| Encoding | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 01110101 |  | direct |  | immediate | |
| Operation | MOV  (direct) = immediate |
| Example | MOV P2, #0FFh |

|  |
| --- |
| **MOV direct, A** |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | C | .AC. | F0. | RS1. | RS0. | OV.-. |  | P | |

|  |  |
| --- | --- |
| Bytes | 2 |
| Cycles | 1 |
| Encoding | |  |  |  | | --- | --- | --- | | 11110101 |  | direct | |
| Operation | MOV  (direct) = A |
| Example | MOV P0, A |

|  |
| --- |
| **DJNZ direct, offset** |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | C | .AC | .F0 | .RS1 | .RS0 | .OV.-. |  | P | |

|  |  |
| --- | --- |
| Bytes | 3 |
| Cycles | 2 |
| Encoding | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 11010101 |  | direct |  | offset | |
| Operation | DJNZ  PC = PC + 2  (direct) = (direct) - 1  IF (direct) <> 0  PC = PC + offset |
| Example | DJNZ 40h, LABEL |

1. **Структурная схема и общее описание её работы**

Построение структурной (блок) схемы – первый этап в проектировании схемы ЭВМ на основе программной модели.

Следующие принципы учтены при выборе структуры:

**1.** Иерархический подход к проектированию схем, который поддерживается в MaxPlus. На первом этапе выполняется функциональное неформальное разбиение схемы на функциональные блоки с учетом распределения памяти по блокам и функциональным элементам, определяемым в программной модели.

**2.** Используется шинная организация соединений, достоинствами которой являются:

− максимально параллельное исполнение разнообразных передач между регистрами, регистрами и блоками иерархической памяти и выполнение элементарных операций в АЛУ;

− Используется регулярная схема управления, в которой применяется адресация при выборе регистров и определении функций записи и чтения, вместо одиночных управляющих сигналов. При этом можно ожидать более простую схему кодирования и декодирования микрокоманд.

**3.** Применяются, по возможности, простые регистры-защелки для хранения выбранных из памяти данных и промежуточных результатов. Операции счета и сдвига могут быть выполнены комбинационными схемами

при передаче данных между регистрами.

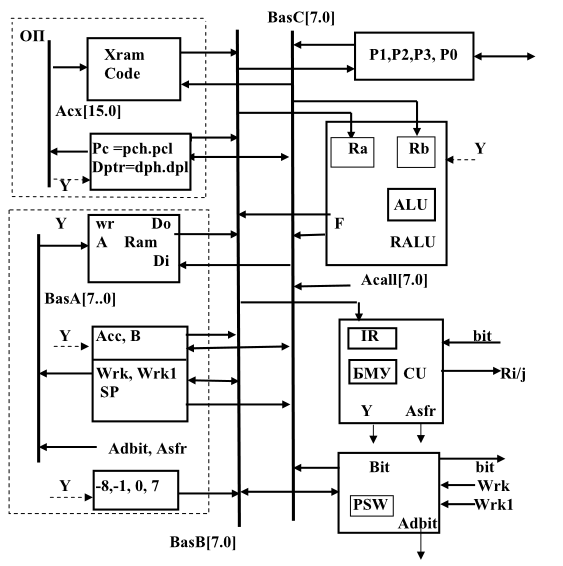
**4.** Для максимально параллельного выполнения операций счета и сдвига используются накапливающие синхронизированные регистры-счетчики и сдвигатели. Выполнение этих микроопераций совмещается с передачами между регистрами и памятью, в которых активно используются шины.

**5.** К регистрам может быть обеспечен как регулярный адресный доступ через мультиплексоры, так и непосредственный для контроля и работы с отдельными битами и полями битов.

**6.** Неявно используемые регистры SFR для сокращения обращения к памяти дублируются с использованием их теневого отображения в памяти и непосредственного доступа в схеме при чтении.

Таким образом, сначала выбирается блочная структура памяти с учетом ее организации в программной модели (микроархитектуре). В один блок объединяются элементы памяти с одинаковыми интерфейсами. Признаками интерфейса являются – способ доступа (адресный – задаваемый режимами адресации в командах, адресный – через мультиплексоры, прямое обращение к регистрам), форматы и типы данных (слова, байты, биты).

**22**



В схеме представлены функциональные устройства, специальные блоки для преобразования битовых данных и формирования констант.

Шины данных и адреса = Acx[15.0], BasA[7.0], BasB[7.0], BasC[7.0].

Блок основной памяти ОП. Устройство управления CU. Регистровое арифметико-логическое устройство РАЛУ. Y – сигналы управления блоками и устройствами ЭВМ формируются в CU.

1. **Блок основной памяти (ОП)** включает:

1) Постоянную программную память **Code**(ограничиваемся 256 байтами).

2) Память данных **Xdata** (256 байт).

3) В ОП используется общая адресная 16-битовая шина **ACX**[15.0], слово памяти – байт.

4) **16-битовые адресные регистры-**счетчики с прямым и адресным доступом (**DPTR, PC**), которые формируют 16-битовый адрес на адресной шине ACX[15.0].

2. **Блок внутренней быстрой памяти**:

1) **RAM**(256 байт) объединяет **Data** и **SFR** с общей 8-битовой шиной адреса **BasA[7..0]** и 8-разрядным словом данных.

2) **8-битовые арифметические регистры с прямым и адресным доступом (ACC, B),** используемые в арифметических и логических операциях, регистры имеют теневое отображение в SFR.

3) **Адресные и рабочие регистры (SP, Wrk, Wrk1)** – формируют 8-битовый адрес, хранят операнды и параметры команды, являются счетчиками циклов в операциях умножения и деления и сдвигателями. SP имеет теневое отображение в SFR.

3. **Регистровое арифметико-логическое устройство (RALU)** включает арифметико-логическое устройство (**АЛУ**), регистры временного хранения операндов RA, RB.

4. **Устройство управления (CU)** содержит регистр команд IR, блок микропрограммного управления (**БМУ**) с декодером микрокоманд Y.

5**. Блок двунаправленных портов ввода-вывода (P0,P1,P2,P3)** связан с внешними контактами микросхемы, содержит одноименные регистры с прямым доступом и с теневым отображением в SFR.

6. **Блок формирования констант (0, -1, -8, 7)**.

7. **Блок выборки** и выполнения битовых операций **BIT**. Блок подключается к рабочим регистрам Wrk,Wrk1 и содержит регистр **PSW**. В блоке формируется адрес доступа к битам в Adbit, значение бита BIT для условных микрокоманд в CU.

Для соединения модулей памяти, регистров и других функциональных элементов используются шины, мультиплексоры и селекторы.

**Мультиплексирование** – физическое подключение элементов (в пространстве) к общей шине, включая последовательный во времени адресный выбор и подключение элементов к шине и запись с шины.

Каждый мультиплексор входных данных **BasB, BasC, BasA, ACX** позволяет прочитать по адресу данные только из одного источника (регистра, памяти).

Запись с мультиплексированных шин в регистры и память выбирается адресным **декодером (селектором) WrB** – с шины **BasB**, **декодером WrC** – c шины **BasC**. Сигнал записи обозначается единицей на одном из **2n** выходов декодера, где **n**-разрядность адреса выбираемого функционального элемента на шинах BasB и/или BasC. При этом нуль на всех остальных выходах декодера обозначает параллельное чтение, но выбор одного из читаемых значений осуществляется мультиплексором шины.

1. **Листинг эмулирующей работу МК программы с функциональными микропрограммами.**

#region vars

static byte[] Ram = new byte[256];

static byte[] Xdata = new byte[64];

static byte[] Code = new byte[64];

static byte IR, //регистр команд

Wrk1, //рабочий регистр

Wrk, //рабочий регистр

ACC, //аккумулятор с прямым доступом

SP, //указатель стека

RA, //рабочий в RALU

RB,

PSW, //слово состояния C.AC.F0.RS1.RS0.OV.-.P

B, //рабочий

P3, //регистр порта Р3

TCON, //tf1 tr1 tf0 tr0 ie1 it1 ie0 it0

IE, //EA . . es et1 ex1 et0 ex0

IE1, //буфер регистра IE

int0, //бит запроса прерывания 0

int1, //бит запроса прерывания 1

icod; //регистр вектора прерывания;

static uint PC = 0; //PCH=PC>>8,PCL=PC

//адреса теневых регистров в SFR

const byte Sp = 0x81;

const byte Acc = 0xe0;

const byte Dph = 0x83;

const byte Dpl = 0x82;

const byte b = 0xf0;

const byte Psw = 0xd0;

const byte p3 = 0x90;

static byte[] R = new byte[8];

#endregion

#region Datafull

private static void DataFull()

{

byte Value = 0x91;

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

{

R[i] = Value++;

}

Ram[Acc] = 0xff;

Ram[b] = 0xf3;

Ram[p3] = 0xab;

Ram[R[1]] = 0xfc;

Ram[1] = 0x2D;

Ram[2] = 0x56;

Ram[3] = 0x02;

Wrk = 0x07;

}

#endregion

#region Codefull

private static void Codefull()

{

//ANL direct, A

Code[PC++] = 0x52;

Code[PC++] = p3;

//ANL direct, #immediate

Code[PC++] = 0x53;

Code[PC++] = 0x01;// Адерс в Ram

Code[PC++] = 0xcc; //Прямое значение

//RRC A

Code[PC++] = 0x13;

//MOV direct, Rn

Code[PC++] = 0x89;

Code[PC++] = b;

//MOV direct, #immediate

Code[PC++] = 0x75;

Code[PC++] = b;

Code[PC++] = 0x45;//Прямое значение

//MOV direct, A

Code[PC++] = 0xf5;

Code[PC++] = 0x02;// Адерс в Ram

//DJNZ direct, offset

Code[PC++] = 0xd5;

Code[PC++] = 0x03;// Адерс в Ram

Code[PC++] = Wrk;

}

#endregion

//Логическое И

#region ANL

private static void ANL(byte P, byte A, bool isA)

{

byte count = 1;

P = Convert.ToByte(P & A);

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

{

if (Convert.ToBoolean((P >> i) & 1)) count++;

}

Ram[Psw] = ((byte)(Ram[Psw] & ~0x01 | count % 2));

if (isA)

Console.WriteLine("Acc=" + A + " P3=" + P);

else

Console.WriteLine("#data=" + A + " direct=" + P);

PSWOut();

return;

}

#endregion

//Сдвиг содердимого вправо

#region RRC

private static void RRC()

{

byte count = 1;

byte C = (byte)(Ram[Psw] & 128); //Сохраняем значение бита С

Ram[Psw] = (byte)((Ram[Psw] & ~0x80) | ((Ram[Acc] & 1) << 7)); //Закидываем нулевой бит аккумулятора в флаг С

Ram[Acc] = (byte)(Ram[Acc] >> 1);

Ram[Acc] = (byte)(Ram[Acc] & ~128 | C);

Console.WriteLine("Acc=" + Ram[Acc]);

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

{

if (Convert.ToBoolean((Ram[Acc] >> i) & 1)) count++;

}

Ram[Psw] = ((byte)(Ram[Psw] & ~0x01 | count % 2));

PSWOut();

}

#endregion

//Перенос значения

#region Move

private static void Move(byte Com, byte direct, byte data)

{

//Получаем тип команды

byte type = (byte)(Com & 0xf8);

switch (type)

{

case 0x88:

byte number = (byte)(Com & 0x07); //Получаем номер регистра

Ram[direct] = Ram[R[number]];

Console.Write("B=" + Ram[direct]);

RNout();

break;

case 0x70:

Ram[direct] = data;

Console.WriteLine("B=" + Ram[b]);

break;

case 0xf0:

Ram[direct] = Ram[Acc];

Console.WriteLine("direct=" + Ram[direct]);

break;

default:

Console.WriteLine("Error");

break;

}

PSWOut();

}

#endregion

private static byte DJNZ(byte direct, byte offset)

{

Ram[direct] -= 0x01;

if (Ram[direct] == 0)

{ Console.WriteLine("\nЗначение равно нулю. Переход не осуществлен"); return 0; }

PC -=offset;

PSWOut();

return 1;

}

//Вывод банка регистров Rn

#region RNout

public static void RNout()

{

Console.WriteLine("\tR0 R1 R2 R3 R4 R5 R6 R7");

Console.Write("\t");

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

{

Console.Write(Ram[R[i]] + " ");

}

Console.WriteLine();

}

#endregion

//Вывод содержимого PSW

#region PSWout

public static void PSWOut()

{

Console.WriteLine("PSW \tP.-.OV.RS0.RS1.F0.AC.C ");

Console.Write("\t");

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

Console.Write(((Ram[Psw] >> i) & 1) + " ");

Console.WriteLine();

}

#endregion

static void Main(string[] args)

{

bool cycle = true;

Console.WriteLine("Команды");

Console.WriteLine("1. ANL direct,A \n2. ANL direct, #immediate \n3. RRC A \n4. MOV direct, Rn \n" +

"5. MOV direct, #immediate \n6. MOV direct, A \n7. DJNZ direct, offset\n");

DataFull();

Codefull();

PC = 0;

while (cycle)

{

Console.WriteLine("Нажимите Enter для продолжения\n");

Console.ReadKey();

switch (Code[PC++])

{

case 0x52://ANL (логическое И) direct, A

Console.WriteLine("ANL direct,A");

Console.WriteLine("------------------------------");

Console.WriteLine("ACC=" + Ram[Acc] + " P3=" + Ram[p3] + " PC=" + (PC-1));

ANL(Ram[Code[PC++]], Ram[Acc], isA: true);

break;

case 0x53://ANL direct, #immediate

Console.WriteLine("ANL direct, #immediate");

Console.WriteLine("------------------------------");

Console.WriteLine("direct=" + Ram[1] + " #data=0xcc" + " PC=" + (PC - 1));

ANL(Ram[Code[PC++]], Code[PC++], isA: false);

break;

case 0x13://RRC (сдвиг содержимого аккумулятора вправо через флаг переноса) A

Console.WriteLine("RRC A ");

Console.WriteLine("------------------------------");

Console.WriteLine("ACC=" + Ram[Acc] + " PC=" + (PC - 1));

RRC();

break;

case 0x89://MOV direct, Rn

Console.WriteLine("MOV direct, Rn");

Console.WriteLine("------------------------------");

Console.WriteLine("B=" + Ram[b] + " PC=" + (PC-1));

Move(Code[PC - 1], Ram[Code[PC++]], 0);

break;

case 0x75://MOV direct, #immediate

Console.WriteLine("MOV direct, #immediate");

Console.WriteLine("------------------------------");

Console.WriteLine("B="+Ram[b]+"Data=" + Code[PC+1] + " PC=" + (PC-1));

Move(Code[PC - 1], Code[PC++], Code[PC++]);

break;

case 0xf5://MOV direct, A

Console.WriteLine("MOV direct, A");

Console.WriteLine("------------------------------");

Console.WriteLine("ACC=" + Ram[Acc] + " direct=" + Ram[Code[PC]] + " PC=" + (PC-1));

Move(Code[PC - 1], Code[PC++], 0);

break;

case 0xd5://DJNZ direct, offset

Console.WriteLine("DJNZ direct, offset");

Console.WriteLine("------------------------------");

Console.WriteLine(" direct=" + Ram[Code[PC]] + " PC=" + (PC-1));

if (DJNZ(Code[PC++], Code[PC]) == 0)

cycle=false;

break;

default:

Console.WriteLine("Error "+"PC="+PC);

break;

}

}

Console.WriteLine("\nВыполнение команд завершено");

Console.ReadKey();

}

1. **Текстовое описание выполнения команды INC A и расчет сложности схемы.**

* Выборка команды.

1. Содержимое счетчика команд, записывается в регистр адреса.
2. Содержимое ячейки памяти, на которую указывает регистр адреса, заносится в регистр команд.

* Декодирование команды.

На данном этапе происходит определение команды, в данном случае это команда RRC A.

* Исполнение.

1. Содержимое аккумулятора в АЛУ сдвигается вправо с учетом флагов.
2. Содержимое счетчика команд в АЛУ увеличивается на 1.

*Расчет сложности схемы:*

Для реализации данной команды необходимы следующие элементы:

* Регистр команд (8 бит),
* Регистр адреса (8 бит),
* Аккумулятор (8 бит),
* АЛУ: два регистра (по 8 бит каждый) и сумматор,
* Мультиплексор 256:1 для выборки ячейки памяти.
* PSW (8 бит)

Для каждого бита регистра нужен один D-триггер. Каждый D-триггер состоит из 4 логических элементов И. 2-ухбитный сумматор состоит из 2 элементов исключающее ИЛИ, 2 элементов И, 1 элемента – логическое ИЛИ. Для 8-мибитного сумматора необходимо: 16 – элементов исключающее ИЛИ, 16 – элементов И, 8 – элементов логическое ИЛИ. Мультиплексор 256:1 состоит из 256-ти элементов И, 8-х элементов НЕ и 1 элемента ИЛИ (265 элементов).

Количество необходимых логических элементов будет равно:

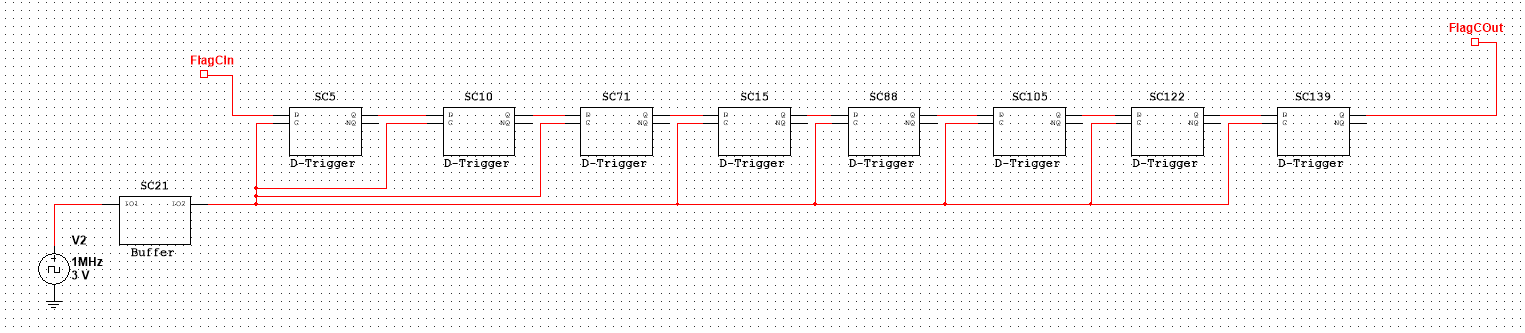
N = ( 8\*4 + 8\*4 + 8\*4 + 2\*(8\*4)+(16+16+8) + (256+8+1) +8\*4) = 497

1. **Структурная схема для команды INC A, выполненная в программе MultiSim**

АЛУ



Регистр сдвига на D-триггерах

*SM8 (8-мибитный сумматор) состоит из нескольких SM, имеющих вид:*



Регистры имеют вид



1. **Оценка сложности всей схемы**

*Для реализации схем выполнения команд необходимы следующие элементы*:

* Регистр команд (8 бит),
* Регистр адреса (8 бит),
* Регистр флагов (8 бит),
* Два мультиплексора 8:1 для адресации битов аккумулятора и регистра флагов,
* Банк регистров общего назначения (8 регистров по 8 бит) с аккумулятором (8 бит),
* АЛУ: два регистра (по 8 бит каждый), сумматор, 8 логических элементов И,
* Мультиплексор 256:1 для выборки ячейки памяти.

Для каждого бита регистра нужен один D-триггер. Каждый D-триггер состоит из четырех логических элементов И. 2-ухбитный сумматор состоит из двух элементов исключающее ИЛИ, двух элементов И, одного элемента – логическое ИЛИ. Для 8-мибитного сумматора необходимо: 16 – элементов исключающее ИЛИ, 16 – элементов И, 8 – элементов логическое ИЛИ. Мультиплексор 8:1 состоит из 8-ми элементов И, 3-х элементов НЕ и одного элемента ИЛИ (12 элементов).

Количество необходимых логических элементов будет равно:

N = ( 8\*4 + 8\*4 + 8\*4 + 2\*(8+3+1) + (8\*8\*4 + 8\*4) + (2\*8\*4 + 8\*2 + 265) ) = 753

1. **Заключение**

В ходе выполнения курсовой работы были:

* разработана схема ЭВМ, реализующая команды **ANL direct, A**; **ANL direct, #data**; **RRC A**; **MOVE direct, Rn**; **Move direct, #data**; **MOVE direct, A; DJNZ direct, offset** .
* реализованы функциональные микропрограммы на языке C#
* произведена оценка сложности и приведено описание команды RRC A
* произведена оценка сложности всей схемы построения ЭВМ

1. **Список литературы**

* <http://www.keil.com>
* Довгий П.С., Скорубский В.И. Проектирование ЭВМ: пособие к выполнению курсового проекта. – СПб: СПбГУ ИТМО, 2009.
* Схемотехника ЭВМ. Кустарев П.В.