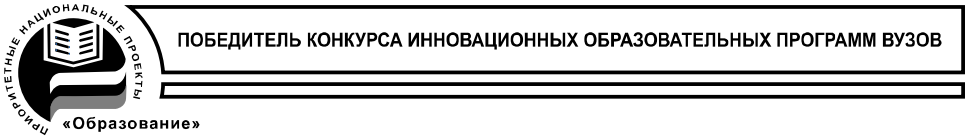
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ



П.С. Довгий, В.И. Скорубский

ПРОЕКТИРОВАНИЕ ЭВМ

Пособие к выполнению курсового проекта

Санкт-Петербург

2009

Довгий П.С., Скорубский В.И. Проектирование ЭВМ: пособие к выполнению курсового проекта. – СПб: СПбГУ ИТМО, 2009. – 68 с.

Пособие содержит методику проектирования компьютера со сложной системой команд CISC.

Для демонстрации методики используется программная модель микрокомпьютера MCS51, которая изучается в курсе лабораторных работ Организация ЭВМ.

Приводится более подробное описание системы команд, ориентированное на разработку схемы и микропрограммирование.

Проектирование выполняется в интегрированной среде проектирования MaxPlusII фирмы Altera с использованием библиотеки функциональных элементов и программируемых логических модулей.

Приведены примеры проектирования структурных схем и синтеза функциональных схем в MaxPlus. На разных этапах проекта выполняется тестирование и верификация с использованием моделирования в BorlandC++ и Симулятора MaxPlus.

Пособие предназначено для студентов, изучающих курс «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 8 декабря 2009 г., протокол № 5.

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.

Содержание

Введение

Целью курсового проекта является разработка микропрограммного управления и схемы ЭВМ с архитектурой CISC и системой команд микроЭВМ (микрокомпьютер, MCU) **MCS51**.

Исходными данными являются программная модель на уровне Ассемблера, перечень команд, выполняемых схемой, и элементная база **MaxPlus**.

При выполнении проекта предполагается, что пройден курс лабораторных работ, где изучается программная модель [1].

Для функционального описания микропрограмм и моделирования могут быть использованы языки программирования, наиболее близким из которых является язык Си в системе **BorlandC++**.

Схема проекта разрабатывается в системе **MaxPlus** и загружается в ПЛИС фирмы Алтера[2]. Верификация проекта выполняется в Симуляторе MaxPlus.

Для описания, визуального моделирования, кодирования и создания загрузочных файлов в проекте MaxPlus используется система **Borland C++.**

I. Программная модель

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

В описании системы команд Keil/Help [1] фирма ссылается на обозначения соответствующих устройств и ресурсов, для описания содержания команд применяются элементы формальных языков. В проекте используем элементы неформального языка регистровых передач в виде **комментариев** к командам в Ассемблере.

Программная модель включает распределение памяти ЭВМ между различными функциональными блоками. Принципы работы компьютера приведены в описании системы команд и сохраняют общий порядок **(цикл)** исполнения программы − выборка кода команды из программной памяти **Code** по адресу в программном счетчике PC, выборка операндов из памяти данных, исполнение операции и сохранение результатов в памяти данных. Существенным отличием **MCU** от **ЭВМ общего назначения** с конструктивным разделением процессора и памяти является интеграция этих устройств в одной схеме, разделение памяти программ и данных, выполнение памяти программ **Code** в ПЗУ.

В проекте не рассматривается работа и организация последовательного канала ввода-вывода, системы прерываний и таймеров. Предполагается, что эти вопросы относятся к курсу «Системы ввода-вывода».

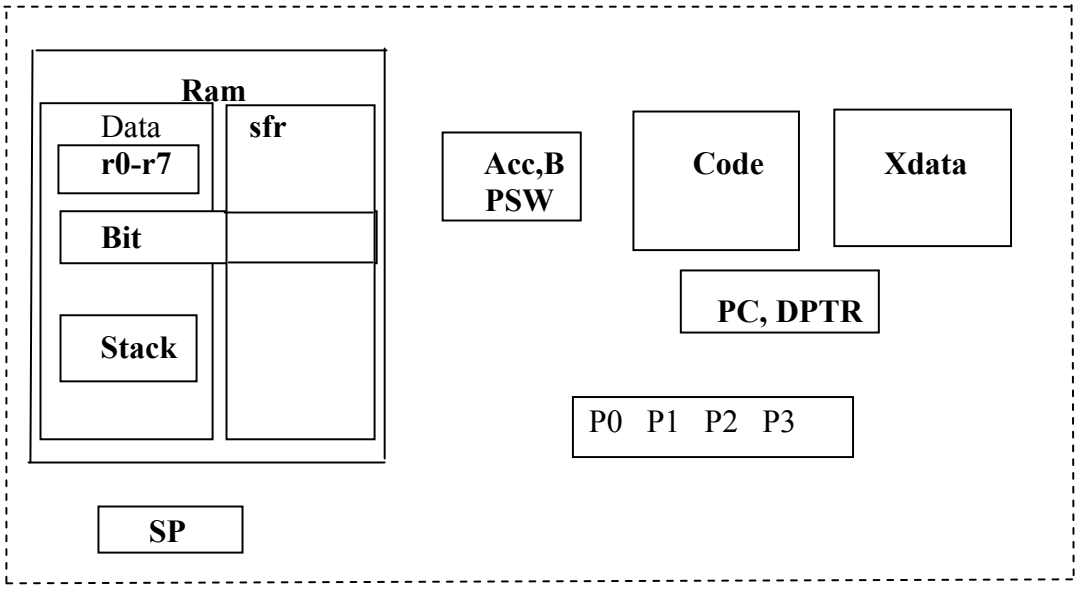


Рис.1.1. Программная модель

1.1. Структура памяти, команды обмена данными

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

Память различается по способу доступа к информации:

- **регистровая с прямым доступом** по назначению, определяемому содержанием команды. В Ассемблере регистрам присваиваются смысловые идентификаторы. Состояние регистров изменяется аппаратно и программно;
- **память с адресным доступом** – адрес ячейки (слова) памяти задается кодом команды. Значения ячеек изменяются программно (Ram, Xram) или загрузчиком программы в память Code типа EPROM;
- **регистры с совмещенным доступом.** В аппаратуре предусмотрены **рабочий регистр** с быстрым прямым доступом для чтения и записи и **теневой регистр** в SFR с адресным доступом. Состояния регистров идентичны и обновляются при изменении состояния рабочего регистра.

1. Основные Регистры.

Используются в основном цикле исполнения программы – их относят к памяти процессора при выделении его в архитектуре ЭВМ.

a(Acc) – основной регистр-аккумулятор, применяемый во всех арифметических и логических операциях с неявным (прямым) доступом и явным адресным. При этом в Ассемблере используется обозначение **mov a, B**, где аккумулятор неявно (прямо) доступен, и **mov Acc,B**, где аккумулятор адресуется в Ram.

B − рабочий регистр, также неявно подразумеваемый в командах **mul ab, div ab** или адресуемый в команде **mov a,B**.

Регистр состояния **PSW=C.AC.F0.RS1.RS0.OV.-.P** содержит признаки результата арифметических операций – **C** (перенос, заем), **AC** − полуперенос, **OV** (знаковое переполнение), **P**(бит четности двоичного кода)**, F0** (бит пользователя), **RS1-RS0** – номер активного регистрового банка. Регистр PSW неявно изменяется при выполнении операций и доступен по адресу в команде **mov a, PSW**.

PC − 16-разрядный программный счетчик, или регистр адреса команды. При включении питания автоматически сбрасывается. Таким образом, в MCS51 начальный запуск программы с адреса 0000. PC доступен только неявно в командах управления программой и адресации к массиву данных.

DPTR=DPH.DPL – 16-разрядный адресный регистр (Data Pointer Register). Используется для адресации памяти Code, Xdata с неявным доступом. Возможно адресное обращение к теневым регистрам Dph, Dpl в Ram.

2. Память Ram – 256 байт разделена на два блока Data [0-7Fh] и SFR[80h-FFh].

Прямая адресация **ad** к RAM в командах.

mov a, 55h ; RAM[55h] → Acc
mov ad, a ; a →RAM[ad**].** где ad - прямой адрес ячейки RAM
mov ad1, ad2 ; RAM[ad1] → RAM[ad2], где ad1,ad2 – первый и второй прямые адреса.
mov 22h, 33h ; RAM[22h] → RAM[33h]

1) Регистры специальных функций **SFR** с прямой адресацией (80-FFh) входят как подмножество регистров в адресуемую память **RAM**.

Объем 128 байт. SFR содержит теневые регистры для основных регистров **(ACC, B, DPTR, PSW, SP)** и адресуемые регистры управления периферией.

2) Оперативная память данных Data – структура иерархическая по назначению и доступу, занимает в MCS51 адреса 0-7Fh памяти данных Ram. Следующий уровень иерархии – регистровая, битовая, стековая память.

a) Активный банк **регистров общего назначения Ri={ R0,R1,..R7** }.

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] → Acc
mov R1,a ; Acc → Data[R1]

Два регистра Rj={R0,R1} используются в **косвенной адресации**

mov a,@R0 ; Data[R0] →Acc
mov @R0, 22h ; Data[22h] → Data[R1]
mov @R0,ACC , SFR[Acc] →Data[R0]

b) Bit – 128 бит, прямой адрес бита 0-7fH, память совмещена с ячейками 20-2f Data, еще 128 бит с с адресами 80h-ffh относятся к SFR

mov c, 0 ; Data[20h.0] → C , где C=20h.0 – нулевой бит ячейки Data
mov ACC.7, c ; **C → Acc.7,**
mov c, x0 ; **x0 -** имя бита

c) 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 SFR[Acc] → Data[+SP]

pop ad

Например**, pop 22h** обозначает Data[SP--] → Data[22h]

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

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

mov a,#d ; Code[PC+] → Acc , #d непосредственный операнд
movc a,@a+pc ; Code[PC + Acc] → Acc **;** адресация относительно
; текущего PC**,** в ACC индекс
movc a,@a+dptr ; Code[dptr + Acc] → Acc ; базовая индексная
адресация- база в DPTR, в ACC смещение (индекс)

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

movx a, @dptr ; Xdata[dptr] → Acc
movx @dptr,a ;
movx a, @r0 ; Xdata[P2.@r0] → Acc, в P2 адрес страницы, @r0 – смещение в странице, P2.@r0 обозначает конкатенацию)

1.2. Арифметические операции

Используются следующие форматы данных:

- знаковое 8-разрядное целое;
- беззнаковое 8-разрядное целое;
- 8-разрядный двоичный код;
- биты;
- 2-х разрядное десятичное число в 8-4-2-1 коде.

а) Знаковая Арифметика.

Отрицательные числа традиционно представлены дополнительными кодами:

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 →
add a,P2 ; a + P2→ a **P2-регистр** порта P2

b) Беззнаковая арифметика.

inc {a, ri, @rj, ad, dptr} **; {..}+1,** признак P
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

c) Десятичная арифметика.

Работа с десятичными данными поддерживается командами:

DA a – десятичная коррекция результатов двоичного сложения или вычитания 2/10 чисел, представленных двумя цифрами в байте.

Swap a – обмен тетрадами в Acc[7.4]←→Acc[3.0]

Xchd a, @rj - обмен тетрадами

1.3. Логические поразрядные операции

anl a, {Ri,@rj,#d,ad} ; a & {..} → a признаки P, 0 → c,
anl ad, {#d, a} ;
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}
cpl a ; не a
rr a ; циклический сдвиг Acc вправо (признак C не изменяется)
rl a ; циклический сдвиг Acc влево (признак C не изменяется)

1.4. Битовые логические операции

anl c,{bit, /bit}
bit – прямой адрес бита, /bit – инверсия бита **;**
Например, **anl c,/ACC.6**
orl c,{bit , /bit}
setb bit **; 1** → bit
clr bit **; 0 → bit**
cpl C

1.5. Параллельный ввод-вывод

Осуществляется через порты P0,P2,P3,P1.

В проекте предполагается общая схема для всех портов и включает регистр и буферы входные и выходные, связанные с внешними контактами микросхемы.

Чтение (ввод) – считывание и передача в регистры состояния контактов(pins), которые в дальнейшем трактуются как биты данных

mov a, P2 ; pin P2 → Acc;

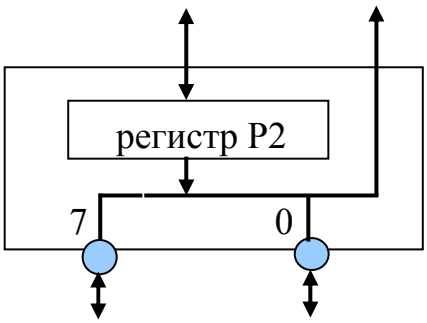


Рис. 1.2. Схема порта

Операции с портами с записью (выводом) в порт обращаются к регистру порта Pi.

mov P2,a – запись в регистр P2, состояние регистра по низкому уровню объединяется с контактами **/P1 v /pin,** для ввода с контактов необходимо в соответствующих разрядах регистра установить единицы.

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

Возможна независимая работа с отдельными битами портов

anl C, P2.5 ; P2.5&C → C, бит читается с регистра P2

mov C, P2.5 ; ввод бита с контакта порта P2.5

1.6. Команды управления программой

К ним относятся команды ветвления, формирующие состояние программного счетчика PC:

jmp метка **;** адрес метки →PC

(**sjmp diff** ; PC+diff , где diff – 8 бит смещение в доп.коде,

ajmp adr10 **;** смещение в текущей странице, PC[15.10].adr10,

ljmp adr16 **;** PC=adr16)

call метка **;** PC → Stack[+SP], адрес метки → PC, переход к подпрограмме

(**acall adr10** ; смещение в текущей странице, PC[15.10].adr10
lcall adr16 ; PC=adr16)
ret ; Stack[SP-] → PC возврат из подпрограммы
jc/jnc diff ; если (с/не с), PC+diff
jz/jnz diff ; переход, если ACC (=0)/(!=0)
jb/jnb bit, diff ; jb ACC.0,start – переход по значению бита
djnz {ri,ad}, diff ; [{..}-1, if ({..}#0), то PC+diff]
cjne (ri,@rj,ad} ,#d, diff ; if ({..}#d), то PC+diff;

Обзор мнемокодов системы команд в приложении 1, подробное описание приведено в Keil. Help.

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

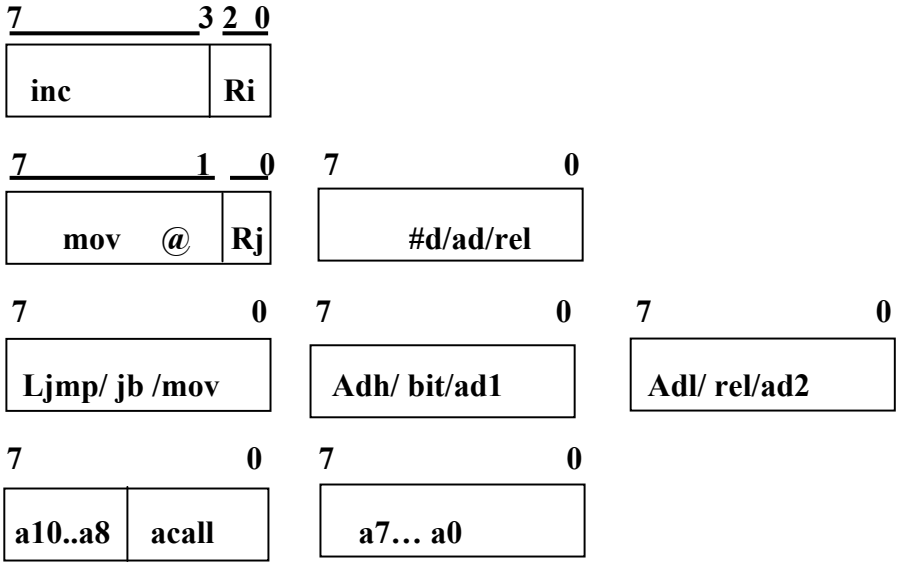


Рис.1.3. Форматы команд MCS51

II. Этапы проектирования ЭВМ

Схемы ЭВМ строятся на основе программной модели по традиционной структуре, включающей **исполнительное устройство**, **память** и **управляющее устройство**.

При большом объеме ручной работы этапы логического синтеза и кодирования автоматизированы и поддерживаются средствами отладки.

В проекте кодирование и отладка микропрограмм выполняется как самостоятельная часть задания.

Одной из доступных и широко распространенных является система проектирования **MaxPlus[2,3]** на основе ПЛИС фирмы Altera или ее расширенная и профессиональная версия **Cuartus**. Высокая сложность ПЛИС позволяет в одном проекте выполнить схемы однокристального компьютера,

Округленные параметры ПЛИС высокой сложности серии FLEX10K приведены в таблице:

Device	LCells	Flipflops	Memory bits	I/O
EPF10K100E	5000	5400	50000	270
200E	10000	10200	100000	460

LCell – ячейка (единица измерения логической сложности), приблизительно эквивалента одному триггеру или 10 битам памяти.

Однако одной ПЛИС недостаточно для реализации всех ресурсов MCS51:

- память Code в полном объеме 65 000*8 бит;
- память Xram 65 000*8 бит;
- память RAM 256*8 бит;
- в управляющем устройстве CU (рис. 6.1.)
  Память ROM ADC 256*8 бит
  Память ROMM 1024*8 бит
  Память ROM DCM 32*100 бит
  Декодеры ROM 7*8 + 20*100 бит
- регистровая память 15*8 =120 dff

Приняты ограничения, позволяющие выполнить проект на EPF10K100E в MaxPlus:

- Code 4*256*8 ~ 4 Кбит
- Xram 4*256*8 ~ 4 Кбит
- RAM 256*8 =1024 бит
  Память ROM ADC 256*8 =1024бит
  Память ROMM 100*8 =1024бит
  Память ROM DCM 32*30 ~1000 бит
  Декодеры ROM 7*8 + 20*30 ~800 бит
- регистровая память 15*8 =120 триггеров

Итого – около 13 кбит памяти и 120 триггеров dff.

Для сложных схем используется иерархический подход к проектированию.

1) На первом этапе выполняется функциональное неформальное разбиение схемы на функциональные блоки.

Организация связей между блоками (внутренние интерфейсы в ЭВМ) при выполнении команд оформляется в виде **структурной схемы**, которая является основой для дальнейшей детализации в виде более подробных структурных

схем отдельных блоков и функциональных схем в графике и функциональных обозначениях MaxPlus.

В структурной схеме связи позволяет описывать всевозможные передачи и преобразования в микропрограммах, накапливать информацию об управлении и перейти к конкретизации, когда структурная схема и микропрограмма работы ЭВМ разработана для всех команд.

2) Функциональная схема строится для каждого блока на управляемых функциональных элементах из библиотеки MaxPlus – регистрах, мультиплексорах, арифметико-логических преобразователях и простой логике, необходимой для формирования признаков и выполнения побитовых операций.

Данные и команды хранятся в модулях памяти, управление и доступ к данным в памяти – адресный.

3) Микропрограммы управления схемой оформляются в содержательном виде в **Си** и являются естественной формализацией описания содержания команд, представленных в программной модели. В дальнейшем детализируется структура микрокоманд и выполняется их упорядочение и кодирование с учетом структуры управляющего устройства. Полученные массивы двоичных кодов оформляются в загрузочные файлы.

Подготавливаются тесты для моделирования микропрограмм и с использованием средств визуализации Borland C++ создается диалоговая среда для исполнения микропрограмм и отображения результатов в терминах программной модели.

4) На основе функциональной схемы средствами MaxPLus компилируется **проект в виде** файлов загрузки в ПЛИС.

5) Верификация проекта в симуляторе MaxPlus с использованием временных диаграмм.

Большой объем неформализованной ручной работы в проекте требует верификации с учетом реального времени и схемотехники элементов библиотеки.

Диаграмма проектирования ЭВМ на основе программной модели представлена на рис 2.1.

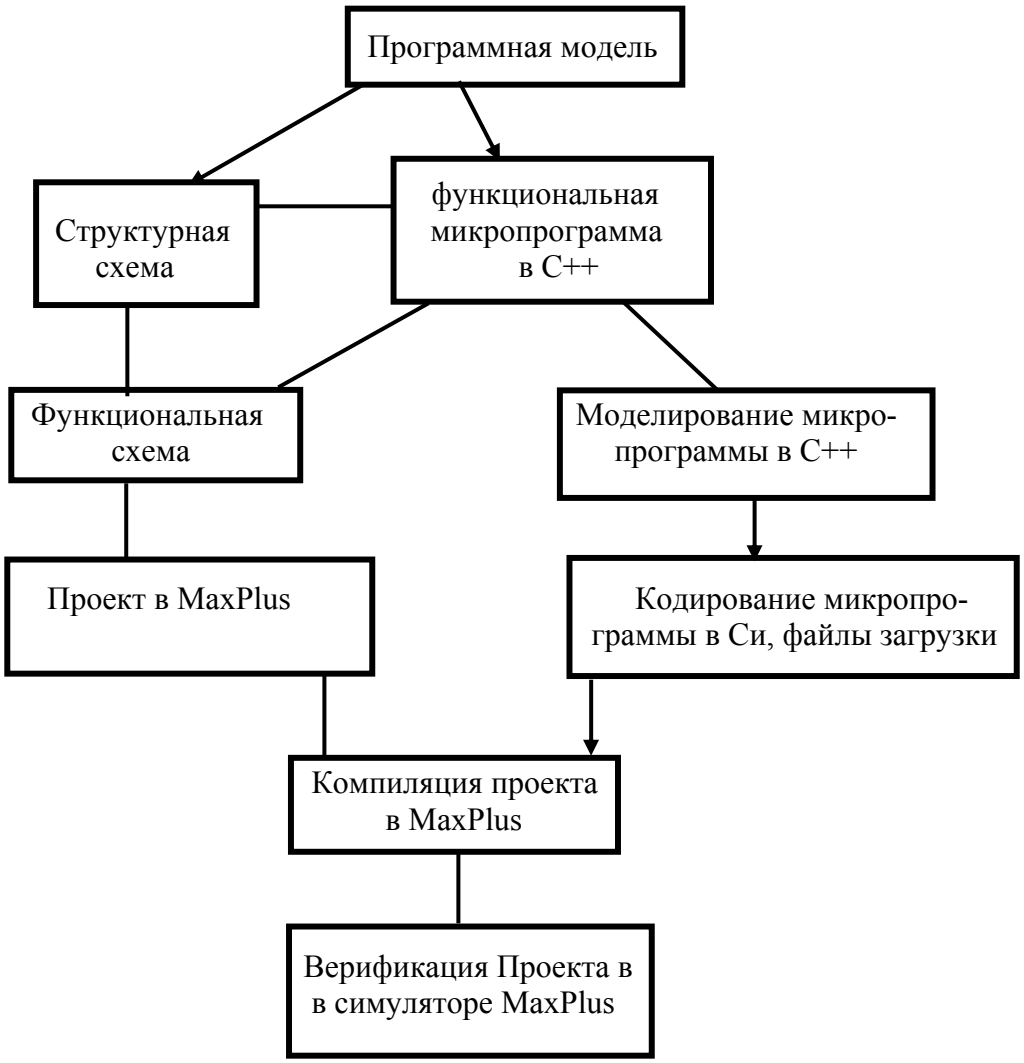


Рис 2.1. Диаграмма процесса проектирования ЭВМ

Демонстрируется общее направление выполнения проекта, этапы сохраняют связь с предыдущими – возможна коррекция на всех предыдущих уровнях.

Полученная в результате проектирования документация, демонстрируемое исполнение микропрограмм составляют содержание курсового проекта:

1) Описание заданных команд ЭВМ (см. Help Keil).
2) Структурная схема и общее описание ее работы.
3) Функциональные микропрограммы и структурные схемы блоков.
4) Функциональные схемы блоков в элементной базе MaxPlus.
5) Микропрограммы в Си и симулятор.
6) Модули кодирования микрокоманд в С++.
7) Загрузочные файлы для блоков памяти (Data, Rom, Code, Xram) в проекте MaxPlus.

8) Текстовое описание исполнения одной из команд по схемам. Необходимо для формирования технического языка, характерного для этой области.

9) Описание тестов верификации и демонстрация результатов по временным диаграммам.

III. Структура ЭВМ

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

Следующие принципы учтены при выборе структуры рис 3.1:

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** [5], 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).

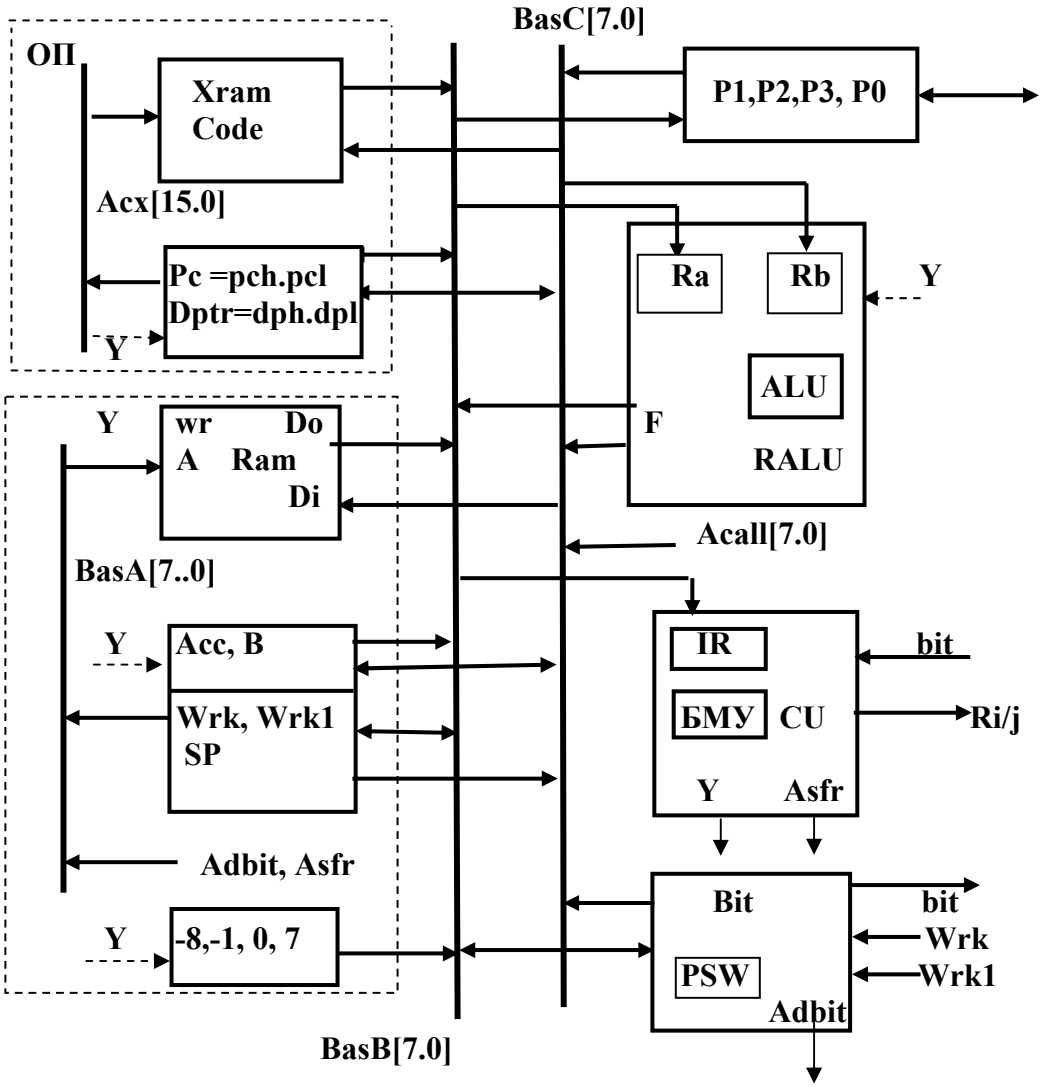


Рис 3.1. Структурная (блок) схема ЭВМ

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** – с шины **BasC.** Сигнал **записи** обозначается единицей на одном из 2^n 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.

Обзор элементной базы (только необходимой в данном проекте) сопровождается ее применением – реальным проектированием структурных схем отдельных блоков структурной схемы 3.1.

Схемотехническое проектирование – **конструктивное** в принципе, опирается на опыт и инженерную интуицию, которая также использует опыт и примеры рассуждений при выборе тех или иных решений. В проекте приводятся детали наиболее сложных при первом знакомстве схем и управления ими. Таким образом, рассмотрена практически полная структурная схема, частично представлены функциональные схемы и микропрограммы.

Одним из признаков сложных систем, помимо иерархичности, масштабности, которые преодолеваются структурированием и подробной спецификацией, является сильная связность ее структуры. По этой причине, начиная любой раздел, необходимо ссылаемся на еще не определенные схемы и устройства, но уже связанные в систему и обозначаем свойства этих схем. Это видимое нарушение логики проектирования и повторения не преодолимы и с ними приходится смириться.

IV. Проектирование в элементной базе MaxPlus

Технологии проектирования цифровых схем опираются на системы автоматизации, ориентированные на схемотехнику высокой степени интеграции – ПЛИС, СБИС.

Меню системы MaxPlus приведено в Приложении 1.

Этапы проектирования:

1. Описание схемы составляется на языках проектирования и/или в графической форме иерархически. В MaxPlus сначала формируются функциональные схемы нижнего уровня в доступных из библиотеки функциональных обозначениях. Схемы определяются как самостоятельные модули с известными входными и выходными контактами. В дальнейшем разрабатывается схема с использованием функциональных обозначений модулей.

2. В функциональных схемах используется библиотека стандартной логики – элементы малой степени интеграции (Primitivs), средней степени интеграции фирмы Techas Instruments, параметризуемые элементы высокой степени интеграции (LPM-модули).

3. При настройке параметров элементов выбирается разрядность, управление и организуются связи с использованием шин и мультиплексоров, определяются входные и выходные контакты.

Для использования унифицированных схем из библиотеки учитываются ограничения, которые влияют и на структуру схемы.

4. Выбирается тип схемы ПЛИС, для которой выполняется реализация. Схемы выбираются из библиотеки и характеризуются такими параметрами как сложность – количество логических ячеек, числом выводов, задержкой вентиля и др.

5. Компиляция включает логический синтез и распределение контактов, разбиение исходной схемы и фильтрацию. Контролируется синтаксис и ограничения микросхемы. Если ресурсов одной ПЛИС недостаточно, то логика распределяется на две и более однотипные ПЛИС.

Для каждого блока можно перечислить и привести необходимые обозначения элементарных микроопераций доступа и преобразования данных, конкретизировать функциональную схему, используя функциональные элементы библиотеки системы проектирования MaxPlus.

Проектирование в MaxPlus может опираться на языки проектирования DDL (Verilog, VHDL) и традиционное графическое схемотехническое изображение. Для проектирования таких сложных объектов, как ЭВМ, графика имеет преимущества – наглядность и масштабируемость. В изображениях элементов используется стандарт для структурных схем, применяемый в ЕС ЭВМ [2], а на уровне функциональной схемы – принятые в MaxPlus обозначения.

Для описания управления элементами используем:

1) **Функциональные микрокоманды** на содержательном языке регистровых передач (в комментариях).

2) **Функциональные микрокоманды** на языке Си (используется для моделирования работы схем). Универсальные средства языка Си позволяют определить исполняемые функциональные микрокоманды управления битами и формирования шин.

3) **Структурные микрокоманды** определяют в содержательной форме управление мультиплексорами, селекторами и другими функциональными элементами в MaxPlus и используются в Си для кодирования микропрограммы. Задаются в виде текстовых строк и определяют кодирование полей двоичной микрокоманды.

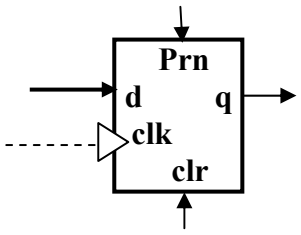
Система MaxPlus предлагает несколько библиотек элементов для проектирования цифровых схем – **Primitives, Macrofanctions(LPI), Megafanctions.**

4.1. Библиотека Primitives

Библиотека включает простые элементы, которые используются при работе с битами и организации регистров временного хранения данных.

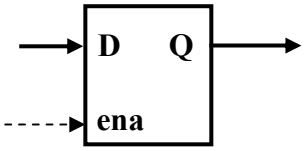
1) Flipflop & Latch Primitives – синхронизированные триггеры и защелки.

DFF – синхронный D-триггер, обычно используется для построения специальных синхронизированных регистров – преобразователей данных (счетчики, сдвигатели), как регистр состояний Конечного автомата.



Prn: q=1
clr q=0
Запись бита d по положительному (L/H) перепаду синхросигнала clk

LATCH – защелка, простой элемент памяти для хранения одного бита данных.



Запись по сигналу ena=1.

2) Input & Output Primitives/Ports – входные/выходные контакты.

На структурных схемах не имеют специальных обозначений. Обязательно присутствуют в функциональных схемах проекта в MaxPlus.

Типы контактов:

BIDIR – двунаправленные входы-выходы (контакты).

Элементы, выходы которых подключены к контактам, могут находиться в одном из трех состояний – H(1), L(0), Z(Высокий импеданс). Если контакт используется для ввода, то элементы переключаются в Z-состояние.

INPUT или **IN** – входные контакты используются только для ввода, возможно подключение внешнего блока с 3-х значным выходом, Z-состояние на входе воспринимается как неопределенное, но MaxPlus его не различает.

OUTPUT или **OUT** – выходные контакты. Предполагается, что они используются только для вывода и подключаются только к входным контактам внешних схем, принимают двоичные значения {0,1}.

3) Logic Primitives – простая логика с фиксированным числом входов.

В структурных схемах используем стандартные обозначения логики, принятые в России.

В функциональных схемах проекта MaxPlus используется Американский стандарт.

AND, OR – количество входов 2, 3,4,8,12
NOT – один вход
XOR – два входа
NOR, XNOR, NAND – инверсный выход
BAND, BOR, BXOR – инверсные входы
BNAND, BNOR, BXNOR – инверсные входы и выход

GND – земляная шина в схеме L(0)-уровень
VCC = H(1) – уровень питания

4.2. Программируемые логические модули (LPM)

Библиотека содержит **систему** функциональных элементов с программируемыми параметрами. Полезные свойства элементной базы – **многофункциональность** и выбор **двухмерной разрядности параметров q[m][n].**

Библиотека содержит большую часть необходимых для синтеза схем функциональных элементов.

Все элементы имеют графическое изображение – стандартное, применявшееся в логике ЕС ЭВМ на структурном уровне, и специальные в MaxPlus сопровождаются **окном выбора и программирования** параметров.

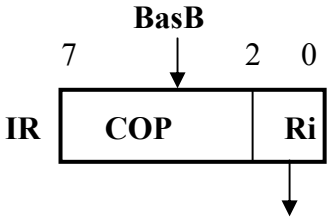
4.2.1. Регистры с прямым доступом

1. lpm_latch n-разрядный регистр-защелка данных d[n] на основе LATCH-триггеров, управлениие записью gate=1.

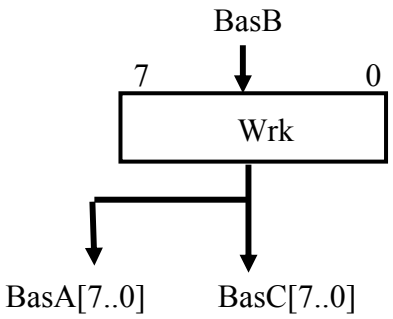
Регистр определяется форматом, который задается длиной (**разрядностью**) двоичного кода и именем. Нумерация битов обычно задается как в целых числах.

Параметры, обозначаемые отдельными полями, определяют **структуру формата регистра**. В этом случае подразумевается прямой доступ к выделенным полям кода.

IR – регистр инструкций предназначен для хранения параметров команды, Структуру регистра IR определяет загруженный в него код команды **mov a,Ri,** где Ri – трехбитный адрес регистра.



Микроопреация IR=BasB обзначает управление записью в регистр IR с шины BasB. Изображение на структурной схеме рабочего регистра **Wrk.**

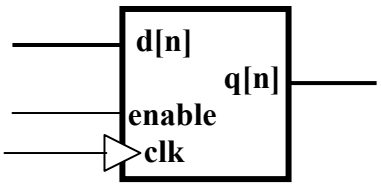


Микрооперация BasA=Wrk обозначает управление чтением регистра на шину BasA.

BasC=Wrk – управление чтением на шину BasC.

Wrk=BasB – управление записью с шины BasB.

2. lpm_ff – синхронизированный регистр на основе синхронизированных DFF-триггеров. Изображение на функциональной схеме в MaxPlus:



enable – управляющий сигнал разрешения записи по L/H фронту синхросигнала clk.

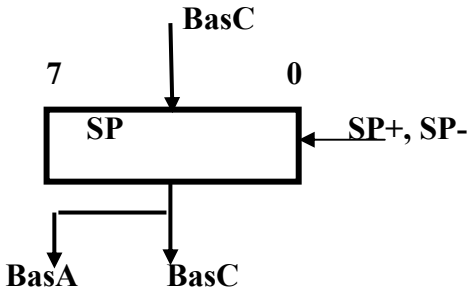
Переключение состояния и сохранение нового состояния относится к концу текущего такта относительно управляющих сигналов, установленных в начале такта. Значения сигналов изменяются с задержкой по отношению к фронту сигнала clk в следующем такте синхронизации (см. синхронизацию ЭВМ в управляющем устройстве CU).

Синхронизированная запись может быть использована для параллельной записи и чтения по двум независимым шинам функциональной микрокомандой **Wrk → ACC, ALU → Wrk**.

Синхронизированные регистры используются также в виде счетчиков и сдвигателей.

3. lpm_counter управляемый синхронизированный реверсивный двоичный n-разрядный счетчик с записью по входу q[n] ← d[n].

Изображение на структурной схеме (n=8).



Микрооперации с использованием регистра:
SP+ – синхронизированный сигнал инкремента
(SP-) – синхронизированный сигнал декремента
SP=BasC – синхронизированная запись с шины BasC
BasA=SP – выбор SP как источника 8-битового адреса на адресную шину
BasC=SP – чтение SP на шину BasC

4.2.2. Адресуемая память RAM

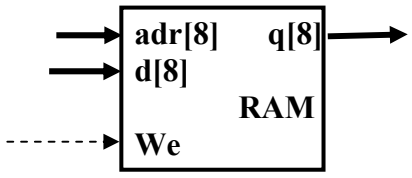
1. lpm_ram_dq память типа RAM, n-разрядное слово памяти, **запись и чтение** по m-разрядному адресу adr[m], раздельные входы записи d[n] и выхода чтения q[n]. Программируются значения **n** и **m**.

Можно рассматривать RAM как множество адресуемых регистров-защелок с интегрированным декодером адреса и управлением записью и чтением.

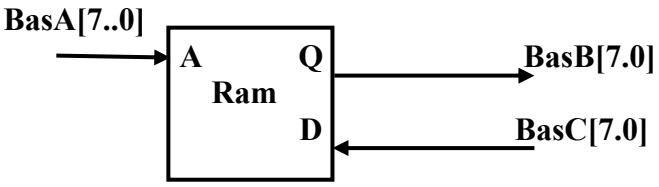
В проекте используются два типа памяти Ram:

- восьмиразрядная иерархическая память **RAM** с 8-разрядным адресом, включающая Data и SFR;
- восьмиразрядная расширенная память данных **Xram** с 16-битовым адресом.

Изображение на функциональной схеме **RAM** в MaxPlus.



Изображение в структурной схеме блока памяти RAM, управление записью и чтением неявно подразумеваются включением соответствующих шин.



В структурной схеме управляющие сигналы записи формируются **декодером (селектором)**.

Если We=1, то RAM находится в состоянии записи и значение на выходной шине Q не определено (Z-состояние).

Корректное значение на выходе формируется при установке адреса и We=0. Значение выбирается на шину сигналом BasB=Ram, обозначающим адресуемый вход мультиплексора шины BasB.

Ram объединяет в схеме 3.1. **Data**(нижние 128 байт 0-07f) и **SFR** (верхние 128 байт 080-0ff).

Микрокоманды управления обмена данными:

1) Функциональная микрокоманда чтения из Ram в ACC по адресу в Wrk:

ACC=Ram[WRK]

Структурная микрокоманда выборки из Ram и записи в ACC.

{“BasA=Wrk, BasB=Ram, Acc=BasB”} обозначает формирование кодов управления мультиплексорами BasA, BasB

2) Чтение ячейки Data в команде **mov a,Ri.**

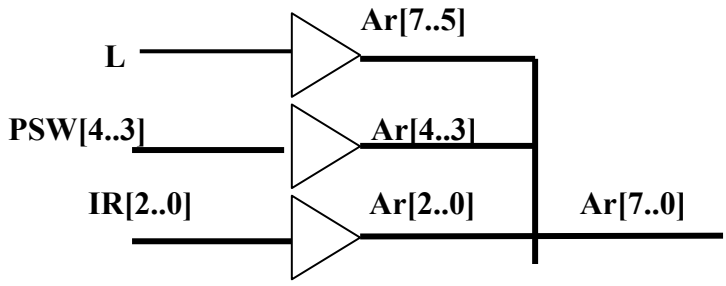
Младшие ячейки памяти Data – четыре банка регистров R0-R7. Банк выбирается 2-битным полем регистра **PSW[4.3]**. Для выборки операнда Ri в команде **add a,Ri** формируется 8-разрядный адрес Data конкатенацией полей битов **000.PSW[4.3].Ri.**

Функциональная микрокоманда чтения регистра **в Си**

Wrk=Ram[(PSW&0x18)|(IR&0x3)]

Дополнение нулями формата в Си подразумевается, но в функциональной схеме проекта MaxPlus должно быть задано явно. Конкатенация **000.PSW[4..3].IR[2..0]** формируется соединением битов на вспомогательной шине Ar, выбирается мультиплексором на адресную шину BasA и поступает на адресный вход **Ram.**

В проекте MAXPlus используется следующая схема формирования адреса с переименованием:



3) Стековая память в схеме рис.3.1 размещается в Ram и адресуется указателем вершины SP.

Пре-автоинкремент при записи в Стек

PUSH ad или Ram[++SP] ←Ram[ad]

Пост-декремент при чтении из Стека

POP ad или Ram[ad] ←Ram[SP--]

Функциональная микропрограмма записи в Стек состоит из двух микрокоманд, так как пре-инкремент не допустим (см раздел CU).

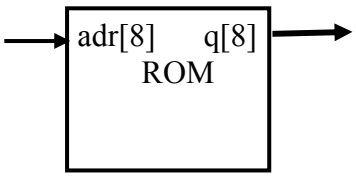
0 SP++;
1 Ram[SP]=WRK;

4.2.3. Постоянная память (ROM)

lpm_rom, в рабочем режиме допускается только чтение, запись осуществляется в проекте MaxPlus из заданного **внешнего файла данных** .

Программируемые параметры элемента – n-разрядное слово памяти q[n], m-разрядный адрес adr[m].

Изображение на функциональной схеме в MaxPlus:



ROM используется для реализации программной памяти Code, хранения микропрограммы в управляющем устройстве и декодирования управляющих сигналов.

Функциональная микрокоманда чтения первого байта команды

IR= Code[PC++];

Структурная микрокоманда управления схемой рис.3.1.

“AcX=PC, BasB=Code, IR=BasB, incPC”

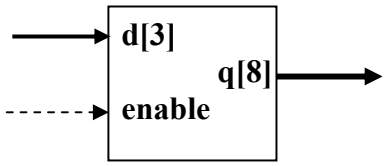
4.2.4. Управление шиной

1) lpm_decode – декодер (селектор) преобразует n-разрядный код в 2^n – разрядный унитарный код, единице равен только один выход 2^n

Программируется разрядность **n**, может быть задано управление enable.

При enable=0 на всех 2^n выходах декодера значение 0.

Изображение на функциональной схеме в MaxPLus.



В проекте селектор применяется для формирования сигналов **записи WrB** и **WrC** в элементы памяти с шин BasB и BasC по адресу приемника. При этом на вход **d[i]** поступает адрес элемента памяти, подключаемого к соответствующим шинам. На вход enable поступает общий сигнал записи **wb** или **wc**. На выходах $\mathbf{2^i}$ формируются адресуемые сигналы записи элемента, вход которого подключается к шине.

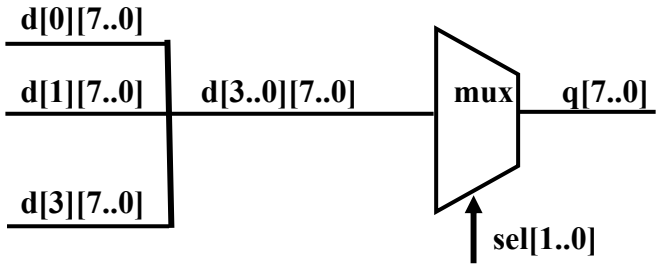
В дальнейшем сигналы оформляются в таблице 4.1. и им присваиваются конкретные адреса i и символические обозначения.

2) lpm_mux – мультиплексор шины

выбирает один из m источников n-разрядных данных по адресу sel[log(m)], адрес декодируется элементом. Выход n-разрядный.

Программируется число входов **m,** разрядность входов **n** и разрядность адреса **log(m)**.

Изображение на функциональной схеме в MaxPlus.



А) Применение мультиплексоров для управления шинами.

Мультиплексоры **BasB** или **BasC** присутствует в структурных схемах неявно .

Выход мультиплексора **формирует шину**, которая может быть использована для передачи данных к различным приемникам данным – регистрам, памяти и устройствам преобразования данных.

Тогда на структурном уровне используется изображение, в котором могут быть совмещены как мультиплексированные и адресуемые входы на шину, так и управление селектором сигнала записи. В схеме рис.3.1. управляемые шины BasB[8], BasC[8]. BasA[8], AcX[16]

В виде **полей микрокоманды управления схемой** ЭВМ оформляются адреса входов мультиплексоров BasB, BasC, BasA и адреса декодеров сигналов записи WC и WB , подключаемых к соответствующим шинам BasC и BasB. Символические обозначения сигналов выборки и записи включаются в таблицу 4.1. и используются для кодирования микрокоманд.

Таблица в дальнейшем дополняется и корректируется при детализации блоков структурной схемы и проектировании функциональной схемы в MaxPlus.

Таблица 4.1.

	4	**4**	**3**	**3**	**3**	**4**
Код	**BasB**	**BasC**	**BasA**	**ACX**	**WrC**	**WrB**
1	L(0)	P1	WRK	PC[15.0]	Ram	SP
2	Ram	ACC	Ar	WRK.WRK1	ACC	B
3	Xram	B	Adbit	Intr[15.0]	Xram	WRK
4	Code	WRK	Asfr	DPTR[15.0]	PCH	WRK1
5	ACC	WRK1	SP		DPTR	RA
6	#	PSW			RB	PSW
7	PCL	PCH	-		DPH	IR
8	DPL	DPH				PSW
9	H(0xFF)	L(0)				PCL
A	BitPSW	Acall				DPL
B	ALU(F)	SP				
C	B	H(0xFF)				
D		Bita				
E		ALU(F)				

Код=0 зарезервирован как признак отсутствия управления в поле микрокоманды. В столбцах указана разрядность соответствующих полей микрокоманды.

В дальнейшем в Си будут определены строки кодирования поля микрокоманды в виде:

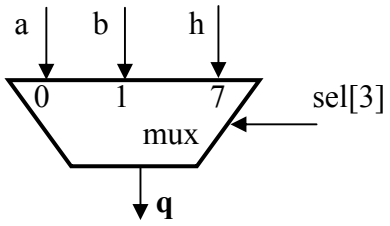
char basc[ ]=”P1, ACC, B, WRK, WRK1, PSW, PCH, DPH, L, Acall, SP, H, Bita, alu”;

1 2 3 4 5 6 7 8 9 10 11 12 13 14, где порядковый номер символа обозначает код микрооперации в поле BASC.

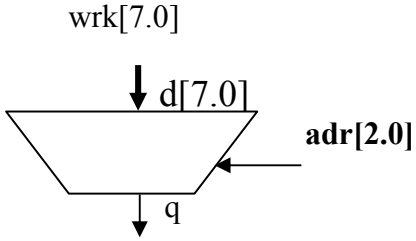
Б) Применение мультиплексора для адресации битов регистра.

В частном случае**,** мультиплексоры формируют значение бита данных при m=1. Так как адрес sel[..] всегда определен, выход мультиплексора всегда двузначный {0,1}.

Изображение в структурной схеме n=8 для лексически (по смыслу) упорядоченных битов a, b, c, , h



или при выборе упорядоченных битов 8-разрядной шины



4.2.5. Примеры синтеза блоков ЭВМ и микропрограмм управления

1. Структурная схема блока адресуемой памяти с 16-разрядным адресом:

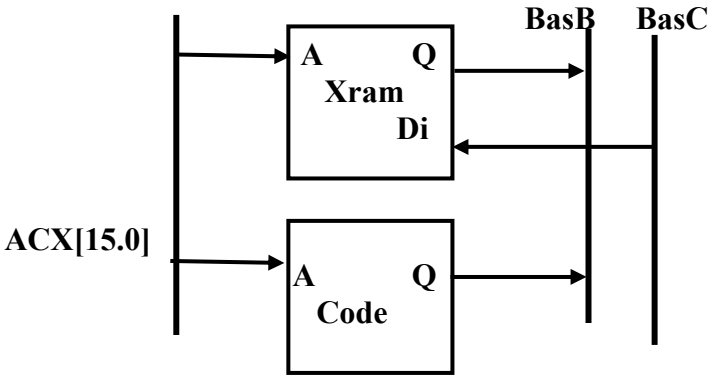


Рис.4.1. Структурная схема блока памяти с 16-разрядным адресом

Xram – память типа Ram с раздельными входами.
Code – память типа ROM.

Слово, которое по умолчанию читается из памяти, выбирается на шину BasB[7..0] мультиплексором, выбор Xram на шину BasB обозначается **BasB=Xram**, а чтение памяти Code на шину BasB – сигналом **BasB=Code**.
Функциональная микропрограмма косвенной регистровой адресации к расширенной памяти данных Xram в команде

movx a,@dptr.

0 ACC = Xram[dptr]; //запись в рабочий регистр

1 Ram[Acc] =ACC; //запись в теневой регистр

Объединить их в одну микрокоманду невозможно, так как записью управляет один селектор WrB.

Структурные микрокоманды:

0 { “ACX=DPTR, BasB=Xram, ACC=BasB,Wb”}

1 { “BasA=Acc, BasB=ACC, Ram=BasB, Wb”}

2. Структурная схема 16-битовых адресных регистров.

Блок включает два адресных регистра из числа основных регистров ЭВМ: **PC** – 16-битный программный счетчик (адрес команды), инкрементируется сигналом **PC+**.

DPTR – 16-битный указатель адреса, инкрементируется сигналом **DPTR+**

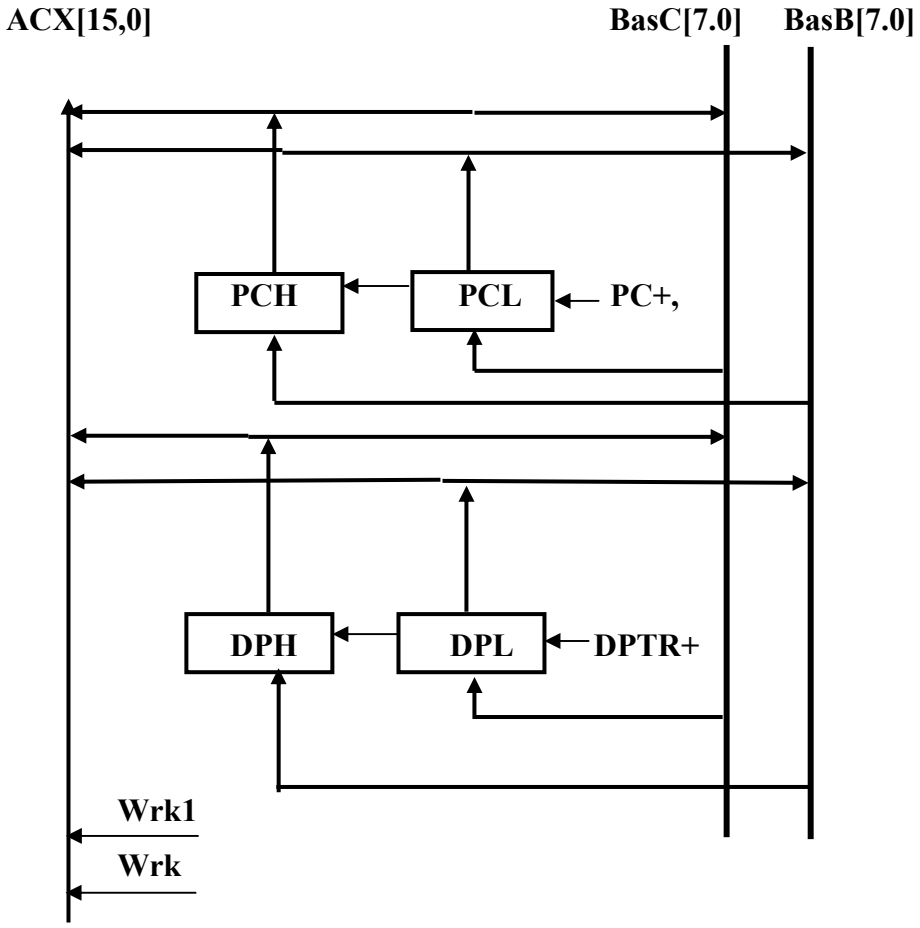


Рис. 4.2. Структурная схема блока адресных регистров

Регистры выполнены в проекте MaxPlus на 8-разрядных элементах **lpm_counter**, необходим раздельный доступ к байтам PCH и PCL при выполнении переходов **sjmp rel, jz rel**

16-битный счетчик инкрементируется по сигналу PC+, передается через 16-битовую адресную шину ACX[15..0] на адресный вход программной памяти Code для выборки команды.

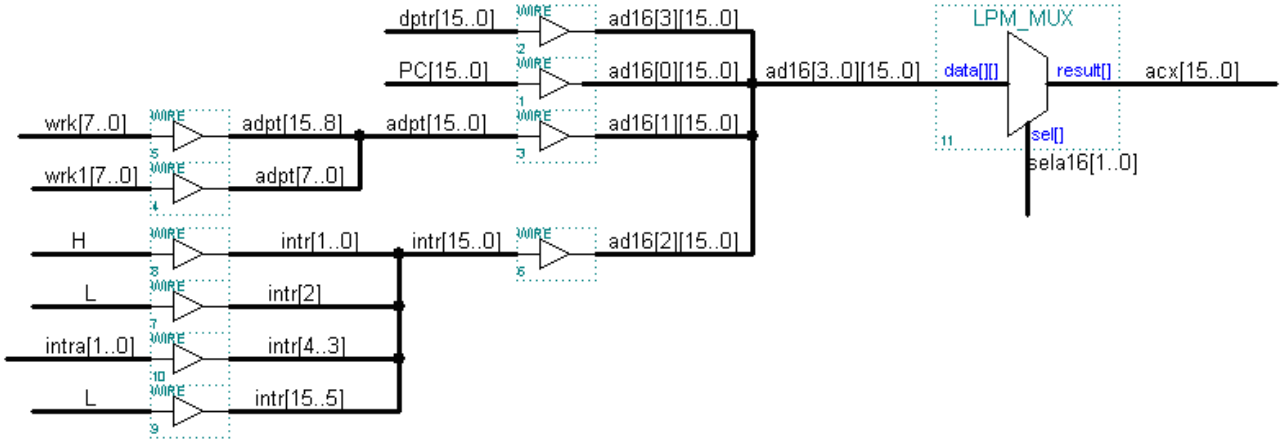
Аналогично организован 16-битный адресный регистр **DPTR с** той лишь разницей, что старший и младший байты DPH и DPL отображаются как теневые в SFR.

Управляющие сигналы формируются селектором шины ACX[15.0], в которой объединяются 8-разрядные шины ACX[15.8] и ACX[7.0] Через шины передаются старший и младший байты адреса из регистров PCH и PCL в одном такте.

16-разрядный адрес памяти Code в командах **movc a, @a +dptr,**

movc @a +pc формируется побайтно в регистрах Wrk и Wrk1 и передается через мультиплексор ACX на адресный вход памяти Code. Рабочие регистры Wrk размещаются в блоке 8-разрядных регистров.

Шина ACX[15.0] в проекте MaxPlus:



3. Микропрограмма выполнения операции mov a, ad

При обращении к SFR по адресу в команде в формате:

mov a	ad

Функциональная микропрограмма имеет вид:

0 WRK= Code(PC++); // выбрать адрес из второго байта
1 Wrk=Ram[Wrk]; // выбрать байт из Ram по адресу Wrk
2 ACC=Wrk;
3 Ram[Acc]=ACC; // записать ACC в теневой регистр в SFR

Структурные микрокоманды

0 “Acx=PC, BasB=Code, Wrk=BasB, incPC,Wb”
1 “BasA=Wrk, BasB=Ram, Wrk=BasB,Wb”
2 “BasC=Wrk, ACC=BasC,Wc”
3 “BasA=Asfr, Asfr=Acc, BasC=ACC, Ram=BasC,Wc”

4 “ACX=PC, BasB=Code,IR=BasB,Wb”

Asfr - неявная адресация, адрес **Acc** формируется блоком управления CU. В Проекте MAXPlus используется промежуточное кодирование адресов.

Табл 4.2

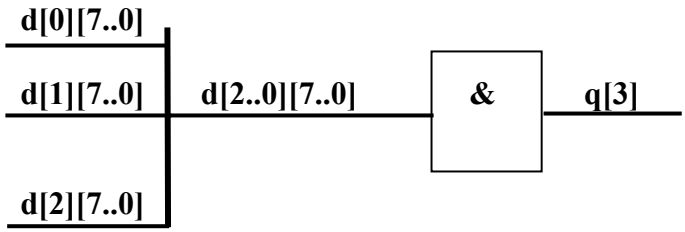
Adsfr	SFR	Адрес
0	-	0
1	ACC	0xE0
2	B	0xF0
3	SP	0x81
4	DPH	0x83
5	DPL	0x82
6	P1	0x90

3-битное поле адреса Adsfr микрокоманды определяет 7-8 битный адрес регистра в Ram. Wb, Wc – сигналы записи с шин BasB и BasC.

4.3. Вентильные схемы

lpm_and – m элементов n-входных И.

Изображение на функциональной схеме в MaxPlus элемента MUX[3][8]



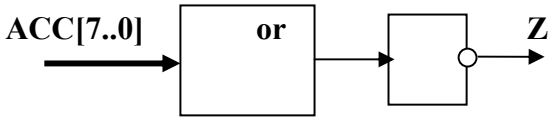
lpm_inv –m инверторов

lpm_or –m элементов n-входных ИЛИ.

В частном случае (m=1) элемент может быть использован для организации сравнения ACC с константой 0 в командах ветвления **jz rel.** На выходе схемы значение равно нулю, если ACC=0.

Для получения прямого значения признака Z подключаем к выходу простой инвертор.

Изображение на функциональной схеме блока BIT:



lpm_mult – умножитель в виде комбинационной схемы из **N** сумматоров, соединенных со сдвигом на один разряд. Сложность 8-разрядного умножителя в ПЛИС – 160 LCells.

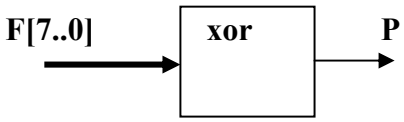
В проекте ограничимся более простым по затратам микропрограммным (алгоритмическим) решением, в котором используется уже существующая регистровая схема и АЛУ.

lpm_divide – схема деления имеет примерно вдвое большие оценки сложности и также ее заменяем микропрограммным(алгоритмическим) решением, в котором могут быть совмещены операции умножения и деления.

lpm_constant – программируемая константа, задается значение и разрядность.

lpm_xor – m элементов n-разрядных ИСКЛ-ИЛИ

Элемент может быть использован при m=1 для формирования бита четности результата операции в АЛУ.



4.4. Элементы и схемы преобразования данных (библиотека Macrofunctions)

В отдельных случаях можно также использовать библиотеки элементов, которые применялись в виде стандартных серий модулей средней степени интеграции таких, как ИК155, 555, 531 и др (прототипы элементов серии SN фирмы TI). Все эти элементы имеют фиксированные параметры – разрядность, число входов, выходов

4.4.1. Арифметико-логическое устройство (ALU)

74382 (s[2..0], a[3..0], b[3..0], cin) 4-х разрядное ALU

Изображение в проекте MaxPlus:

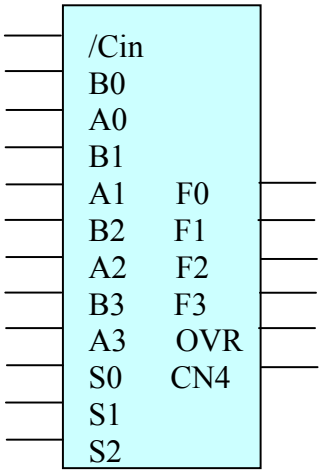


Таблица 4.3. Кодирование операций ALU

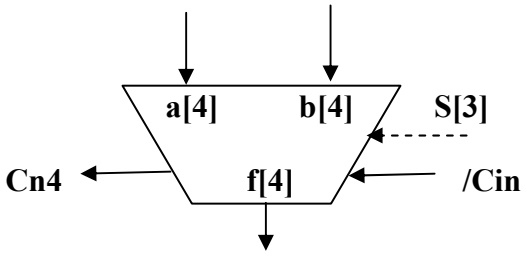
S2	S1	S0	F[3..0] (+логика)	код	ALU
L	L	L	L	0	res
L	L	H	B - A – Cn	1	suba
L	H	L	A - B – Cn	2	subb
L	H	H	A + B + Cn	3	add
H	L	L	A $ B =A | B	4	or
H	L	H	A # B = A^B	5	xor
H	H	L	A & B	6	and
H	H	H	H	7	set

S2-S0 – значения сигналов управления

F[3.0] – операции, выполняемые ALU

ALU - обозначение поля микрокоманды и символические обозначения микрокоманд

Изображение в структурной схеме**:**



Для выполнения операции заданы: операнды A и B, /Cin – входной перенос и S[2..0] – микрокоманда выбора операции АЛУ.

4.4.2. Регистровое арифметико-логическое устройство (RALU)

Структурная схема RALU выполнена на основе **74382 ALU** и регистрах-защелках.

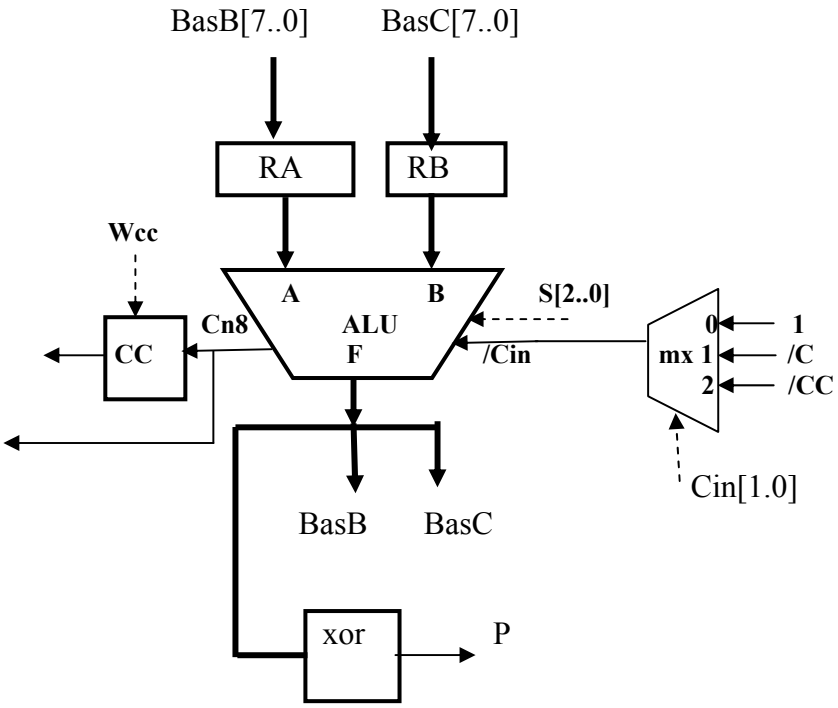


Рис.4.3. Структурная схема блока RALU

Входные данные записываются в регистры RA, RB. Операция в ALU выбирается микрокомандой **S[2..0]**. Результат операции передается по шинам BasB и BasC для записи в рабочие регистры, в частности, в аккумулятор ACC.

Входной перенос Cin формируется мультиплексором mx1. Адрес мультиплексора **Cin[1.0]** выбирает инверсное значение на входе 1, /C, /CC

Бит CC сохраняет значение выходного переноса cn8 при суммировании и вычитании. Значение Cn8 передается в блок BIT для записи в C регистра PSW и сохраняется в триггере CC.

Функциональные микрокоманды в RALU

ACC= RA-RB-C, PSWC(“subb”);

Функция **PSWC(“subb”)** формирует признаки результата

ACC= RA+RB, PSWC(“add”) ;

ACC= RA+RB+C, PSWC(“addc”) ;

4.4.3. Формирование признаков результата

В арифметических операциях в регистре PSW формируются **признаки результат**а. Для формирования признаков в Си определяется функция

PSWC(char *OP), где код **OP= “add, subb , and, or, … ”,**

1) **PSW[7]=C** ***перенос*** в операциях сложения или
заем в операции вычитания, сохраняется в командах ADD, ADDC, SUBB, в SUBB формируется прямое значение заема /C.

А) Значение **C в PSWC(“addc”)** для **Addc:**
Вычисление значения признака в Си
PSW= ((RA+RB+(PSW>>7))>=0x100) ? PSW|0x80 : PSW&0x7F ;
В проекте MAXPlus перенос PSW[7]= Cn8 рис 4.3.

Б) PSWC**(“subb”)**
Вычисление признака в Си
PSW= ((RA - RB - ((PSW)>>7))<0x100) ? PSW|0x80 : PSW&0x7F ;
В проекте MAXPlus перенос Cn8 инвертируется. Выбирается мультиплексором и записывается в PSW[7].

2) Признак переполнения **PSW[6]=OV**
А) Вычисление признака в PSWC(“addc”)
PSW= (~(RA^RB)& ((RA+RB+(PSW>>7))^PA)) ? PSW|0x40 : PSW&0xBF ;
В проекте MAXPlus в PSW записывается выход OVR ALU.

b)Вычисление признака в **PSWC(“subb”)**
PSW= (~(RA^RB)& ((RA+RB+(PSW>>7))^PA)) ? PSW|0x40 : PSW&0xBF ;

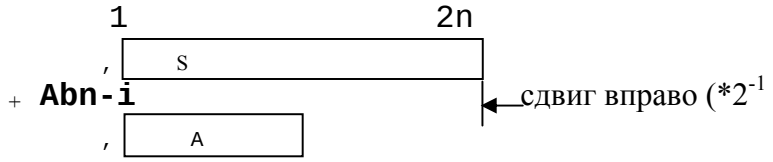
Б) Признак четности **PSW[1]= P=F[7]+F[6]+ …F[0]**.
В MAXPlus используется программируемая макрофункция **lpm_xor.**
Функция в Си
```
For(i=0;i<8;i++) P^=(ACC<<1)&0x80;
    PSW= (ACC&0x80) ? PSW|0x02 : PSW&0xED;
```

4.4.4. Схема умножения mul ab

Рекуррентная формула вычисления дробного произведения [1]

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

Схема вычисления инвариантна к формату и может быть использована как для вычисления 2n-разрядного дробного, так и 2n-разрядного целого произведения двух n-разрядных сомножителей.



Распределение регистров в команде **mul ab**

S=ACC.B – непрерывный 16-битовый регистр формирования произведения дробных чисел.

В регистр B записывается множитель.

В регистре ACC п хранится множимое. В начале операции множимое переписывается и сохраняется в рабочем регистре Wrk.

В регистре Wrk1 организуется счетчик циклов.

Основной цикл $S=\mathbf{2^{-1}(S_i+Ab_{n-i})}$ повторяется 8 раз. Если цифра множителя B[0]=0, то с учетом размещения операндов на регистрах выполняется сдвиг вправо ACC.B=sr(ACC.B), где ACC.B – соединение соответствующих регистров.

Если B[0]=1, то CC.ACC.B= sr((ACC+A).B), Формат ACC выбирается как int для сохранения переноса при сложении.

Общая схема (диаграмма) умножения:

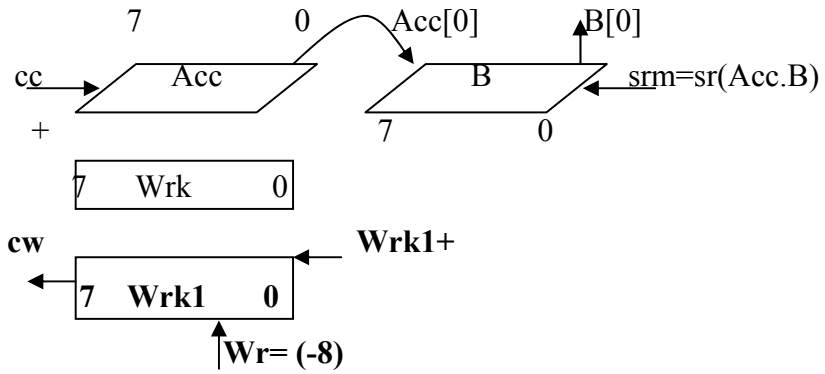


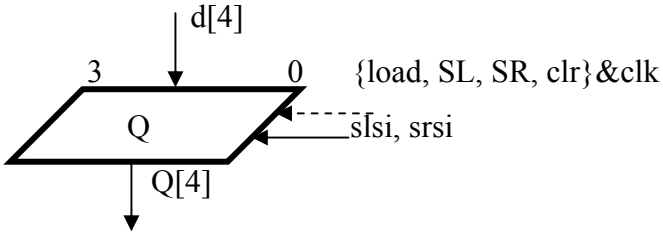
Рис.4.4. Диаграмма выполнения операции умножения

При суммировании в АЛУ может возникнуть переполнение, которое сохраняется в регистре CC и сдвигается в старший разряд ACC.

В каждом цикле (такте) инкрементируется счетчик тактов Wrk1, который в начале операции устанавливается в состояние (-8). Начальное значение произведения в ACC устанавливается в 0. Окончание операции контролируется при переполнении счетчика – сигнал cw=1. Обозначены необходимые сигналы управления srm – сдвиг вправо, Wr-запись константы (-8).

Для реализации счетчика используем **lpm_counter**, регистр Wrk-защелка, регистры ACC и B в проекте выполнены на элементах **74194 Parallel Load 4-Bit** – реверсивный сдвигающий регистр.

Изображение в структурной схеме:



Сдвиг влево **SL**: Q[4]=SL(Q.slsi), slsi – последовательный вход при сдвиге влево.

Сдвиг вправо **SR**: Q[4]=SR(srsi,Q) , srsi – последовательный вход при сдвиге вправо

Загрузка load: Q[4]=d[4]

Сброс clr: Q[4]=0

Все микрооперации выполняются по фронту L/H синхросигнала clk.

8-разрядные сдвигающие регистры ACC и B соединяем последовательно в соответствии с рассмотренной схемой умножения.

Структурная схема выполнения операции умножения

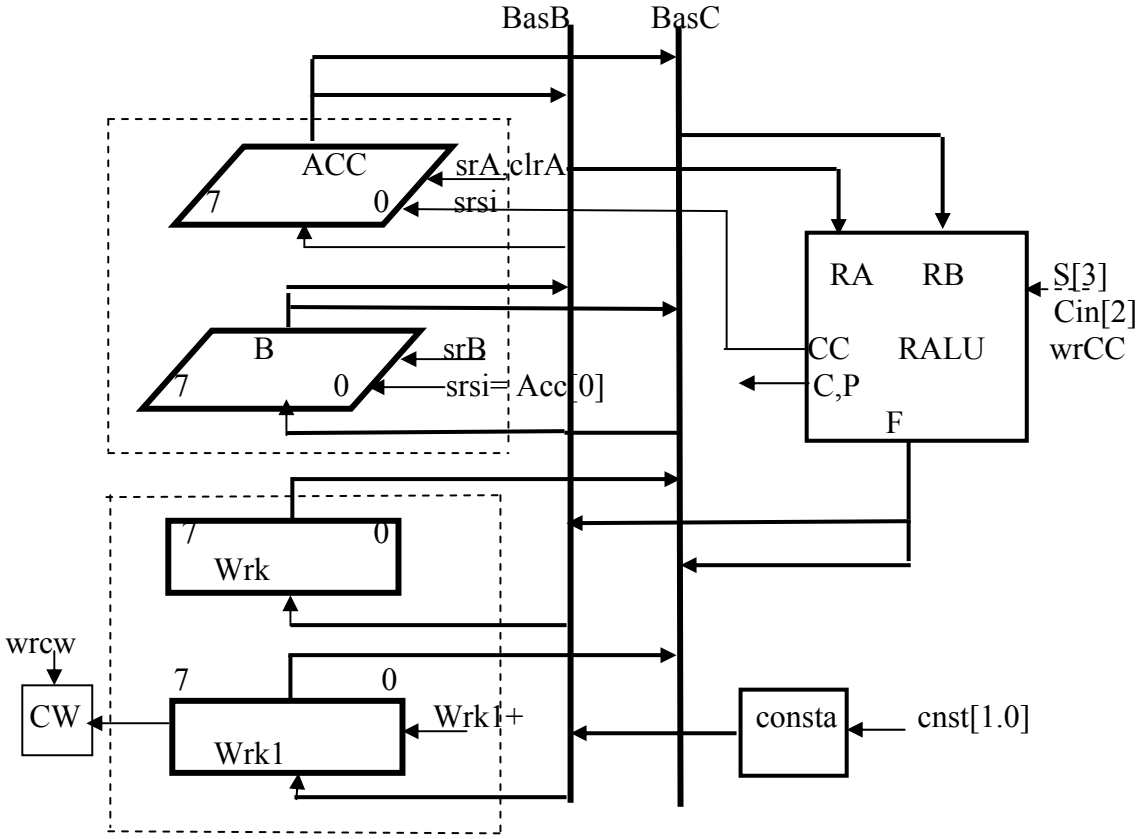


Рис.4.5. Структурная схема выполнения умножения
с блоком арифметических регистров

Функциональная микропрограмма умножения:

0 RAMM=ADC[Code[PC++]]; //выборка команды
1 Wrk=Acc; Acc=0; //размещение операндов, в Wrk множимое
2 Wrk1= -8; //Wrk1-счетчик циклов,
3 Ra=Acc; Rb=Wrk; CW=0; //выборка операндов, CW- признак завершения цикла
4 if(B[0]) {Acc=Ra+Rb; CC=C;} // B[0]-цифра множителя, CC- признак переноса
5 Sr(CC.Acc.B); Wrk1++; CW=(Wrk1>0xFF); //сдвиг произведения и множителя
6 if(неCW) goto 3 //повторение цикла, если счетчик не сбросился
7 ACC=B; B=ACC; PSW("mul"); //размещение результата по спецификации mul ab
8 Ram[Acc]=ACC; //сохранение результата в теневом регистре
9 Ram[Bb]=B; //сохранение результата в теневом регистре
10 Ram[Psw]=PSW, //Psw-адрес теневого регистра

Комментарии к отдельным микрокомандам.

1. Запись осуществляется по L/H фронту общего синхросигнала. Параллельное исполнение **Acc → Wrk, 0 → Acc**.

2. Формирование константы (-8) можно соединить с другими, обозначенными в блоке констант рис. 4.5, и выбирать из **lpm_ROM.**

Таблица 4.4. Константы в ROM

Адрес	Константа	Содержание	Мнемоника	Применение
0	00	0	Zero	сброс регистров
1	FFh	-1	Mone	
2	F8h	-8	Moct	счетчики в mul, div
3	20h		Abi	адрес бита в Data
4	07		Sp0	адрес в SP при сбросе

5. Sr(CC.Acc.B), CW.Wrk1++; биты переполнения CC, CW рассматриваются как расширение формата операндов.

Параллельно выполняются две микрооперации – сдвиг и инкремент Wrk.

Сдвиг вправо регистров ACC и B разрешен общим сигналом управления. При этом вход последовательного сдвига B соединен жестко с выходом A[0].

Переполнение сложения в АЛУ в предыдущей микрокоманде из CC передается на вход последовательного сдвига ACC.

Используется триггер CW для сохранения выходного переноса CW при добавлении единицы к Wrk1.

7. Передачи выполняются параллельно по двум шинам BasB и BasC, новые состояния регистров устанавливаются по фронту синхросигнала clk.

В PSW бит C=0, признак OV=1, если B!=0. определяется P, остальные биты не изменяются.

Функция **PSWC("mul")** формирует эти признаки в Си, в MaxPlus проекте PSW устанавливаются схемами блока обработки битов **BIT**.

4.4.5. Схема выполнения беззнакового деления div ab

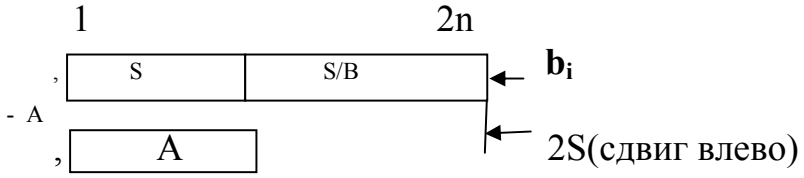
ACC содержит 8-разрядное делимое, регистр B - делитель. Частное формируется в ACC, остаток в B. Признаки C и OV в PSW сбрасываются. Возможное переполнение при делении на 0 не контролируется.

Вычисления по следующей рекуррентной формуле [1] деления без восстановления остатка для дробных чисел – сначала определяется знак разности **$2S_i$-A**, остаток сохраняется, если знак положительный. S – делимое, A – делитель, $B=0.b_1b_2..bn$ – двоичное частное

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

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

Схема вычисления частного без восстановления остатка:



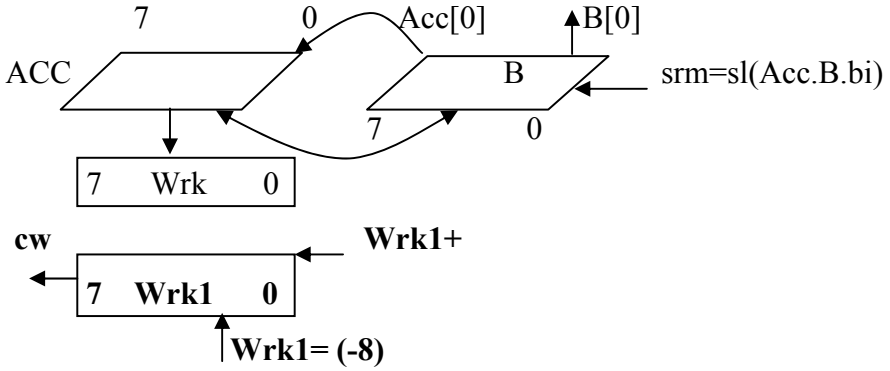
Частное B совмещается с младшими разрядами 2n-разрядного делимого S.

Схема деления объединяется со схемой операции умножения.

В регистре ACC при выборке команды хранится делимое, в регистре B – делитель. В начале операции осуществляется обмен данными **Acc ⟷ B**, делимое переписывается в регистр B, после этого делитель из Acc сохраняется в рабочем регистре Wrk и ACC сбрасывается. В регистре Wrk1 организуется счетчик циклов

S=00.ACC – начальное состояние 16-разрядного регистра дробного делимого. В регистре Wrk записывается делитель.

Общая схема (диаграмма) деления в проекте после размещения операндов на регистрах:



Если остаток **S_{i+1}=$2S_i$-A** положительный, то цифра bi=1 записывается при сдвиге остатка влево в младший разряд регистра B. Если остаток отрицательный, то сохраняется предыдущее значение положительного остатка и bi=0. Ес-

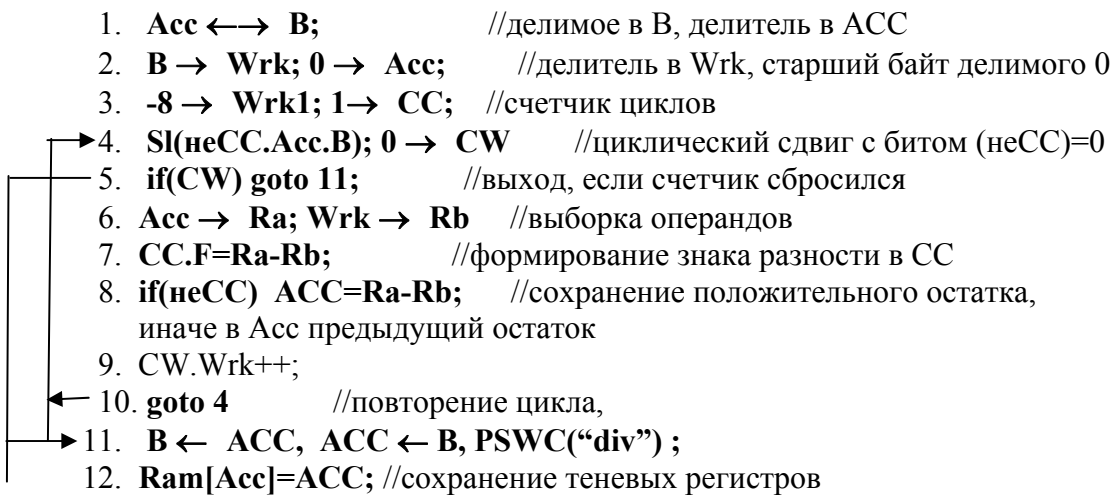
ли остаток всегда сохраняем в ACC, то его предыдущее положительное значение можно повторно прочитать в регистре ACC.

Перенос **Cn8** в схеме RALU рис.4.3. – выход сумматора, при вычитании (суммировании в дополнительных кодах) равен инверсии заема. Для контроля знака также необходимо учитывать, что операция вычитания выполняются при сдвинутом влево остатке и теряется старший бит, который также необходимо учитывать для контроля знака. Сохраняем этот бит в триггере CC.

Признак отрицательного результата операции в CC с учетом сдвига формируем схемой

/CC&CN8

Функциональная микропрограмма (алгоритм) выполнения деления:



Комментарии к микрокомандам:

1) Обмен данными Acc ⟷ B в проекте MaxPlus возможен, так как выполняется по разным шинам и фиксируется по фронту синхросигнала.

7) Вычитание без сохранения отрицательного остатка, формируется знак разности в CC. В схеме РАЛУ (рис.4.3) добавляется схема контроля знака разности и сдвиг вправо.

8) Повторное вычитание с сохранением остатка при положительной разности, при отрицательном остатке предыдущий остаток сохраняется в ACC.

V. Управляющее устройство (CU)

Схема формирования управляющих сигналов **CU** реализует микропрограмму управления выполнением конкретных команд, обеспечивает выборку и формирование микрокоманд, декодирование команд и микрокоманд. Предполагаем и ограничиваемся многотактным исполнением микропрограммы, конвейерное исполнение может быть предметом дальнейшего усовершенствования CU и существенного изменения схемы ЭВМ.

5.1. Синхронизация схем ЭВМ

В большинстве существующие ЭВМ строятся на основе синхронных схем, в которых учитывается **общий синхросигнал**.

При этом изменение состояния двухтактных регистров при записи синхронизируется фронтом синхросигнала Clk. Период синхросигнала называют **тактом** работы CPU.

Для записи в память и регистры-защелки используется синхронизированный управляющий сигнал.

Чтение адресуемой памяти при записи в обычную однопортовую память запрещено, так как используется один общий альтернативный сигнал записи/чтения Wr (0-чтение, 1-запись), постоянный в течение такта.

Управляющие сигналы Y изменяются на входах функциональных элементов с задержкой **(t)** относительно фронта синхросигнала Clk, но сохраняются в момент переключения управляющих сигналов Y. Новые значения Y сохраняются в течение всего последующего такта в соответствии с диаграммой рис. 5.1.

Временная диаграмма одного такта работы ЭВМ приведена на рис 5.1.

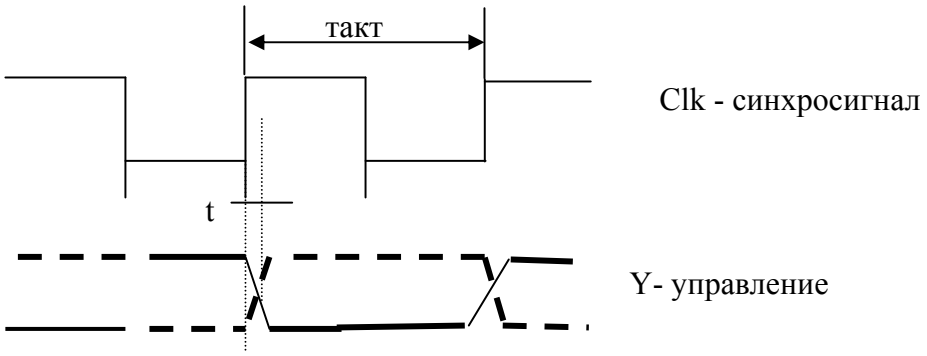


Рис.5.1. Временная диаграмма синхронизации.

Микрокоманда формируется CU с задержкой t и исполняется в течение такта: начинается по фронту, в котором изменяется состояние регистров, и завершается по фронту, который фиксирует новое состояние.

Время переключения логики, формирующей новые состояния регистров, определяет длительность такта.

Порядок выполнения микрокоманд в **микропрограмме** зависит от состояния регистров и признаков, формируемых в АЛУ.

Микрооперации счета, сдвига могут быть выполнены в регистрах-счетчиках или сдвигателях на основе синхронизированных регистров. Применение таких элементов позволяет параллельно (с совмещением) выполнять несколько микроопераций в одном такте. Например, **IR=Code(PC+)** – чтение первого байта команды и пост-автоинкремент программного счетчика.

Однако пре-инкремент или пре-декремент в этой синхронизации не возможны, так как относятся к пост-инкременту (пост-декременту) в предыдущем такте.

5.2. Блок микропрограммного управления

Схема CU может быть организована на основе **адресного счетчика микрокоманд**, **микропрограммной памяти и декодеров (Блок микропрограммного управления – БМУ[4]).** Микропрограмма кодируется как последовательность двоичных кодов микрокоманд (трасса) и хранится в ПЗУ.

Структурная схема блока CU приведена на рис. 5.2.

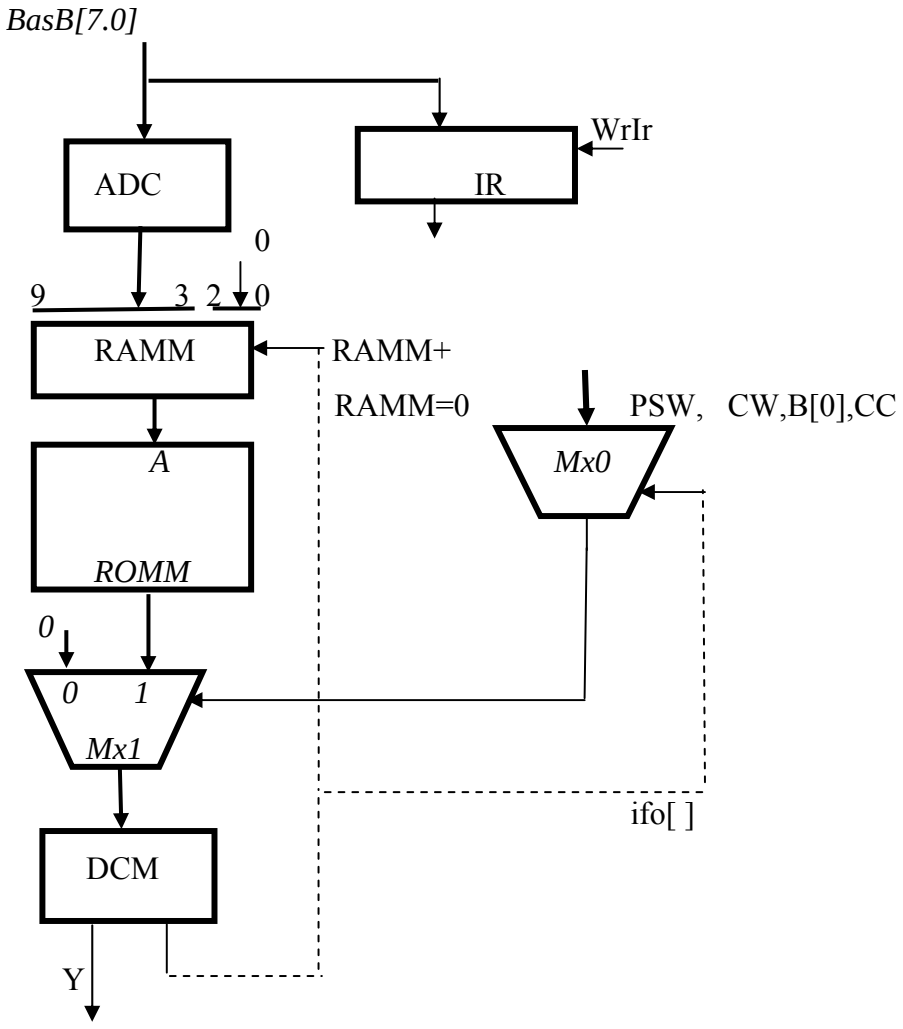


Рис.5.2. Структурная схема микропрограммного управления CU

IR – регистр-защелка команд;
ADC – адресный декодер;
RAMM – регистр адреса микрокоманд;

ROMM – микропрограммное ПЗУ с закодированными функциональными микрокомандами;

Mx0 – мультиплексор условий;

Mx1 – мультиплексор микрокода позволяет сформировать значение нуль, если выбранное мультиплексором Mx[ifo] значение бита нуль;

DCM – первая ступень декодирования на основе ПЗУ, которое содержит двоичные коды структурных микрокоманд.

Первый байт команды из программной памяти Code поступает в адресный преобразователь ADC и фиксируется по синхросигналу в регистре IR. Ряд команд содержат в первом байте параметры – Ri, Rj, A_{10}-A8, которые сохраняются в IR.

Количество различимых типов команд в системе команд MCS51 всего 102 и $]\log_2 102[$=7-разрядность минимального **различающего кода команды**

Декодер ADC выполнен в ПЗУ, где для всех 256 возможных значений первого байта сохраняется семиразрядный двоичный код. При выборке код дополняется тремя нулевыми битами и фиксируется на 10-разрядном регистре адреса микрокоманд RAMM по синхросигналу Clk. Этот адрес является началом линейной трассы **кодов функциональных микрокоманд** микропрограммы в ROMM и соответствуют выбранному коду команды. Далее микрокоманды выбираются последовательно по счетчику (не более 8 микрокоманд).

Количество различимых функциональных микрокоманд существенно меньше полного их перебора, определяемого длиной и числом кодов полей структурной микрокоманды. Этот факт позволяет выделить для хранения **двоично-кодированных структурных микрокоманд** быструю **нанопамять** DCM с широким словом (до 32 и более бит). Так в предлагаемых проектах количество различимых микрокоманд (структурных или функциональных) менее 32 и для их адресации в DCM достаточно 5 битов и память ROMM хранит микропрограммы из 5 битовые кодов микрокоманд.

Структурные микрокоманды, выбранные из DCM, частично декодируются в отдельные управляющие сигналы (ACC+, SP+, SP-, slA, slB и др.), а поля адресов мультиплексоров и селекторов декодируются элементами функциональной схемы проекта.

В течение такта происходит выборка кода функциональной микрокоманды из ROMM, выборка структурной микрокоманды из DCM, декодирование и исполнение микрокоманды. Таким образом, существенная задержка при выборке и исполнении микрокоманды определяет длительность такта.

Возможно сокращение длительности такта переходом в **микроконвейерный** (совмещенный) режим исполнения и выборки микрокоманды с дополнительным регистром микрокоманд.

Первая микрокоманда **IR=Code[PC++],** общая для всех операций, формируется ROMM по адресу RAMM=0. Эта микрокоманда выбирает текущую команду по счетчику PC и сохраняет адрес начала **трассы**

микрокоманд в RAMM[9.0]. Микрокоманда считывается по адресу RAMM из постоянной памяти микрокоманд ROMM.

В микропрограммах используются **условные микрокоманды** – указывается значение условия, при котором выполняются заданные в микрокоманде операции. Если условие не выполняется, то микрокоманда пропускается.

На схеме рис. 5.2 условие из PSW и отдельных битов выбирается мультиплексором **mx0** по адресу **ifo.**

Если выбранное условие **False** (0), то мультиплексор **mx1** выбирает структурную микрокоманду 0, что эквивалентно пропуску такта. Таким образом, исключаются условные переходы, микропрограмма всегда линейная трасса и микрокоманды выбираются по счетчику.

На этой стадии после выбора общей схемы БМУ возникает вопрос о размещении первой микрокоманды **IR=Code(PC++)** выборки команды.

Если эту микрокоманду разместить как последнюю, то она всегда общая во всех микропрограммах и исполнение команды всегда начинается с декодирования, но микрокоманда будет многократно повторяться, так как расположена по разным адресам.

В целом предпочтительно размещать эту микрокоманду как первую в общей микропрограмме по фиксированному адресу RAMM=0, что упрощает начальный запуск и позволяет выделить одну общую фазу при исполнении команд **Instraction Fatch (IF),** количество тактов для выполнения команды увеличивается на один и микропрограмма стартует с первой (нулевой) микрокоманды трассы. Следовательно, последняя микрокоманда в трассе должна сбрасывать регистр RAMM=0 для выборки и исполнения микрокоманды **IR=Code[PC++]**.

5.3. Реализация конечных автоматов в CU

Многотактные команды **mul ab, div ab** используют вспомогательные счетчики тактов, анализируют состояния динамических признаков. При этом требуются безусловные и условные переходы в микропрограммах с явным значением адреса одного из переходов. Это исключает возможность построения линейных микропрограмм.

Альтернативой БМУ с хранимой микропрограммой является автоматный подход к синтезу управляющего блока в виде **жесткой схемы [4],** в которой допустимы любые переходы между микрокомандами.

Рассмотрим функциональную микропрограмму умножения.

```
0 RAMM=ADC[Code[PC++]];
1 Wrk=Acc;  Acc=0;
2 Wrk1= -8;
3 Ra=Acc; Rb=Wrk; CW=0;
4 if(B[0]) {Acc=Ra+Rb; CC=C;}
5 Sr(CC.Acc.B); Wrk1++; CW=(Wrk1>0xFF);
```

```
6 if(неCW) goto 3
7 ACC=B;  B=ACC; PSW(“mul”);
8 Ram[Acc]=ACC;
9 Ram[Bb]=B;
10  Ram[Psw]=PSW,
```

Микропрограмму интерпретируем конечным автоматом (КА). Для запуска КА формируется признак A=1; При этом выделяем часть микропрограммы, которую относим к БМУ и часть относим к схеме КА.

БМУ:

```
RAMM=ADC[Code[PC++]];
Wrk=Acc;  Acc=0;
Wrk1=-8;  A=1;
ACC=B; B= ACC; PSW(mul);
Ram[Acc]=ACC;
Ram[Bb]=B;
Ram[Psw]=PSW,
```

Автомат

```
0     if(/A) goto 0   //микрокоманда ожидания запуска
1.  Ra=Acc; Rb=Wrk; CW=0;  // выборка операндов
2   if(B[0])  goto 3  else   goto 4;
3  ACC=Ra+Rb; CC=C;
4.   Sr(Acc.B); Wrk1++; CW=(Wrk1>0xFF);;
5   if(неCW) goto 1
6.   A=0; goto 0;  //возврат к БМУ и завершение работы КА
```

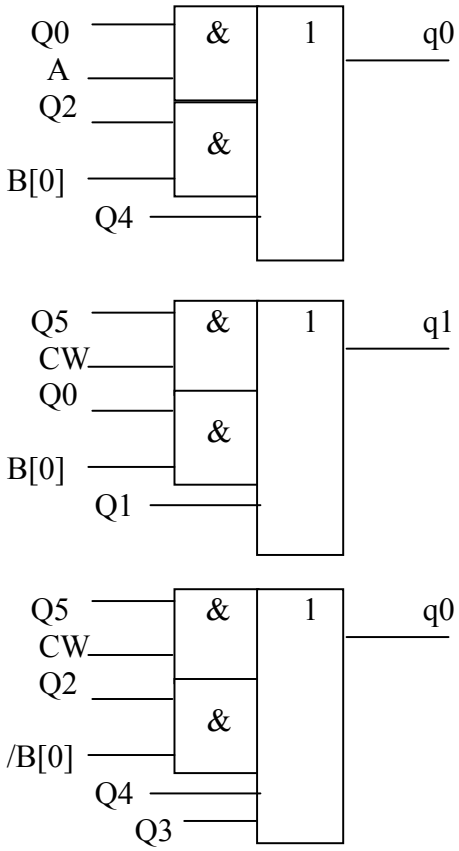
Каждая i-ая микрокоманда КА обозначается как состояние Qi. Состояния нумеруются порядковыми номерами микрокоманд, Q0-начальное и Q6-конечное. Каждому состоянию Qi ставим в соответствие код функциональной микрокоманды из общего списка микрокоманд в DCM. Система уравнений переходов и адресации микрокоманд в КА с естественным кодированием состояний:

```
q0= (Q0&A) v (Q2 & B[0]) v Q4
q1= Q1 v   Q2 & B[0]) v  Q5 & CW
q2= Q3 v Q2 & /B[0] v Q4 v Q5 & CW
```

Каждое состояние формирует код функциональной микрокоманды или адрес соответствующей структурной микрокоманды в DCM.

```
0     0
1.    Ra=Acc; Rb=Wrk; CW=0;  // выборка операндов
2     0
3     ACC=Ra+Rb; CC=C;
4.    Sr(Acc.B); Wrk1++; CW=(Wrk1>0xFF);;
5     0
6.    A=0;
```

Функциональная схема формирования битов следующего состояния qi в MaxPlus:



Структурная схема управляющего автомата, согласованная со схемой CU, на элементах MaxPlus приведена на рис 5.3.

Схема перехода qi – поразрядная установка битов регистра состояний Q. Если в качестве регистра состояний используется **lpm_ff**, то схемой поразрядно формируется код следующего состояния и записывается в Q синхросигналом Clk.

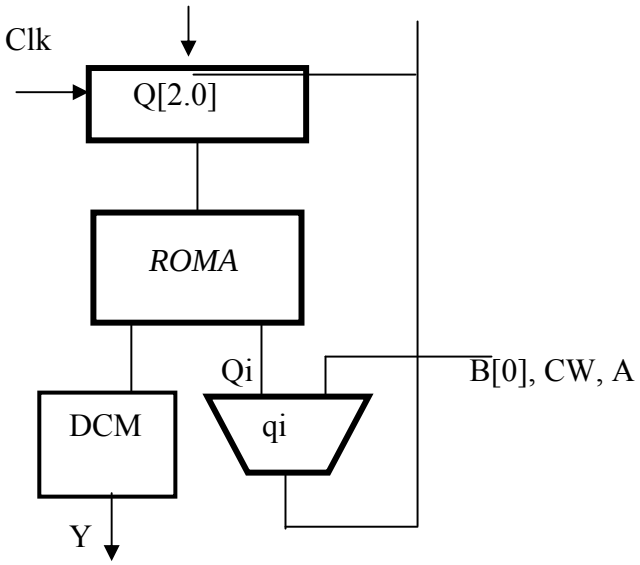


Рис 5.3. Структурная схема КА

Q[2.0] – синхронизированный регистр состояний.

ROMA – постоянная память КА для каждого кода состояния хранит двоичный код функциональной микрокоманды (адрес структурной микрокоманды в DCM) и декодированный признак текущего состояния Qi.

qi – схема формирования значений битов кода следующего состояния. Схема CU, совмещающая БМУ и КА, реализует микропрограмму умножения рис 5.4:

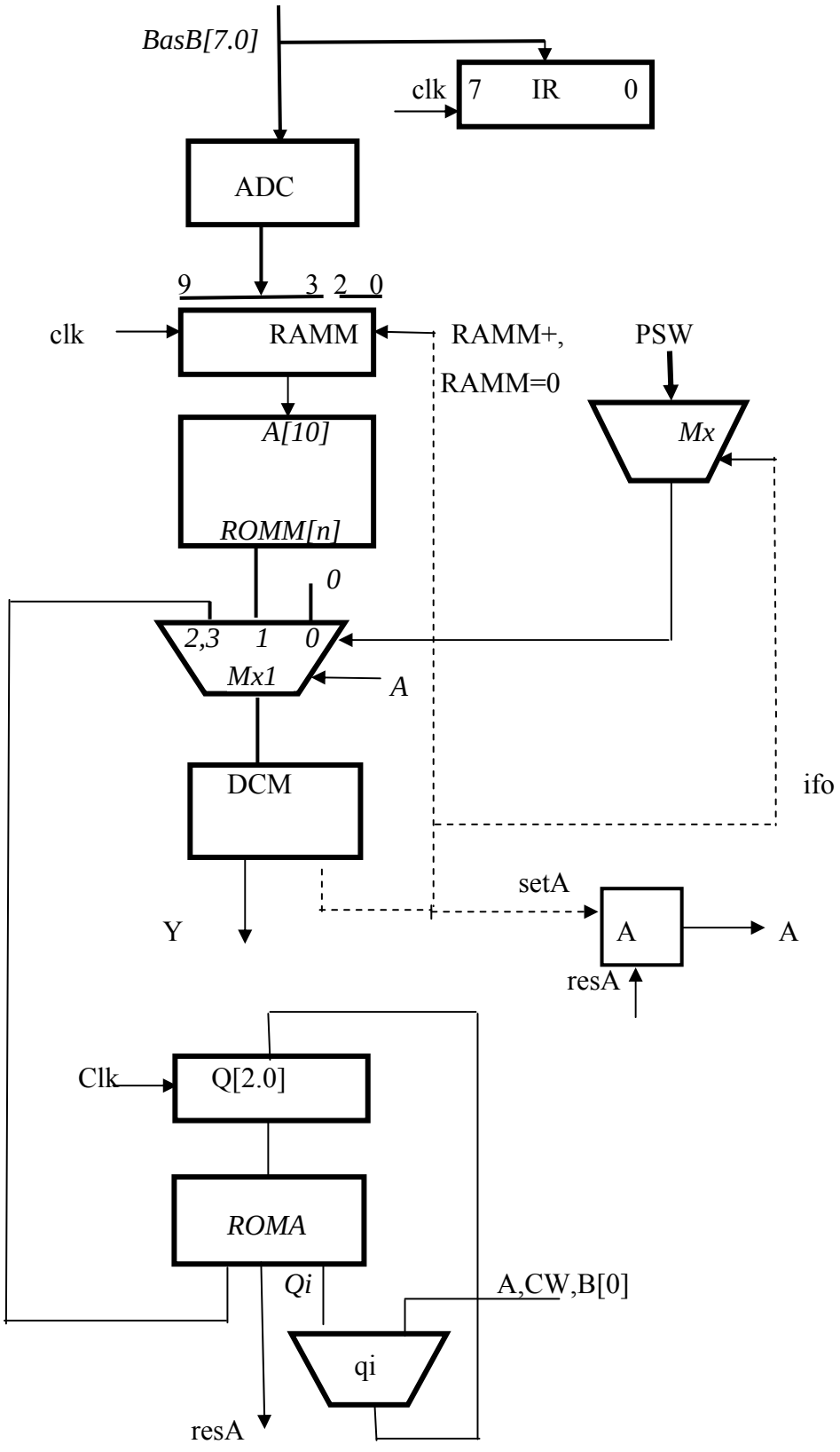


Рис.5.4. Совмещенная схема БМУ и КА в CU

Функциональная схема **qi** в системе элементов библиотеки **Primitives** реализует систему уравнений для функции перехода в следующее состояние. ROM формирует код функциональной микрокоманды, поступающий с выхода мультиплексора Mx1 на вход общего ПЗУ **DCM**. Коды адресов Mx1 представим таблицей.

Таблица 5.1

A	**Mx**	**адрес**	**значение на выходе Mx1**
0	0	0	значение 0
0	1	1	Микрокоманда из ROMM
1	0	2	микрокоманда КА
1	1	3	микрокоманда КА

ifo[3] – поле условий в микрокоманде контролирует признаки результата. Если условие истинно, то микрокоманда выбирается и выполняется.

Табл.5.2.

код	условие	обозначение
0	0	jam
1	C	jc
2	/C	jnc
3	W[7]	jw7
4	/W[7]	jnw7
5	CC	jcc
6	/CC	jncc

Завершаем демонстрацию проектирования схемы КА обзором методики проектирования:

1. Выделение в микропрограмме операции фрагмента, выполняемого в КА, и разделение ее на блоки, выполняемые в БМУ и КА. Использование триггера A1,A2.. для управления переключением. Каждую новую операцию идентифицирует свой бит A1,A2.... Микропрограмма преобразуется с обозначением полной системы переходов (Преобразования могут быть выполнены с предварительным графическим представлением микропрограммы в виде блок-схемы).

2.Кодирование состояний автомата и формирование полной системы дискретных функций перехода.

Для автомата указывается полное множество переходов – как для прямых, так и инверсных условий. Схемы автоматов A1,A2,..объединяются в общем ROMA, где различаются адресами, и по ИЛИ объединяются выходы

схем bi. Регистр состояний дополняется битами A1,A2... При этом биты-признаки состояний Qij различаются в разных j=1,2,.. конечных автоматах.

3. Система уравнений формирования битов qji следующего состояния.

4. Кодирование совмещенных микрокоманд, формируемых ROMA и содержащих код функциональной микрокоманды и биты Qi, идентифицирующие следующее состояние КА.

5. Построить функциональную схему, реализующую систему уравнений qji.

VI. Примеры схем и микропрограмм

6.1. Команда ветвления JZ rel

Содержание команды - if(ACC=0) goto rel
Адрес следующей команды формируется алгебраическим сложением PC=PC+rel8, если ACC=0.

	Функциональная микропрограмма	**Структурная микропрограмма**
0	**{WRK=Code(PC); RAMM++;}**	**{"ACX=PC,BasB=Code, WRK=BasB, WrB "}**
1	**if(ACC!=0) PC++ ; RAMM=0;**	**{"ifo=/Z, incPC"}**
2	**{RA=Wrk; RB=PCL RAMM++;}**	**{" BasB=WRK, RA=BasB , BasC=PCL,RB=BasC,WrB,WrC"}**
3	**{PCL=RA+RB; RAMM++; }**	**{"mop=add, BasB=ALU, PCL=BasB, WrB"}**
4	**{ RB=PCH; RA=0; RAMM++;}**	**{"BasC=PCH, RB=BasC, BasB=L, RA=BasB, WrC,WrB"}**
5	**{ if(Wrk&0x80) RA= -1 ; RAMM++; }**	**{"ifo=Wrk7. BasB=-1, RA=BasB, WrB"}**
6	**{PCH=RA+RB+CC; RAMM=0;}**	**{ " mop=add, BasB=F, PCH=BasB, C0=CC, WrB"}**

6.2. Команда десятичной коррекции DA A

Команда выполняется после двоичного сложения или вычитания. Описание из Keil Help.

If (A_{3-0} >9) or (AC=1)
A=A+6
If (A_{7-4} >9) or (C=1)
A=A+60h

Схема в MAXPlus:

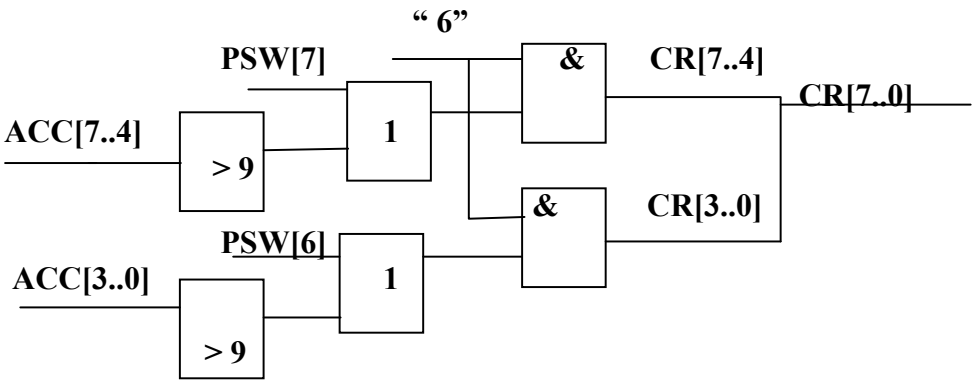


Рис.6.1. Схема десятичной коррекции

Функция (A[7.4] >9) = A[7] &(A[6] v A[5])
Функциональная микропрограмма коррекции

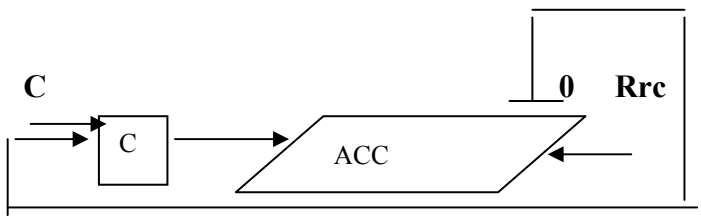
```
0   PA=ACC, PB= CR( ); RAMM++;   //CR(  ) – функция C++
//моделирует схему формирования кода коррекции
 1   ACC =PA+PB ; RAMM++ ;
 2   RAM[Acc]=ACC ; RAMM=0 ;
```

6.3. Команда циклического сдвига RRC A – циклический сдвиг расширения C.ACC влево

Функциональная микрокоманда:

```
( int)ACC=(int)ACC|((PSW&0x80)<<1); PSWC(“rrc”);  ACC=ACC>>1;
  {  ……..
     case “rrc”:  PSW=(ACC&0x80) ? PSW|0x80 : PSW&0x7F;
    ……..
   }
```

Структурная схема операции в MaxPlus:

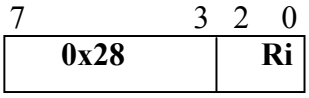


Rrc- управляющий сигнал сдвига вправо и записи PSW[7].ACC

6.4. Арифметические команды

Форматы команд:

add a, Ri



add a, #d

Ri – поле адреса регистра
#d – байт непосредственной константы

Микропрограмма выполнения арифметических команд в Си

```
Char    IR=0,     // регистр команд
         Wrk,       //рабочий регистр
         ACC,        //аккумулятор
       Ram[256],   //память Ram
       Code[0x7ff],
        ADC[256];  //декодер инструкций
 Int PC=0;//программный счетчик-

//кодирование команд в ADC блока управления CU.
For(i=0x28;i<0x28;i++)   ADC[i]=0x11;  //add a,R0-R7
 ADC[0x24]=0x12;                        //add a.#d
 ADC[0]=0x10;

Main()
{
  while(ПУСК)   //цикл исполнения команд
  {
      RAMM=0; IR=Code[PC++];  //выборка команды – первая микрокоманда цикла
       switch(ADC[IR])   //адрес начала трассы микропрограммы;
       {
       case 0x11:       //add a,ri
           switch(step)
           { case 0: RA=ACC, RB= Ram[(PSW&0x18)|(IR&0x7)];RAMM++; break;
             case 1: ACC=RA+RB, PSWC(Add); RAMM++; break;
             case 2: Ram[Acc]=ACC; RAMM++; break;
             case 3: Ram[Psw]=PSW; RAMM=0;  break;
           }
                  break;
       case:  0x12    //add a,#d
           switch(step)
             {
             case 0: RA=ACC, RB=Code[PC++];RAMM++; break;
               case 1: ACC=RA+RB, PSWC(Add); RAMM++; break;
             case 2: Ram[Acc]=ACC; RAMM++; break;
             case 2: Ram[Psw]=PSW; RAMM=0;  break;
             }
```

```
            break;
    }
```

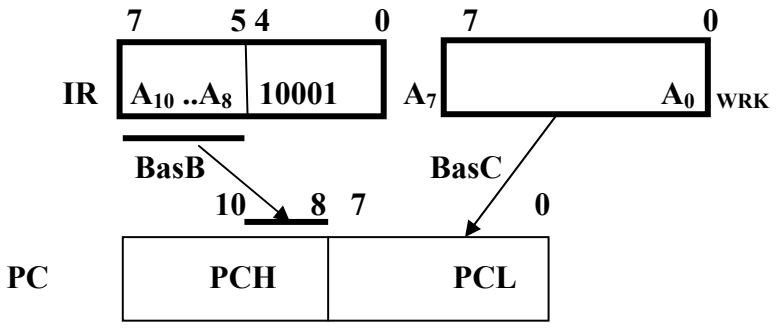
Преобразование switch(ADC[IR]) выбирает одну из трасс микрокоманд **add a,ri** или **add a,#d, .**

Распределение по тактам с использованием переменной **step** (младшие разряды адресного регистра RAMM)

6.5. Команда ACALL

Содержание операции – переход к подпрограмме в странице размером 2 Кб. Выбрать два байта команды – старший байт в регистр IR, второй байт – во вспомогательный рабочий регистр WRK. После этого продвинутый адрес из PC переписывается в СТЕК косвенно по адресу в указателе стека SP с пре--инкрементом ++PC. Формируется новый адрес текущей страницы в PC - биты $\mathbf{A_{10}}$ **..**$\mathbf{A_0}$**.**

Схема формирования адреса



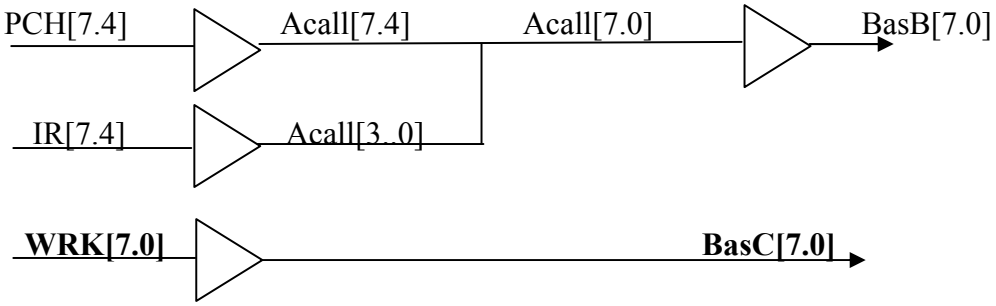
Функциональная микропрограмма:

```
switch(step)
  {
   case 0: Wrk=Code[PC++]; SP++;RAMM++; break;
   case 1: Ram[SP++]=PC;RAMM++; break;  //запись в Стек PC(7-0)
   case 2: Ram[SP]=(PC>>8); RAMM++;//запись в Стек PC(15-8)
   case 3;  PC=((PC&0xf800)|Wrk)|(((IR&0xE0)>>5)<<8);RAMM++; break;
                  // формирование  PC
    case 4: Ram[Sp]=SP; RAMM=0; //сохранение продвинутого указателя
        }
```

Реализация в MaxPlus функциональной микрокоманды

PC=((PC&0xf800)|Wrk)|((IR&0xE0)>>5)<<8; и управление передачей данных через шины

"BasB=Acall, BasC=WRK, PCL=BasC, PCH=BasB, WrC,WrB"

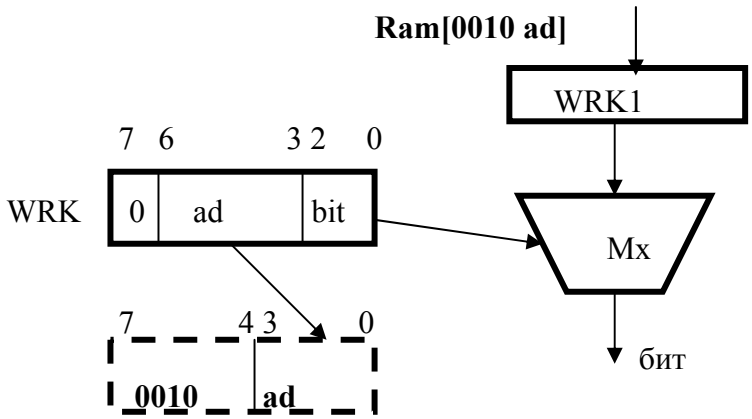


6.6. Операции с битами

Аппаратная поддержка однобитовых вычислений – уникальное свойство архитектуры MCS51. Формат битовой команды **anl c,bit**

Структура адреса бита в памяти Data **0.ad.bit**,
где **ad** – четырехбитовое смещение относительно начального адреса **20h** определяет адрес слова памяти, трехразрядное поле **bit** – номер бита в слове.

Схема выборки бита в **Data:**



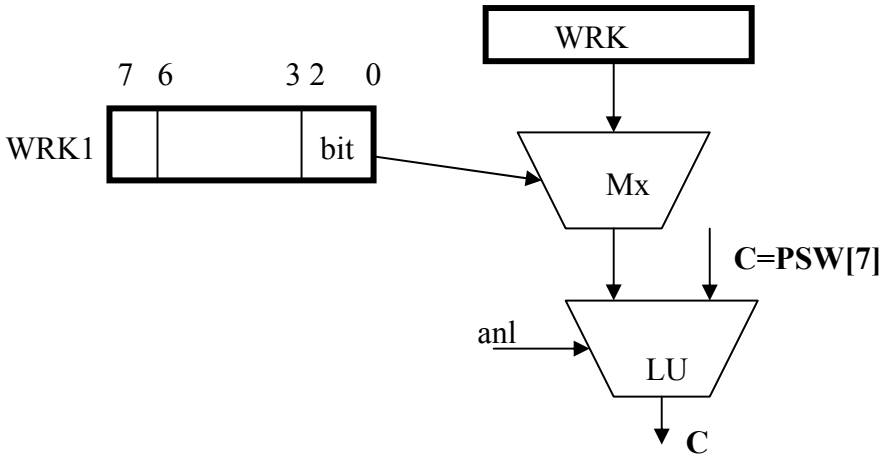
Функциональная микрокоманда чтения бит-адресуемого байта из Data:

If(!(WRK&0x80)) WRK1=Ram[0x20|(WRK&0x78)]

В SFR адрес кодируется как **1.ad.bit,** выбор регистра SFR с битовым доступом задается микрокомандой

If (WRK&0x80) WRK1=Ram[WRK&0xF8]

Схема выполнения логической операции с битом:



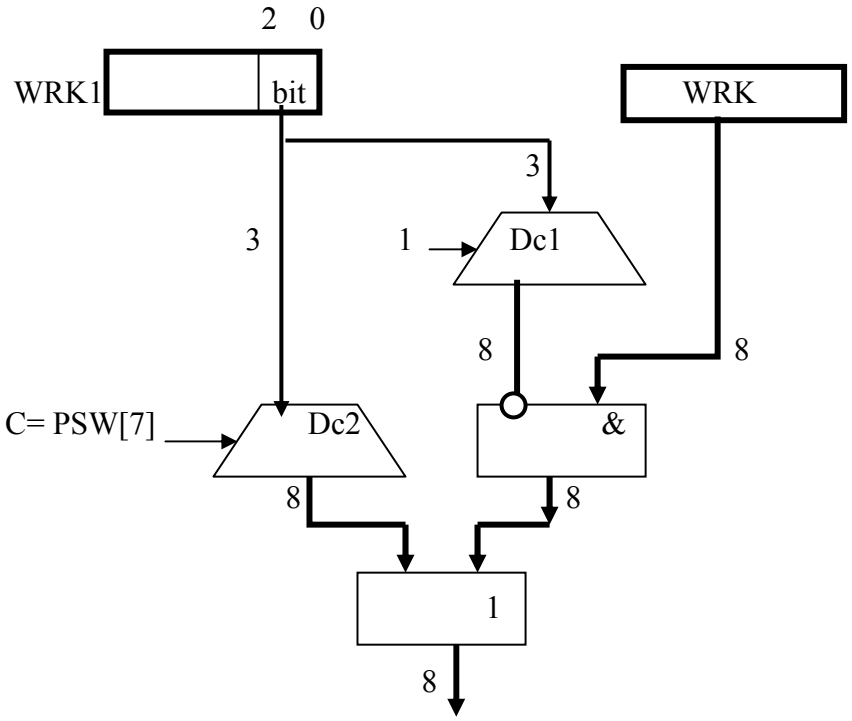
Функциональная микропрограмма выборки бита и выполнения операции в команде **anl C,bit**:

```
0   WRK1=Code[PC++];
1   if(WRK1&0x80)  WRK=Ram[WRK1&0xF8]&(1<<(WRK1&0x3));
2   if(!(WRK1&0x80))  WRK=Ram[0x20|(WRK&0x78)]&(1<<(WRK1&0x3));
4   PSWC("banl");
5   Ram[Psw]=PSW;
```

Функция формирования PSW в **PSWC(banl)**

PSW= (WRK&(1<<(WRK1&0x7))) ? PSW : PSW&0x7f;

Схема формирования слова для записи бита **mov bit,C**



Декодер DC2 декодирует номер бита в унитарный 8 битовый код, если бит C равен единице. DC1 также декодирует номер бита, полученный инверсный унитарный код сбрасывает выбранный бит в регистре Wrk.

Восемь параллельных двухвходовых схем ИЛИ формируют слово, в котором устанавливается значение бита C.

Функциональная микропрограмма:

```
0  WRK1=Code[PC++]; RAMM++;
1  if(WRK1&0x80)  {WRK=Ram[WRK1&0xF8]; RAMM++;}
2  if(!(WRK1&0x80))  {WRK=Ram[0x20|(WRK1&0x78); RAMM++;}
3  If(PSW>>7)     { WRK=  WRK |(1<<(WRK1&0x7); RAMM++;}
4  If(!(PSW>>7))   {WRK=  WRK | (~(1<<(WRK1&0x7)); RAMM++;}
5  if (!(WRK1&0x80))  {Ram[0x20|(WRK1& 0x78)=WRK; RAMM=0;}
6  if (WRK1&0x80)  {Ram[WRK1&0xF8]=WRK; RAMM=0;}
```

Демонстрационный проект в MaxPlus состоит из следующих графических файлов **.gdf**

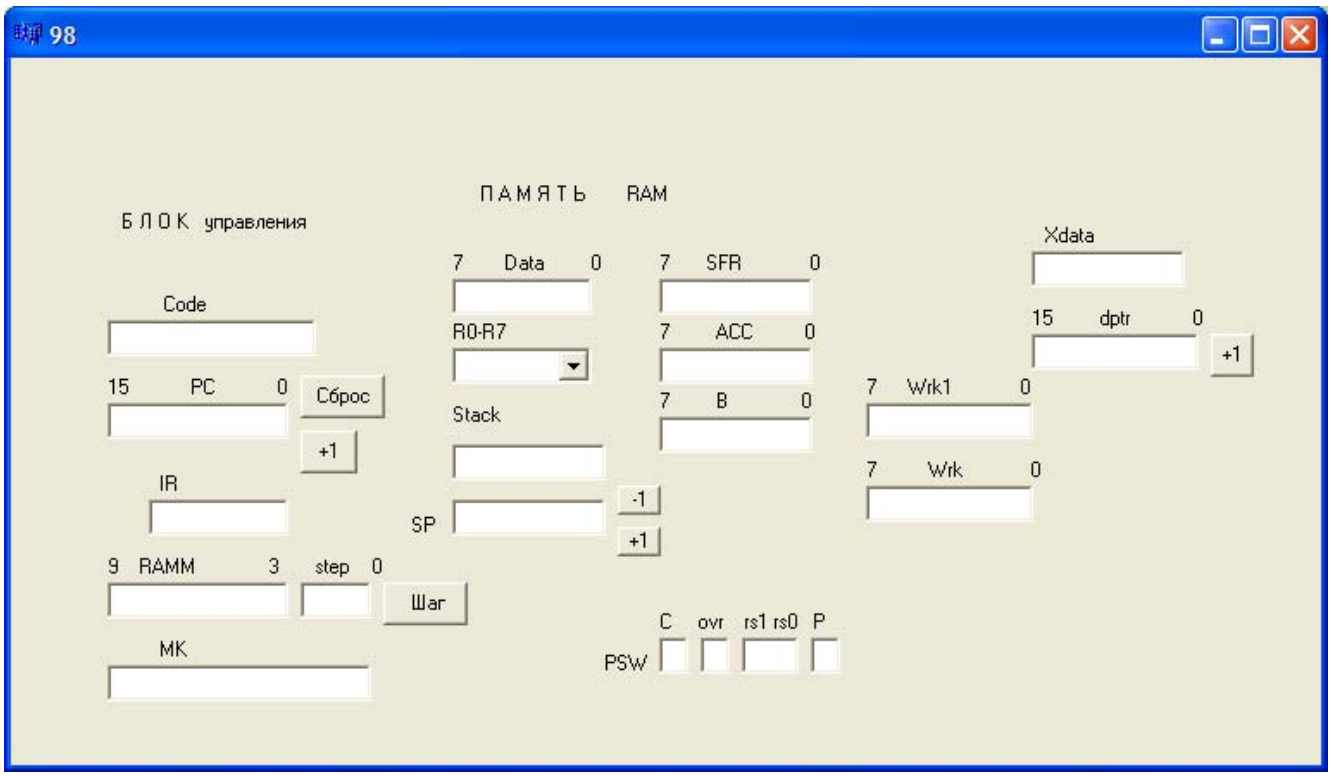
– **alb** – схема всего проекта из 6 функциональных блоков
- **16mem** – блок 16-разрядных регистров PC, DPTR и памяти Crom и Xdata
- **reg8** - блок 8-разрядных регистров-Acc,B,Wrk1,Wrk,SP и памяти Ram
- **bas** - управление шинами BasB, BasC
- **ralu** - арифметико-логическое устройство
- **bit** - выполнение операций с битами и регистр PSW
- **control** - устройство управления CU

VII. Микропрограммирование в Си

7.1. Функциональное моделирование в Borland C++

Ниже приведено окно, отображающее элементы структурной схемы рис.3.1.

Такты формируются в пошаговом режиме, в каждом такте выполняется одна микрокоманда. Состояние регистров отображаются в окнах. Пример программы моделирования с демонстрацией результатов в окне.



char Code[7]= {0x24,0x15,0x28,0x25,0x90,0x25,0x10}; //тест в кодах команд
const char P1 = 0x90;
long DAC[16]; //память декодирования кода операции в адрес микрокоманды

//==========функции кодирования микропрограммы
struct cod_mc{
char ifo;char basa; char basb; char wrb; char basc;
char ACX; char ALU; char wrc; char adsfr;
char unicod; char Cin; char biCC; char consta;
} mema0; **//поля структурной микрокоманды ЭВМ**

//==========микрооперации в полях структурной микрокоманды
char *basb[ ]=" Ram, xram,code, acc, dd, pcl, dpl, H, BitPSW, ALU, B";
char *basc[ ]="P1, ACC, B, WRK, WRK1, PSW, PCH, DPH, L, Acall, ALU, SP, H, Bita, alu";
char basa[ ]=" WRK, Ar, Adbit, Asfr, SP";
char acx[ ]=" PC, WRK, WRK1, Intr, DPTR";
char wrc[ ]=" Ram, ACC, Xram, PCH,DPH, RB";
char wrb[ ]="SP, B, WRK, WRK1, DPH, RA, PSW, IR, DPL, PCL";
char alu[ ]=" res, suba, subb, add, or, xor, and, set";
char const[ ]=" zero, mone, moct, abi, sp0";
char const ifo[ ]=" B0, W7, neCW, CC";
char biCC[ ]=" C8, sign, ACC0";
char Cin[ ]=" H, L, CC, PSW7";
char adsfr[ ]=" SP, ACC, DPH, DPL, P1, PSW";
char unicod[ ]= " PCPlus, Splus, DPTRPlus, RAMMPluc, RAM0" ;

char __fastcall TForm1**::findcode**(char *pole, char *name)
{ //в строке *pole найти имя *name и вернуть ее номер 0,1,2, }

long codDCM**;**
void __fastcall TForm1**::Coding**(char *pol) //записать код поля в структуру микрокоманды

```
{
        if(pole== “ifo” )                 mema0.ifo=findcode(ifo,pol);
          if(pole== “basa” )              mema0.basa=findcode(basa,pol);

          if(pole== “basb” )              mema0.basb=findcode(basb,pol);
     ……………………………. }
void __fastcall TForm1::Codmema(void) //формировать бинарный код микрокоманды , найти ее адрес DCM[16] и сохранить адрес в ROMM[RAMM]
{
   char i;

codDCM=((((((((mema0.ifo<<3)+mema0.basa)<<4+mema0.basb)<<4+mema0.wrb)<<4+mema0.basc)<<4+mema0.wrc)<<4)+mema0.adsfr)<<3)+mema0.unicod;
   for(i=0; i<16; i++) //i-адрес, mcod-декод в таблице кодов
   {if(DCM[i]==codDCM) continue;
   if(DCM[i]==0) { DCM[i]=codDCM;  //i mk
   ROMM[RAMM]=i;  continue;}


void  MicroCodMem( char *SS)
{  //лексический разбор строки SS
  //найти имя поля *pole и выполнить  Coding(*pole)
  // выполнить Codmema(void) }

//=================================================-моделирование

void __fastcall TForm1::Reset(TObject *Sender)
{  //функция клавиши  Сброс
   //tab1.mom.aa=5;

PC=0;  Ram[Sp]=07;
for(char i=0x28;i<0x28;i++) ADC[i]=0x11;  // декодирование команд add a,r0-r7
ADC[0x24]=0x12;           // декодирование команды add a,#d

for(char i=0x11;i<0xF1;i=i+0x10) ADC[i]=0x14;  // декодирование команд acall met
ADC[0x82]=0x15; // декодирование команды anl c, bit
//начальное состояние
        Ram[0]=0x11;
        ACC=0;     //Edit6->Text='0';
        PC=0;        //Edit2->Text='0';
        SP=07; Ram[Sp]=SP;
        RAMM=0;  //Text='0';
}

void __fastcall TForm1::PSWC(char *name)
{ //вычисление признаков результата для операции *name }

void __fastcall TForm1::Step(TObject *Sender) /Функция клавиши ШАГ
{
{ IR=Code[PC++];  //ROMM[0]  выборка команды в начале цикла
MicroCodMem(“BasA=PC, basB=Code, Mr=IR, incPC”);
        RAMM=ADC[IR]<<3;  //начальный адрес трассы микрокоманд
switch(ADC[IR])        //декодирование команд
 {
   case 0x11:             //микропрограмма add a,ri
```

```
        switch(RAMM&0x7)
       {  case 0: RA=ACC, RB= Ram[(PSW&0x18)|(IR&0x3)];RAMM++; break;
MicroCodMem("BasC=ACC, BasB=RAM,  BasA=IR, WrB=RA,WrC=RB, RAMM++");
            case 1: ACC=RA+RB, PSWC("add"); RAMM++; break;
MicroCodMem(" BasB=ALU, WrB=ACC, ALU=add,  WrSFR, RAMM++ ");
            case 2: Ram[Acc]=ACC;  RAMM++;  break;
MicroCodMem("BasC=ACC, WrC=Ram, BasA=Acc, RAMM++");
             case 3: Ram[Psw]=PSW; RAMM=B;
MicroCodMem("BasC=PSW, WrC=SFR, BasA=Psw, RAMM++");
                          }
                   break;
    case 0x12:      //микропрограмма add a,#d
          switch(RAMM&0x7)  {
            case 0: RA=ACC, RB=Code[PC++];RAMM++; break;
MicroCodMem("BasA=PC, BasB=Code, BasC=ACC, WrB=RA, WrC=RB,  RAMM++");
            case 1: ACC=RA+RB, PSWC(«add»); RAMM++; break;
MicroCodMem("BasB=ALU, WrB=ACC, ALU=add, WePSW, RAMM++ ");
            case 2: Ram[Acc]=ACC;  RAMM++;  break;
MicroCodMem("BasC=ACC, WrC=Ram, BasA=Acc, RAMM++");
            case 3: Ram[Psw]=PSW; RAMM=0;
MicroCodMem("BasC=PSW, WrC=Ram, BasA=Psw, RAMM=0");
       }
              break;
     }

//==================================Вывод состояния регистров
    Instr->Text=itoa(IR,stro,16);
    Acu->Text=itoa(ACC,stro,16);
    Work->Text=itoa(Wrk,stro,16);
    Work1->Text=itoa(Wrk1,stro,16);
    ProgCnt->Text=itoa(PC,stro,16);
    //Ramm->Text=itoa(RAMM,stro,16);
}

void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
   CheckBox1->State=cbChecked;

}
//=======================создать файлы для загрузки памяти в MaxPlus
void __fastcall TForm1::files(char *name)
{
     ADC[  ] → failadc
     ROMM[ ] → failromm
     DCM[  ] → faildcm
    Const[ ] → failconst
    Adsfr[ ] → failadsfr
    Code[ ]→ failcode  }
```

Для всех объектов экрана предусмотрены события ввода начальных значений. Каждая команда может быть выполнена индивидуально в пошаговом режиме и в последовательности команд теста.

7.2. Формат микрокоманды

Кодирование полей микрокоманды приведено в таблицах 4.1.- 4.4, 6.1. 6.2.

Структура микрокоманды объединяет все используемые в структурных микрокомандах поля.

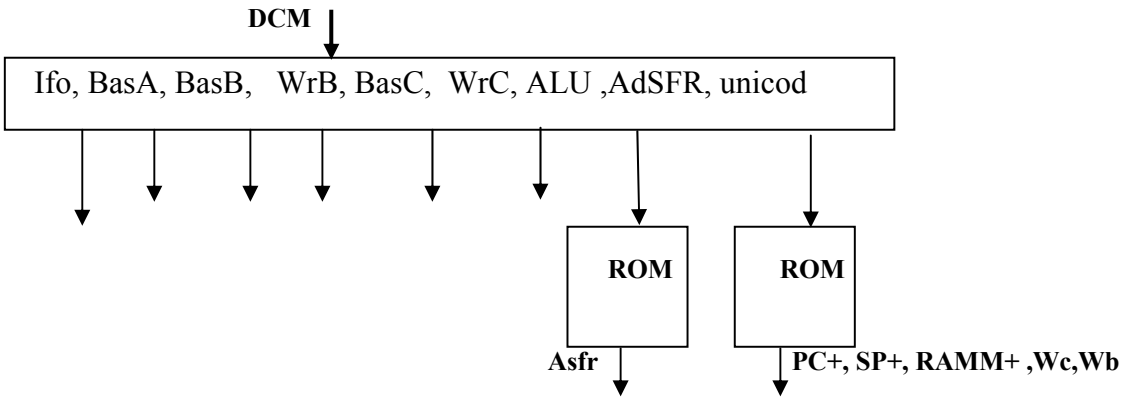
Ifo,	**BasA,**	**BasB,**	**WrC,**	**BasC,**	**WrB,**	**ALU ,**	**Cin,**	**AdSFR,**	**PCPlus,**	**SPlus,**	**RAMMPlus,**	**RAMM0,**	**consta**
3	4	4	3	4	4	3	2	3	1	1	1	1	3

Уникальные сигналы **PC+, SP+, RAMM++** могут быть закодированы максимальным 3-х битным кодом **unicod** или сгруппированы и закодированы в микрокоманде с учетом их распределения по функциональным блокам.

Итого ожидаем 36 бит в двоичной структурной микрокоманде, выбираемой из DCM в схеме CU рис.5.2.

Ifo,	**BasA,**	**BasB,**	**WrC,**	**BasC,**	**WrB,**	**AdSFR,**	**ALU,**	**Cin,**	**unicod,**	**consta**
3	**4**	**4**	**3**	**4**	**4**	**3**	**3**	**2**	**3**	**3**

Декодирование микрокоманды после чтения из DCM:



В шинной структуре ЭВМ большая часть управляющих полей декодируется функциональными элементами, управляющими шинами и АЛУ. Остается небольшое число одиночных сигналов, которые объединяются максимальным кодированием в отдельном поле микрокоманды. Выделенные группы закодированных сигналов также могут использовать для декодирования ROM, размещаемые в функциональных блоках.

7.3. Принципы кодирования микрокоманд в Си

Если микропрограмма проверена, то предполагается, что символическое описание структурной микрокоманды корректно и микрокоманда может быть использована для обратного кодирования и формирования файлов загрузки блоков постоянной памяти, используемых в управляющем устройстве.

Процедура кодирования – наиболее трудоемкий этап при переходе к реализации микропрограммы в CU.

Фрагмент микропрограммы, совмещающей моделирование и кодирование, для команды **mov a,ri** в BorlandC++.

```
        switch(RAMM&0x7)  //номер такта
{           case 0: RA=ACC, RB= Ram[(PSW&0x18)|(IR&0x3)];RAMM++; break;
MicroCodMem("BasC=ACC, BasB=RAM,  BasA=IR, WrB=RA,WrC=RB, RAMM++, WrB,WC");
            case 1: ACC=RA+RB, PSWC(«add»); RAMM++; break;
MicroCodMem(" BasB=ALU, WrB=ACC, ALU=add,  WrSFR, RAMM++, WB ");
            case 2: Ram[Acc]=ACC;  RAMM++;  break;
MicroCodMem("BasC=ACC, WrC=Ram, BasA=Acc, RAMM++,WC");
          case 3: Ram[Psw]=PSW; RAMM=B;
MicroCodMem("BasC=PSW, WrC=SFR, BasA=Psw, RAMM++,Wc");
                        }
                  break;
```

В каждом такте выполняется функциональная микрокоманда и функция кодирования MicroCodMem(" "), параметром которой является структурная микрокоманда.

Простым лексическим разбором идентифицируются поля микрокоманды и символические описания соответствующих кодов микроопераций. Например, в тексте "BasC=ACC" указано имя поля микрокоманды BasC и ACC- признак кода микрооперации.

Если представить CU рис.5.2. как многоуровневое обращение к памяти

DCM[ROMM[ADC[IR]]],

то функция кодирования позволяет сформировать код микрокоманды в DCM и найти ее адрес в DCM. Этот адрес является кодом функциональной микрокоманды и хранится в ROMM по известному на стадии исполнения адресу ADC[IR].

Таким образом, при отладке тестов может быть в любой момент выполнено полное кодирование микропрограммы в CU и подготовлены файлы содержимого элементов памяти для MaxPlus.

В разделе 7.1 приведены фрагменты программы моделирования и кодирования в BorlandC++.

VIII. Моделирование схемы проекта в MaxPlus

В проекте схемы ЭВМ в MaxPlus совмещаются результаты выполнения нескольких этапов проектирования – неформальный выбор структурной схемы и схем блоков; неформальное проектирование функциональной схемы ЭВМ в MaxPLus; неформальное микропрограммирование, моделирование и кодирование в BorlandC++.

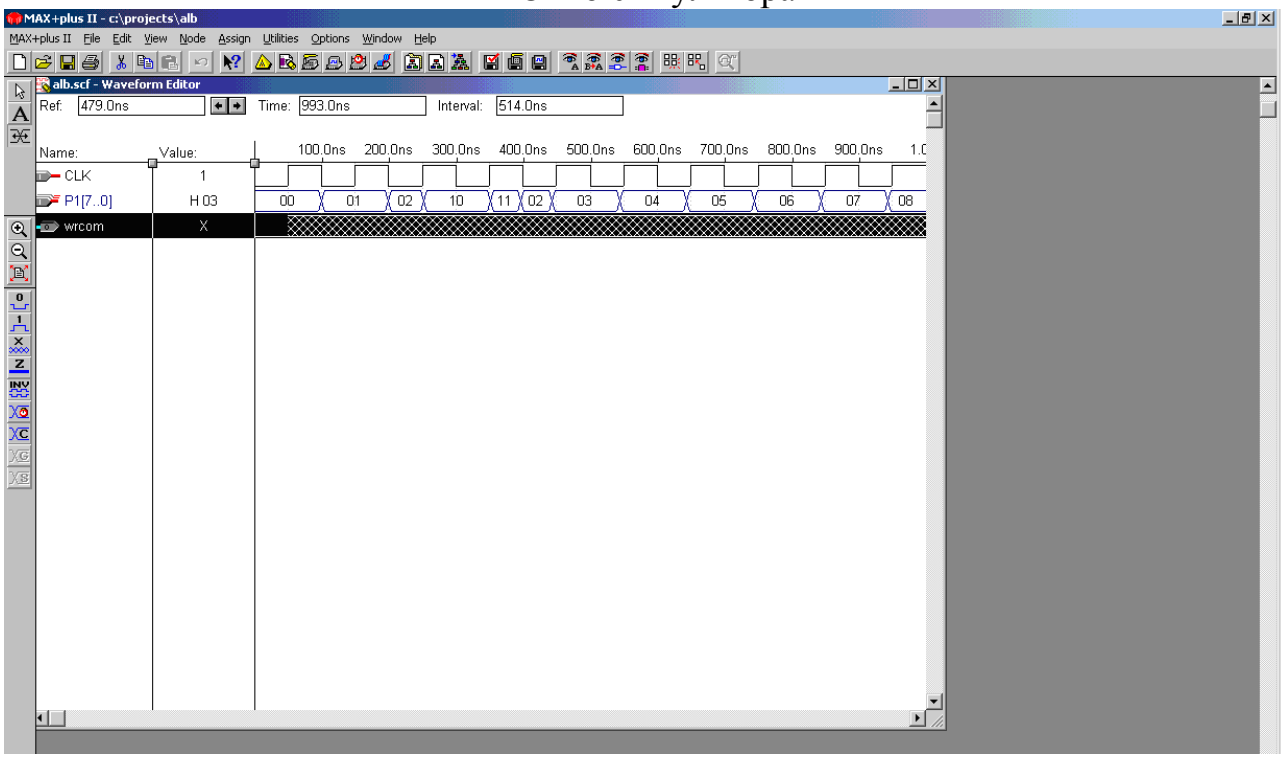
Проект в MaxPlus содержит полное описание схемы ЭВМ и позволяет выполнить ее верификацию в Симуляторе по известной методике [2,3].

Здесь приводится краткое описание основных шагов применения Симулятора и демонстрируется относительная доступность, наглядность и точность методики верификации.

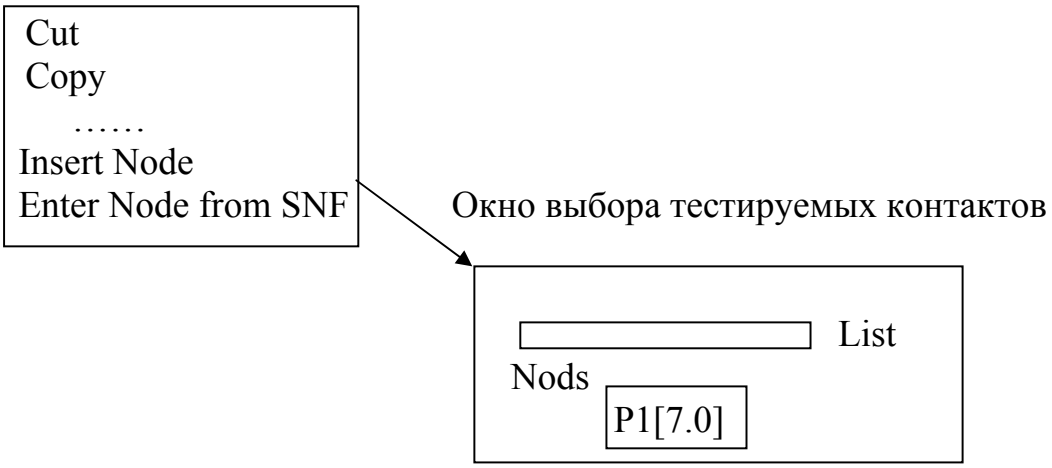
Моделирование схемы ЭВМ с учетом частоты синхронизации и реальных задержек в схемах и функциональных элементах позволяет выполнить не только контроль работоспособности, но и провести исследование для выбора максимальной частоты синхронизации и скорости выполнения команд.

1. Открываем редактор временных диаграмм MaxPlus/Waveform.Editor.

Окно симулятора



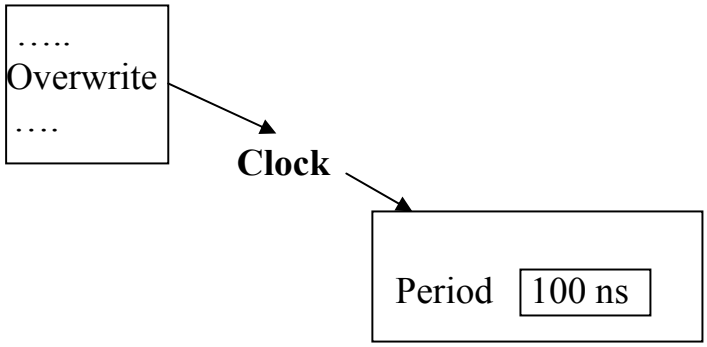
2. В столбце Name Основного меню клавишей мыши вызываем меню выбора контактов и групп узлов



3. Выбираем из списка элементов и контактов SNF узел(имя группы контактов).

Сначала выбирается общий синхросигнал CLK в Окно Симулятора

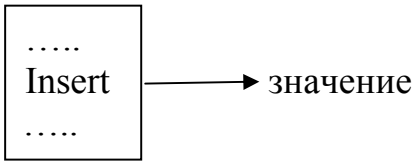
Правой клавишей мыши выбираются последовательно меню



В Окне Симулятора строится временная диаграмма CLK.

4. Выбираем входные и выходные узлы Nods.

Маркером указываем интервал тестирования на временной шкале CLK и правой клавишей мыши вводим значение входных контактов



5. Запуск Симулятора MaxPlus/Simulator

Задаем Start Time (ns) и End Time (μs)

При работе Симулятора формируется временная диаграмма с указанием значений 0/1 для одиночных сигналов и коды для групп сигналов.

Контролируются ошибки исполнения. Возможна коррекция установок по всем этапам.

Литература

1. Довгий П.С. Скорубский В.И. Методическое пособие к лабораторным работам по курсу Организация ЭВМ.
2. Комолов Д.А. Мяльк Р.А. Зобенко А.А. Филиппов А.С. Системы автоматизированного проектирования фирмы Altera Max+Plus II и Quartus II, М: РадиоСофт, 2002.
3. Антонов А.П. Язык описания цифровых устройств AlterHDL, Практический курс. М: Радио Софт, 2001.
4. Майоров С.А. Новиков Г.И. Принципы организации цифровых машин Л: Машиностроение, 1974.
5. Таненбаум Э. Архитектура компьютера, 4-е издание, Питер, 2002,
6. Столлингс У. Структурная организация и арихитектура компьютерных систем, М: Вильямс, 2002 .

Приложение 1

Задание для курсового проекта

Предлагается список команд, для которых по рассмотренному выше плану разрабатываются – структурные и функциональные схемы и микропрограммы выполнения команд, программа моделирования и кодирования на языке Си. В отчете приводится описание работы ЭВМ при выполнении команд задания.

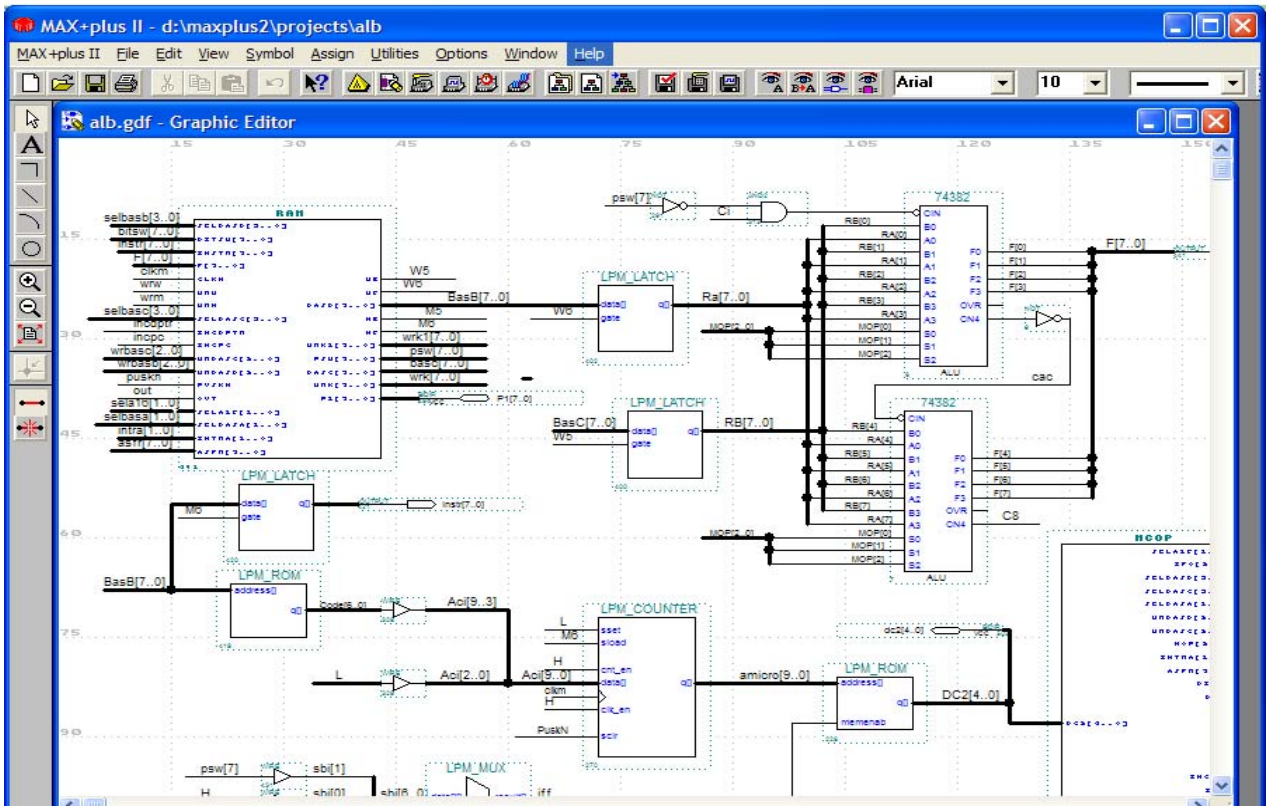
Варианты заданий

1	**add a,{ri,#d}**	**rlc a**	**push ad**	**ajmp ad11**
2	**add a,{@rj,ad}**	**rrc a**	**pop ad**	**sjmp rel**
3	**subb a,{ri,ad}**	**rl a**	**xch a,{@rj,ad}**	**ljmp ad16**
4	**subb a,{@rj,#d}**	**rr a**	**xch a,{ri,ad}**	**acall ad11**
5	**inc {ri,ad,dptr}**	**cpl a**	**xchd a,@rj**	**lcall ad16**
6	**inc {@rj,a,ad}**	**clr {c,bit}**	**swap a**	**jc rel**
7	**dec {ri,@rj,ad}**	**anl c,{bit,/bit}**	**mov a,{ri,#d}**	**jz rel**
8	**dec {@rj,a}**	**orl c, {bit,/bit}**	**mov a,{@rj,ad}**	**jb bit,rel**
9	**anl a,{ri,#d}**	**setb {c,bit}**	**mov {ri,@rj},a**	**jnb bit,rel**
10	**anl a,{@rj,ad}**	**anl c,{bit,/bit}**	**mov {ri,@rj},ad**	**djnz ri,rel**
11	**anl ad,{a,#d}**	**rrc a**	**mov ad,{ri,#d,a}**	**djnz ad,rel**
12	**orl a,{ri,#d}**	**da a**	**mov ad,{@rj,ad}**	**cjne ri,#d,rel**
13	**orl a,{@rj,ad}**	**da a**	**mov {ri,@rj},#d**	**cjne @rj,#d,rel**
14	**orl ad,{a,#d}**	**clr {c,bit}**	**mov dptr,#d16**	**ret**
15	**xrl a,{ri,#d}**	**anl c,{bit,/bit}**	**movc a,@a+dptr**	**sjmp rel**
16	**xrl a,{@rj,ad}**	**orl c, {bit,/bit}**	**movc a,@a+pc**	**ljmp ad16**
17	**xrl ad,{a,#d}**	**setb {c,bit}**	**movx a,@rj,**	**lcall ad16**
18	**mul ab**	**anl c,{bit,/bit}**	**movx @rj,,a**	**jc rel**
19	**div ab**	**orl c, {bit,/bit}**	**movx a,@dptr**	**jz rel**
20	**add a,{@rj,ad}**	**anl c,{bit,/bit}**	**movx @dptr,a**	**jb bit,rel**
21	**add a,{ri,#d}**	**anl c,{bit,/bit**	**push ad**	**ret**
22	**subb a,{ri,ad}**	**rlc a**	**pop ad**	**cjne @rj,#d,rel**
23	**subb a,{@rj,#d}**	**rl a**	**xch a,{@rj,ad}**	**cjne ri,#d,rel**
24	**mul ab**	**clr {c,bit}**	**xch a,{ri,ad}**	**djnz ad,rel**
26	**dec {ri,@rj,ad}**	**mov c,bit**	**movc a,@a+pc**	**jnb bit,rel**
27	**xrl a,{ri,#d}**	**mov bit,c**	**movx a,@dptr**	**lcall ad16**
28	**inc {@rj,a,ad}**	**anl c,{bit,/bit}**	**movx @dptr,a**	**ljmp ad16**

Приложение 2

Работа с проектом в MaxPlus

Окно редактирования схемы в MaxPlus



Меню **MAX Plus**

MAX +PlusII

Графический редактор
Иерархия дисплеев
Редактор Символов
Редактор Текстовый
Редактор Временных диаграмм
Editor Floar Plan
Компилятор
Симулятор
Timing Analyzer
Программатор
Message

File

Project → Name →
New → тип редактора

Open
Delete
Retrieve
Close
Save
Save as
Info - информация по открытому файлу
Size - размер листа для редактора

View

Fit -мелкий масштаб
Zoom in - увеличение масштаба
Zoom out - уменбшение масштаба
Normal
Maximal Size - максимальный масшт**аб**

Symbol

Enter Symbol - выбор из библиотек

Assign

Device - выбор ПЛИС из библиотеки

Options

Show Parametr - показать параметры элементов в gdf-файле

Show Guidelines – показать координаты элементов в gdf-файле

Порядок работы с проектом

1. MAX +PlusII → Графический редактор → Панель для рисования.
2. Маркером отмечаем позицию в панели.
3. Symbol → ввести символ → выбрать элемент из библиотеки → редактировать параметры в открывающемся окне.
4. Элементы можно дублировать и соединять контакты – слева меню рисования.
5. Левой клавишей мыши выделить соединение и правой клавишей ввести имя.
6. Левой клавишей выделить рисунок → правой {копировать, вырезать, выбрать масштаб, повторно – левой клавишей открыть файл выделенного модуля}.
7. File → Создать файл отредактированной схемы для использования в виде модуля в схеме более высокого уровня.
8. Компилировать и исправлять ошибки.

ИТМО
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПРИОРИТЕТНЫЕ НАЦИОНАЛЬНЫЕ ПРОЕКТЫ
«Образование»

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

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.