Университет ИТМО

**Курсовая работа**

по дисциплине «Организация ЭВМ и систем»

на тему: *«Проектирование ЭВМ»*

Выполнил:

студент III курса

группы 3125

Припадчев Артём

Проверил:

Тропченко А.А.

Санкт-Петербург

2015 г.

Оглавление

[Введение 3](#_Toc419318589)

[Программная модель (микроархитектура) на уровне Ассемблера 4](#_Toc419318590)

[Структура памяти, команды обмена данными 4](#_Toc419318591)

[Арифметические операции 5](#_Toc419318592)

[Логические поразрядные операции 6](#_Toc419318593)

[Битовые логические операции 6](#_Toc419318594)

[Команды управления программой 6](#_Toc419318595)

[Форматы команд 6](#_Toc419318596)

[Описание заданных команд ЭВМ 7](#_Toc419318597)

[Команда ANL <байт-назначения>,<байт-источникa> 7](#_Toc419318598)

[Команда DEC <байт> 8](#_Toc419318599)

[Команда MOV 9](#_Toc419318600)

[Команда JZ <re18> 12](#_Toc419318601)

[Структурная схема и общее описание ее работы 12](#_Toc419318602)

[Листинг эмулирующей работу МК программы с функциональными микропрограммами 15](#_Toc419318603)

[Реализация команды Dec на основе реверсивного счетчика 19](#_Toc419318604)

[Проверка работы схемы 21](#_Toc419318605)

[Реализация реверсивного счетчика на основе двухступенчатого D-триггера 21](#_Toc419318606)

[Проверка работы схемы 22](#_Toc419318607)

[Реализация синхронного реверсивного счетчика с параллельным переносом на основе T-триггеров 23](#_Toc419318608)

[Проверка работы схемы 23](#_Toc419318609)

[Исследование различий времени задержки между асинхронным и синхронным с параллельным переносом счетчиками 24](#_Toc419318610)

[Реализация счетчика с использованием языка ANDL (на D-триггерах) 25](#_Toc419318611)

[Вывод 25](#_Toc419318612)

[Список литературы 25](#_Toc419318613)

# Введение

**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, включая кодирование и принцип исполнения
3. Разработать функциональные микропрограммы для команд в системе Микро51 на языке 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.
6. Реализовать структурную схему в MaxPlus2 в текстовом редакторе на языке AHDL.

# Программная модель (микроархитектура) на уровне Ассемблера

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

Рисунок1

Рисунок 1Программная модель

## 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 64кб 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) Асс, в P2 адрес страницы, @r0 –смещение в странице)

## 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

## 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)

## 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**

## 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)

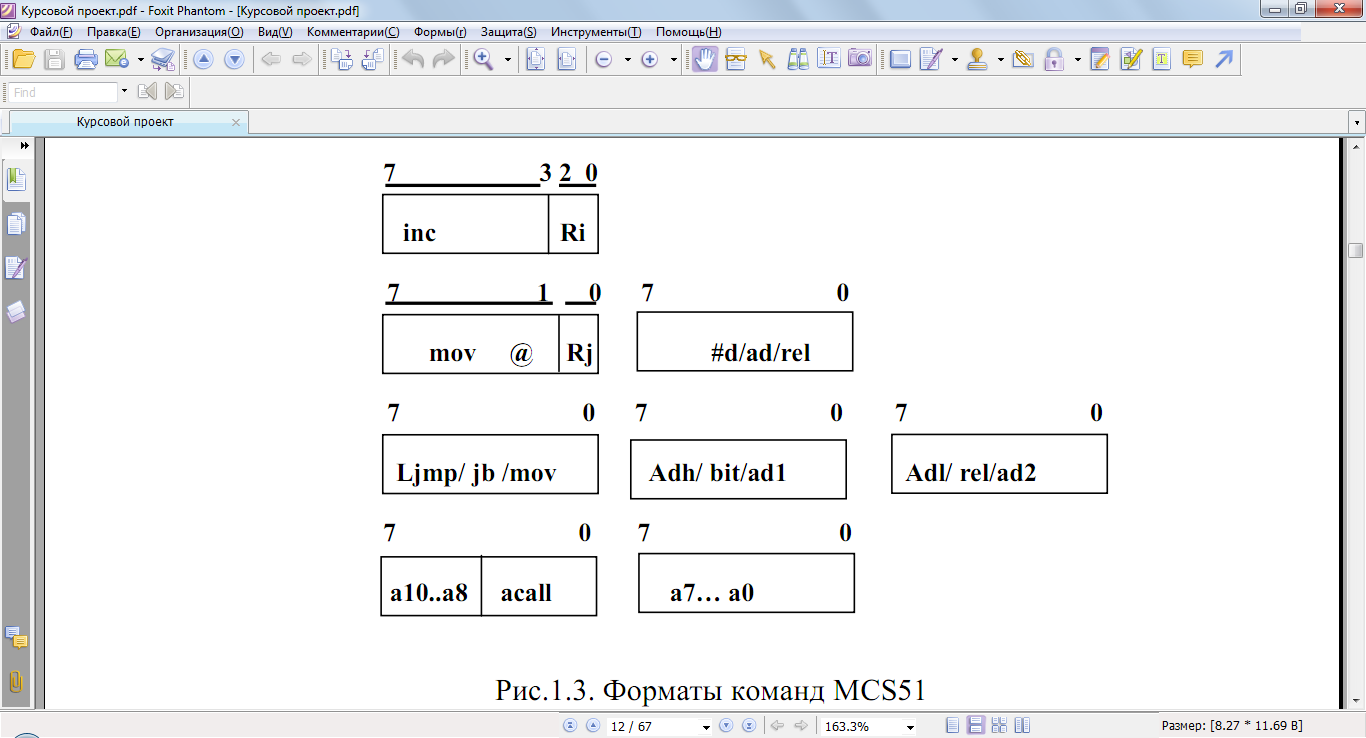
**jb/jnb bit, метка ;**

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;

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



# Описание заданных команд ЭВМ

Команды dec {ri,@rj,ad} anl c,{bit,/bit} mov a,{ri,#d} jz rel

## Команда ANL <байт-назначения>,<байт-источникa>

Команда «логическое "И" для переменных-байтов» выполняет операцию логического "И" над битами указанных переменных и помещает результат в байт-назначения. Эта операция не влияет на состояние флагов.

Для операнда обеспечивают следующие комбинации шести режимов адресации:

байтом назначения является аккумулятор (А):

1. регистровый
2. прямой
3. косвенно-регистровый
4. непосредственный

байтом назначения является прямой адрес (direct):

1. прямой аккумуляторный
2. непосредственный (байт-источник равен константе)

Рассмотрим их.

|  |  |  |
| --- | --- | --- |
| Ассемблер: | ANL A,Rn ; где n=0-7 | |
| Код: | |  | | --- | | 0 1 0 1 1 rrr | | , где rrr=000-111 |
| Время: | 1 цикл | |
| Алгоритм: | (A) : = (A) AND (Rn) | |
| Пример: | ;(A)=FEH, (R2)=C5H  ANL A,R2 ;(A)=C4H, (R2)=C5H | |

|  |  |  |
| --- | --- | --- |
| Ассемблер: | ANL A,<direct> | |
| Код: | |  | | --- | | 0 1 0 1 0 1 0 1 | | |  | | --- | | direct address | |
| Время: | 1 цикл | |
| Алгоритм: | (A) : =(A) AND (direct) | |
| Пример: | ;(A)=A3H, (PSW)=86H  ANL A,PSW ;(A)=82H, (PSW)=86H | |

|  |  |  |
| --- | --- | --- |
| Ассемблер: | ANL A,@Ri ; где i=0,1 | |
| Код: | |  | | --- | | 0 1 0 1 0 1 1 i | | , где i=0,1 |
| Время: | 1 цикл | |
| Алгоритм: | (A) : =(A) AND (Ri) | |
| Пример: | ;(A)=BCH, (ОЗУ [35])=47H, (R0)=35H,  ANL A,@R0 ;(A)=04H, (ОЗУ [35])=47H | |

|  |  |  |
| --- | --- | --- |
| Ассемблер: | ANL A, #data | |
| Код: | |  | | --- | | 0 1 0 1 0 1 0 0 | | |  | | --- | | #data8 | |
| Время: | 1 цикл | |
| Алгоритм: | (A) : = (A)AND #data | |
| Пример: | ;(A)=36H  ANL A,#0DDH ;(A)=14H | |

|  |  |  |
| --- | --- | --- |
| Ассемблер: | ANL <direct>, A | |
| Код: | |  | | --- | | 0 1 0 1 0 0 1 0 | | |  | | --- | | direct address | |
| Время: | 1 цикл | |
| Алгоритм: | (direct) : = (direct) AND (A) | |
| Пример: | ;(A)=55H, (P2)=AAH  ANL P2,A ;(P2)=00H, (A)=55H | |

|  |  |  |  |
| --- | --- | --- | --- |
| Ассемблер: | ANL <direct>, #data | | |
| Код: | |  | | --- | | 0 1 0 1 0 0 1 1 | | |  | | --- | | direct address | | |  | | --- | | #data8 | |
| Время: | 2 циклa | | |
| Алгоритм: | (direct) : = (direct) AND #data | |  |
| Пример: | ;(P1)=FFH  ANL P1,#73H ;(P1)=73H | |  |

## Команда DEC <байт>

Команда "декремент" производит вычитание "1" из указанного операнда. Начальное значение 00Н перейдет в 0FFH. Команда DEC не влияет на флаги. Этой командой допускается четыре режима адресации операнда:

1. к аккумулятору
2. регистровый
3. прямой
4. косвенно-регистровый

Рассмотрим их.

|  |  |  |
| --- | --- | --- |
| Ассемблер: | DEC A | |
| Код: | |  | | --- | | 0 0 0 1 0 1 0 0 | |  |
| Время: | 1 цикл | |
| Алгоритм: | (A) : =(A)-1 | |
| Пример: | ;(A)=11H, (C)=1, (AC)=1  DEC A ;(A)=10H, (C)=1, (AC)=1 | |

|  |  |  |  |
| --- | --- | --- | --- |
| Ассемблер: | DEC Rn ; где n=0-7 | | |
| Код: | |  | | --- | | 0 0 0 1 1 rrr | | где rrr=000-111 |  |
| Время: | 1 цикл | | |
| Алгоритм: | (Rn) :=(Rn)-1 | | |
| Пример: | ;(R1)=7FH,  ;(ОЗУ[7F])=40H, (ОЗУ[7F])=00H  DEC @R1  DEC R1  DEC @R1 ;(R1)=7EH,  ;(ОЗУ[7F])=3FH, (ОЗУ[7F])=FFH | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Ассемблер: | DEC <direct> | | |
| Код: | |  | | --- | | 0 0 0 1 0 1 0 1 | | |  | | --- | | direct address | |  |
| Время: | 1 цикл | | |
| Алгоритм: | (direct) : = (direct)-1 | | |
| Пример: | ;(SCON)=A0H, (C)=1, (AC)=1  DEC SCON ;(SCON)=9FH, (C)=1, (AC)=1 | | |

|  |  |  |
| --- | --- | --- |
| Ассемблер: | DEC @Ri ; где i=0,1 | |
| Код: | |  | | --- | | 0 0 0 1 0 1 1 i | |  |
| Время: | 1 цикл | |
| Алгоритм: | ((Ri) :=((Ri)-1) | |
| Пример: | ;(R1)=7FH,  ;(ОЗУ[7F])=40H, (ОЗУ[7F])=00H  DEC @R1  DEC R1  DEC @R1 ;(R1)=7EH,  ;(ОЗУ[7F])=3FH, (ОЗУ[7F])=FFH | | |

*Примечание.* Если эта команда используется для изменения информации на выходе порта, значение, используемое как исходные данные, считывается из "защелки" порта, а не с выводов БИС.

## Команда MOV

Команда "переслать переменную-байт" пересылает переменную-байт, указанную во втором операнде, в ячейку, указанную в первом операнде. Содержимое байта источника не изменяется. Эта команда на флаги и другие регистры не влияет. Команда "MOV" допускает 15 комбинаций адресации байта-источника и байта-назначения.

|  |  |  |  |
| --- | --- | --- | --- |
| Ассемблер: | MOV A,Rn; где n=0-7 | | |
| Код: | |  | | --- | | 1 1 1 0 1 rrr | | |  | | --- | | где rrr=000-111 | |  |
| Время: | 1 цикл | | |
| Алгоритм: | (A) : =(Rn) | | |
| Пример: | ;(A)=FAH, (R4)=93H  MOV A,R4 ;(A)=93H, (R4)=93H | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ассемблер:** | MOV A, <direct> | | |
| **Код:** | |  | | --- | | 1 1 1 0 0 1 0 1 | | |  | | --- | | direct address | |  |
| **Время:** | 1 цикл | | |
| **Алгоритм:** | (A) :=(direct) | | |
| **Пример:** | ;(A)=93H, (ОЗУ[40])=10H, (R0)=40H  MOV A,40H ;(A)=10H, (ОЗУ[40])=10H, (R0)=40H | | |

|  |  |  |
| --- | --- | --- |
| Ассемблер: | MOV A,@Ri; где i=0,1 | |
| Код: | |  | | --- | | 1 1 1 0 0 1 1 i | |  |
| Время: | 1 цикл | |
| Алгоритм: | (A) : = ((Ri)) | |
| Пример: | ;(A)=10H, (R0)=41H, (ОЗУ[41])=0CAH  MOV A,@R0 ;(A)=CAH, (R0)=41H, (ОЗУ[41])=0CAH | |

|  |  |  |  |
| --- | --- | --- | --- |
| Ассемблер: | MOV A,#data | | |
| Код: | |  | | --- | | 0 1 1 1 0 1 0 0 | | |  | | --- | | #data8 | |  |
| Время: | 1 цикл | | |
| Алгоритм: | (A) :=<#data8> | | |
| Пример: | ;(A)=C9H (11001001B)  MOV A,#37H ;(A)=37H (00110111B) | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Ассемблер: | MOV Rn ,A; где n=0-7 | | |
| Код: | |  | | --- | | 1 1 1 1 1 rrr | | |  | | --- | | где rrr=000-111 | |  |
| Время: | 1 цикл | | |
| Алгоритм: | (Rn) : =(A) | | |
| Пример: | ;(A)=38H, (R0)=42H  MOV R0,A ;(A)=38H, (R0)=38H | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Ассемблер: | MOV Rn, <direct>; где n=0-7 | | |
| Код: | |  | | --- | | 1 0 1 0 1 rrr | | |  | | --- | | direct address | |  |
| Время: | 2 циклa | | |
| Алгоритм: | (Rn) :=(direct) | | |
| Пример: | ;(R0)=39H, (P2)=0F2H  MOV R0,P2 ;(R0)=F2H | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Ассемблер: | MOV Rn,#data; где n=0-7 | | |
| Код: | |  | | --- | | 0 1 1 1 1 rrr | | |  | | --- | | #data8 | |  |
| Время: | 1 цикл | | |
| Алгоритм: | (Rn) :=<#data8> | | |
| Пример: | ;(R0)=0F5H  MOV R0,#49H ;(R0)=49H | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Ассемблер: | MOV <direct>,A | | |
| Код: | |  | | --- | | 1 1 1 1 0 1 0 1 | | |  | | --- | | direct address | |  |
| Время: | 1 цикл | | |
| Алгоритм: | (direct) :=(A) | | |
| Пример: | ;(P0)=FFH, (A)=4BH  MOV P0,A ;(P0)=4BH, (A)=4BH | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Ассемблер: | MOV <direct>, Rn ; где n=0-7 | | |
| Код: | |  | | --- | | 1 0 0 0 1 rrr | | |  | | --- | | direct address | |  |
| Время: | 2 циклa | | |
| Алгоритм: | (direct) :=(Rn) | | |
| Пример: | ;(PSW)=C2H, (R7)=57H  MOV PSW,R7 ;(PSW)=57H, (R7)=57H | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ассемблер: | MOV <direct>, <direct> | | | |
| Код: | |  | | --- | | 1 0 0 0 0 1 0 1 | | |  | | --- | | direct address | | |  | | --- | | direct address | |  |
| Время: | 2 циклa | | | |
| Алгоритм: | (direct) :=(direct) | | | |
| Пример: | ;(ОЗУ[45])=33H, (ОЗУ[48])=0DEH  MOV 48H,45H ;(ОЗУ[45])=33H, (ОЗУ[45])=33H | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Ассемблер: | MOV <direct>,@Ri ; где i=0,1 | | |
| Код: | |  | | --- | | 1 0 0 0 0 1 1 i | | |  | | --- | | direct address | |  |
| Время: | 2 циклa | | |
| Алгоритм: | (direct) :=((Ri)) | | |
| Пример: | ;(R1)=49H, (ОЗУ[49])=0E3H  MOV 51H,@R1 ;(ОЗУ[51])=0E3H, (ОЗУ[49])=0E3H | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ассемблер: | MOV <direct>, #data | | | |
| Код: | |  | | --- | | 0 1 1 1 0 1 0 1 | | |  | | --- | | direct address | | |  | | --- | | #data8 | |  |
| Время: | 2 циклa | | | |
| Алгоритм: | (direct) :=<#data8> | | | |
| Пример: | ;(ОЗУ[5F])=9BH  MOV 5FH,#07H ;(ОЗУ[5F])=07H | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Ассемблер: | MOV @Ri,A; где i=0-7 | | |
| Код: | |  | | --- | | 1 1 1 1 0 1 1 i | | |  | | --- | | где i=0,1 | |  |
| Время: | 1 цикл | | |
| Алгоритм: | ((Ri)) : =(A) | | |
| Пример: | ;(R1)=51H, (ОЗУ[48])=75H, (A)=0BDH  MOV @R1,A ;(ОЗУ[48])=0BDH | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Ассемблер: | MOV @Ri, <direct>, где i=0,1 | | |
| Код: | |  | | --- | | 1 0 1 0 0 1 1 i | | |  | | --- | | direct address | |  |
| Время: | 2 циклa | | |
| Алгоритм: | ((Ri)) :=(direct) | | |
| Пример: | ;(R0)=51H, (ОЗУ[51])=0E3H, (P0)=0ACH  MOV @R0,P0 ;(A)=10H, (ОЗУ[51])=0ACH | | |

|  |  |  |
| --- | --- | --- |
| **Ассемблер:** | MOV Ri,#data ; где i=0,1 | |
| **Код:** | |  | | --- | | 0 1 1 1 0 1 1 i | |  |
| **Время:** | 1 цикл | |
| **Алгоритм:** | ((Ri)) : =<#data8> | |
| **Пример:** | ;(ОЗУ[7E])=67H, (R1)=7EH  MOV @R1,#0A9H ;(ОЗУ[7E])=0A9H, (R1)=7EH | |

## Команда JZ <re18>

Команда "переход, если содержимое аккумулятора равно 0" выполняет ветвление по адресу, если все биты аккумулятора равны "0", в противном случае выполняется следующая команда. Адрес ветвления вычисляется сложением относительного смещения со знаком во втором байте команды (re18) и содержимым счетчика команд после прибавления к нему 2. Содержимое аккумулятора не изменяется. Эта команда на флаги не влияет.

|  |  |  |  |
| --- | --- | --- | --- |
| Ассемблер: | JZ <метка> | | |
| Код: | |  | | --- | | 0 1 1 0 0 0 0 0 | | |  | | --- | | re18 | |  |
| Время: | 2 циклa | | |
| Алгоритм: | (PC):=(PC)+2 если (A)0, то (PC):=(PC)+<re18> | | |
| Пример: | ;(A)=01H  JZ LAB16 ;нет перехода на LAB16  DEC A  LAB16: JZ LAB17 ;переход на метку LAB17  ...  LAB17: CLR A | | |

# Структурная схема и общее описание ее работы

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

Следующие принципы учтены при выборе структуры:

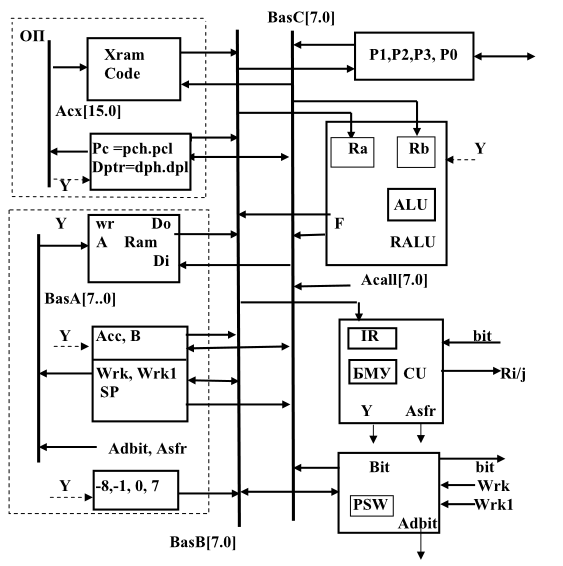
1. Иерархический подход к проектированию схем, который поддерживается в MaxPlus. На первом этапе выполняется функциональное неформальное разбиение схемы на функциональные блоки с учетом распределения памяти по блокам и функциональным элементам, определяемым в программной модели.
2. Используется шинная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

1. 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.
2. 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.
3. К регистрам может быть обеспечен как регулярный адресный доступ через мультиплексоры, так и непосредственный для контроля и работы с отдельными битами и полями битов.
4. 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).



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.

# Листинг эмулирующей работу МК программы с функциональными микропрограммами

class Program

{

#region vars

/// <summary>

/// Аккумулятор

/// </summary>

readonly static Register Akk = new Register(16);

/// <summary>

/// Регистр команд

/// </summary>

readonly static Register CommandReg = new Register(16);

/// <summary>

/// Память

/// </summary>

readonly static MemoryCell[] Memory = new MemoryCell[1024];

#endregion

#region Anl

/// <summary>

/// Метод - эмулятор команды ANL

/// </summary>

/// <param name="a">Регистр назначения</param>

/// <param name="b">Регистр - участник операции</param>

/// <param name="isRef">Флаг косвенной адресации</param>

private static void Anl(Register a, Register b, bool isRef = false)

{

Anl(a, !isRef ? b.Value : Memory[b.Value].Value);

}

/// <summary>

/// Метод - эмулятор команды ANL

/// </summary>

/// <param name="a">Регистр назначения</param>

/// <param name="b">Значение-участник или значение-адрес(в зависимости от isRef)</param>

/// <param name="isRef">Флаг является ли адресация косвенной</param>

private static void Anl(Register a, Int16 b, bool isRef = false)

{

if (!isRef)

{

a.Value = (Int16)(a.Value & b);

}

else

{

a.Value = (Int16)(a.Value & Memory[b].Value);

}

}

#endregion

#region JZ

/// <summary>

/// Метод - эмулятор команды JZ

/// </summary>

/// <param name="a">Адрес перехода</param>

private static void Jz(Int16 a)

{

if (Akk.Value == 0)

CommandReg.Value = a;

}

#endregion

#region DEC

/// <summary>

/// Метод-эмулятор команды DEC

/// </summary>

/// <param name="a">Регистр-участник</param>

/// <param name="isRef">Флаг косвенной адресации</param>

private static void Dec(Register a, bool isRef = false)

{

if (!isRef)

{

a--;

}

else

{

Dec(a.Value);

}

}

/// <summary>

/// Метод-эмулятор команды DEC

/// </summary>

/// <param name="a">Адрес ячейки-участницы</param>

private static void Dec(Int16 a)

{

Memory[a]--;

}

#endregion

#region MOV

/// <summary>

/// Метод - эмулятор команды Mov

/// </summary>

/// <param name="a">Регистр назначения</param>

/// <param name="b">Регистр - участник операции</param>

/// <param name="isARef">Флаг косвенной адресации регистра а</param>

/// <param name="isBRef">Флаг косвенной адресации регистра b</param>

private static void Mov(Register a, Register b, bool isARef = false, bool isBRef = false)

{

if (!isARef)

{

a.Value = !isBRef ? b.Value : Memory[b.Value].Value;

return;

}

Memory[a.Value].Value = !isBRef ? b.Value : Memory[b.Value].Value;

}

/// <summary>

/// Метод - эмулятор команды Mov

/// </summary>

/// <param name="a">Регистр назначения</param>

/// <param name="b">Константа - адрес, или константа-значение (в зависимости от isBRef)</param>

/// <param name="isARef">Флаг косвенной адресации регистра а</param>

/// <param name="isBRef">Флаг косвенной адресации b</param>

private static void Mov(Register a, Int16 b, bool isARef = false, bool isBRef = false)

{

if (!isARef)

{

a.Value = !isBRef ? b : Memory[b].Value;

return;

}

Memory[a.Value].Value = !isBRef ? b : Memory[b].Value;

}

#endregion

static void Main(string[] args)

{

Console.WriteLine("\*\*\* Test MOV(akk, reg) \*\*\*");

Register reg = new Register(16,5);

Akk.Value = 0;

Console.WriteLine("Akk = " + Akk.Value);

Console.WriteLine("Reg = " + reg.Value);

Console.WriteLine("Execute MOV");

Mov(Akk, reg);

Console.WriteLine("Akk = " + Akk.Value);

Console.WriteLine();

Console.WriteLine("\*\*\* Test MOV(akk, @Reg) \*\*\*");

Memory[40] = new MemoryCell(16, 0);

Console.WriteLine("Memory[40] = " + Memory[40].Value);

reg.Value = 40;

Console.WriteLine("Reg = " + reg.Value);

Console.WriteLine("Execute MOV");

Mov(reg, Akk, true);

Console.WriteLine("Memory[40] = " + Memory[40].Value);

Console.WriteLine();

Console.WriteLine("\*\*\* Test negative value \*\*\*");

Akk.Value = -10;

Console.WriteLine("Akk(10) = " + Akk.Value);

Console.WriteLine("Akk(2) = " +Akk.GetValueWithCounting(2));

Console.WriteLine();

Console.WriteLine("\*\*\* Test DEC(reg) \*\*\*");

Akk.Value = 2;

Console.WriteLine("Akk = " + Akk.Value);

Console.WriteLine("Execute DEC");

Dec(Akk);

Console.WriteLine("Akk = " + Akk.Value);

Console.WriteLine();

Console.WriteLine("\*\*\* Test DEC(reg) reg = 0 \*\*\*");

Akk.Value = 0;

Console.WriteLine("Akk = " + Akk.Value);

Console.WriteLine("Execute DEC");

Dec(Akk);

Console.WriteLine("Akk(16) = " + Akk.GetValueWithCounting(16));

Console.WriteLine();

Console.WriteLine("\*\*\* Test DEC(@Reg) \*\*\*");

Memory[1] = new MemoryCell(16, 10);

Console.WriteLine("Memory[1] = " + Memory[1].Value);

Akk.Value = 1;

Console.WriteLine("Akk = " + Akk.Value);

Console.WriteLine("Execute DEC");

Dec(Akk, true);

Console.WriteLine("Memory[1] = " + Memory[1].Value);

Console.WriteLine();

Console.WriteLine("\*\*\* Test DEC(Memory[i]) \*\*\*");

Console.WriteLine("Execute DEC");

Dec(1);

Console.WriteLine("Memory[1] = " + Memory[1].Value);

Console.WriteLine();

Console.WriteLine("\*\*\* Test JZ(15) \*\*\*");

Akk.Value = 0;

Console.WriteLine("Akk = " + Akk.Value);

Console.WriteLine("CommandReg = " + CommandReg.Value);

Console.WriteLine("Execute JZ");

Jz(15);

Console.WriteLine("CommandReg = " + CommandReg.Value);

Console.WriteLine();

Console.WriteLine("\*\*\* Test ANL(reg, @reg) \*\*\*");

Memory[60] = new MemoryCell(16, 9);

Console.WriteLine("Memory[60] = " + Memory[60].GetValueWithCounting(2));

Akk.Value = 5;

Console.WriteLine("Akk = " + Akk.GetValueWithCounting(2));

reg.Value = 60;

Console.WriteLine("Reg = " + reg.Value);

Console.WriteLine("Execute ANL");

Anl(Akk,reg, true);

Console.WriteLine("Akk = " + Akk.GetValueWithCounting(2));

Console.ReadLine();

}

}

/// <summary>

/// Класс, представляющий ячейку памяти

/// </summary>

internal class MemoryCell

{

public static MemoryCell operator --(MemoryCell a)

{

if (a.Value == 0)

a.Value = Int16.Parse(new string('f', (a.Capacity - 1) / 4), NumberStyles.AllowHexSpecifier);

else

a.Value--;

return a;

}

private Int16 \_value;

/// <summary>

/// Текущее значение регистра

/// </summary>

public Int16 Value

{

get { return \_value; }

set

{

if (Convert.ToString(value, 2).Length > Capacity)

throw new Exception("Разрядность регистра меньше устанавливаемого занчения");

\_value = value;

}

}

/// <summary>

/// Строковое представление значения регистра в соответствии с системой исчисления

/// </summary>

/// <param name="counting">Основание системы исчисления</param>

/// <returns>Строковое представление значения регистра в соответствии с системой исчисления</returns>

public string GetValueWithCounting(Int16 counting)

{

var tempResult = Convert.ToString(Value, counting);

var charCounter = Capacity / (Int16)Math.Log(counting, 2);

return tempResult.Length < charCounter

? string.Format("{0}{1}", new String('0', charCounter - tempResult.Length), tempResult)

: tempResult.Substring(tempResult.Length - charCounter);

}

/// <summary>

/// Разрядность регистра

/// </summary>

public byte Capacity { get; set; }

/// <summary>

/// Конструктор класса Register

/// </summary>

/// <param name="capacity">Разрядность регистра(до 32)</param>

public MemoryCell(byte capacity)

{

if (capacity > 32)

throw new ArgumentOutOfRangeException("capacity");

Capacity = capacity;

\_value = 0;

}

public MemoryCell(byte capacity, Int16 value)

: this(capacity)

{

Value = value;

}

}

/// <summary>

/// Класс, представляющий регистр

/// </summary>

internal class Register : MemoryCell

{

public Register(byte capacity)

: base(capacity)

{

}

public Register(byte capacity, Int16 value)

: this(capacity)

{

Value = value;

}

public static Register operator --(Register a)

{

if (a.Value == 0)

a.Value = Int16.Parse(new string('f', (a.Capacity - 1) / 4), NumberStyles.AllowHexSpecifier);

else

a.Value--;

return a;

}

}

# Реализация команды Dec на основе реверсивного счетчика

**Логические элементы, использованные в схеме:**

**Синхронный RS-триггер**





**T-триггер**





**И-ИЛИ**





**Реверсивный счетчик**



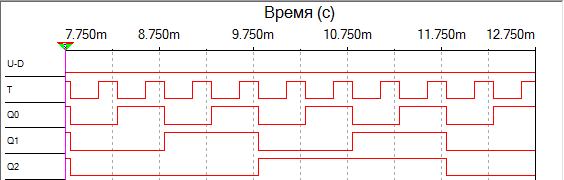


**Схема моделирования:**

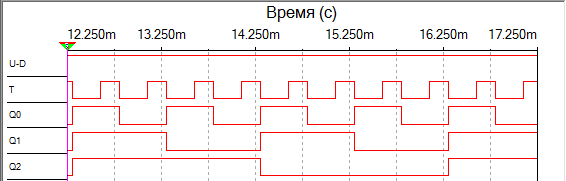
****

Проверка работы схемы (результат моделирования в логическом анализаторе)

Прямой ход (U-D = 0)



Обратный ход (U-D = 1)



Реализация реверсивного счетчика на основе двухступенчатого D-триггера (триггер с управлением по спаду, построенной по технологии Master-Slave)

Элементы, использованные в схеме:

Двухступенчатый D-триггер с инвертором:



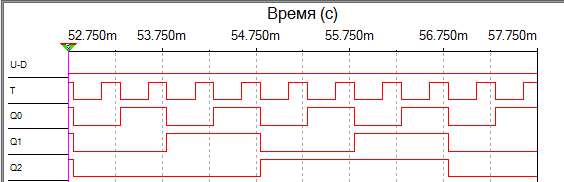




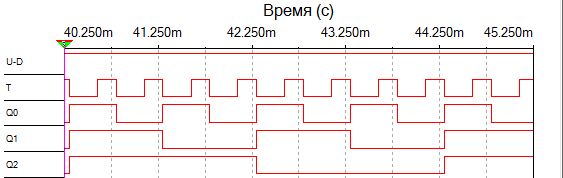


Проверка работы схемы (результат моделирования в логическом анализаторе)

Прямой ход (U-D = 0)



Обратный ход (U-D = 1)

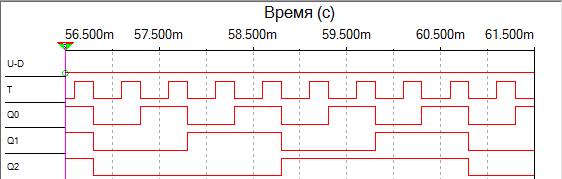


Реализация синхронного реверсивного счетчика с параллельным переносом на основе T-триггеров:

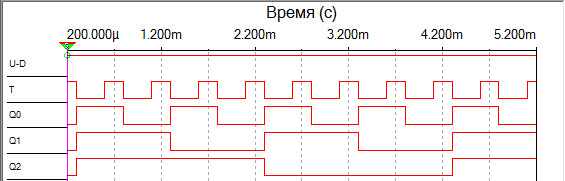


Проверка работы схемы (результат моделирования в логическом анализаторе)

Прямой ход (U-D = 0)



Обратный ход (U-D = 1)



## Исследование различий времени задержки между асинхронным и синхронным с параллельным переносом счетчиками

Для исследования используем комбинацию при которой происходит изменение выходных сигналов Q0,Q1,Q2 c 000 на 111 и обратно:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Асинхронный счетчик | | Синхронный счетчик | |
| По фронту | По спаду | По фронту | По спаду |
| Q0 | 44 ns | 59 ns | 44 ns | 59 ns |
| Q1 | 132 ns | 147 ns | 168 ns | 184 ns |
| Q2 | 221 ns | 236 ns | 184 ns | 198 ns |

|  |  |
| --- | --- |
| Асинхронный по фронту | Асинхронный по спаду |
| Синхронный по фронту | Синхронный по спаду |

Из временных диаграмм и таблицы можно сделать вывод, что использование синхронных счетчиков относительно асинхронных тем целесообразнее, чем больше разрядность счетчика

# Реализация счетчика с использованием языка ANDL (на D-триггерах)

Счетчиками называются последовательные логические схемы для счета тактовых импульсов. В некоторых счетчиках реализован счет вперед и назад (реверсивные счетчики), в некоторые счетчики можно загружать данные, а также обнулять их. Счетчики обычно определяют как D-триггеры (DFF и DFFE) и используют операторы IF.

Ниже приведена реализация 16-битного загружаемого счетчика со сбросом.

SUBDESIGN kurs

(

clk, load, ena, clr, d[15..0] : INPUT;

q[15..0] : OUTPUT;

)

VARIABLE

count[15..0] : DFF;

BEGIN

count[].clk = clk;

count[].clrn = !clr;

IF load THEN

count[].d = d[];

ELSIF ena THEN

count[].d = count[].q + 1;

ELSE

count[].d = count[].q;

END IF;

q[] = count[];

END;

В данном коде в секции VARIABLE объявлены 16 D-триггеров и им присвоены имена от **count0** до **count15.** В операторе IF определяется значение, загружаемое в триггеры по фронту синхросигнала (например, если загрузка запускается VCC, то триггерам присваивается значение **d[** **]**).

# Вывод

В рамках выполнения курсовой работы были рассмотрены некоторые команды (dec {ri,@rj,ad} anl c,{bit,/bit} mov a,{ri,#d} jz rel), их характеристики и особенности, разработана их реализация на языке c#, а также разработана схема, позволяющая реализовывать функционал одной из команд (dec) на основе реверсивного счетчика, схема была создана путем редактирования в графическом редакторе, а также с использованием возможностей языка AHDL.

# 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.