ЗМІСТ

Раздел 1. Обзор существующих решений построения микропроцессорной системы

1.1 Микропроцессорная система

Микропроцессорная система может рассматриваться как частный случай электронной системы, предназначенной для обработки входных сигналов и выдачи выходных сигналов (рис. 1.1). В качестве входных и выходных сигналов при этом могут использоваться аналоговые сигналы, одиночные цифровые сигналы, цифровые коды, последовательности цифровых кодов. Внутри системы может производиться хранение, накопление сигналов (или информации). Если система цифровая (а микропроцессорные системы относятся к разряду цифровых), то входные аналоговые сигналы преобразуются в последовательности кодов выборок с помощью АЦП, а выходные аналоговые сигналы формируются из последовательности кодов выборок с помощью ЦАП. Обработка и хранение информации производятся в цифровом виде.

Характерная особенность традиционной цифровой системы состоит в том, что алгоритмы обработки и хранения информации в ней жестко связаны со схемотехникой системы. То есть изменение этих алгоритмов возможно только путем изменения структуры системы, замены электронных узлов, входящих в систему, и/или связей между ними. Естественно, это практически невозможно сделать в процессе эксплуатации, обязательно нужен новый производственный цикл проектирования, изготовления, отладки всей системы. Именно поэтому традиционная цифровая система часто называется системой на «жесткой логике».

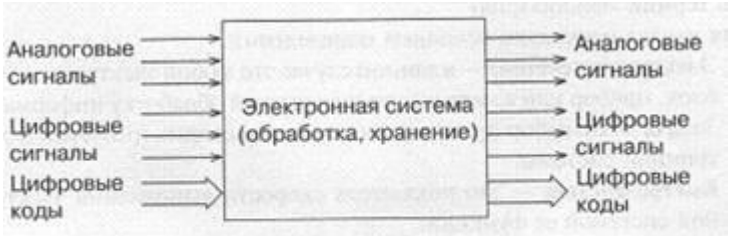


Рис. 1.1. Электронная система.

Любая система на «жесткой логике» обязательно представляет собой специализированную систему, настроенную исключительно на одну задачу или (реже) на несколько близких, заранее известных задач. Это имеет свои бесспорные преимущества:

1. Специализированная система, в отличии от универсальной, никогда не имеет аппаратурной избыточности, то есть каждый ее элемент обязательно работает в полную силу.
2. Специализированная система может обеспечить максимально высокое быстродействие, так как скорость выполнения алгоритмов обработки информации определяется в ней только быстродействием отдельных логических элементов и выбранной схемой путей прохождения информации.

Большим недостатком цифровой системы на «жесткой логике» является то, что для каждой новой задачи ее надо проектировать и изготавливать заново. Это процесс длительный, дорогостоящий, требующий высокой квалификации исполнителей. А если решаемая задача вдруг изменяется, то вся аппаратура должна быть полностью заменена. В нашем быстро меняющемся мире это довольно расточительно.

Путь преодоления этого недостатка довольно очевиден: надо построить такую систему, которая могла бы легко адаптироваться под любую задачу, перестраиваться с одного алгоритма работы на другой без изменения аппаратуры. И задавать тот или иной алгоритм мы тогда могли бы путем ввода в систему некой дополнительной управляющей информации, программы работы системы (рис. 1.2). Тогда система станет универсальной, или программируемой, не жесткой, а гибкой. Именно это и обеспечивает микропроцессорная система.

Но любая универсальность обязательно приводит к избыточности. Ведь решение максимально трудной задачи требует гораздо больше средств, чем решение максимально простой задачи.

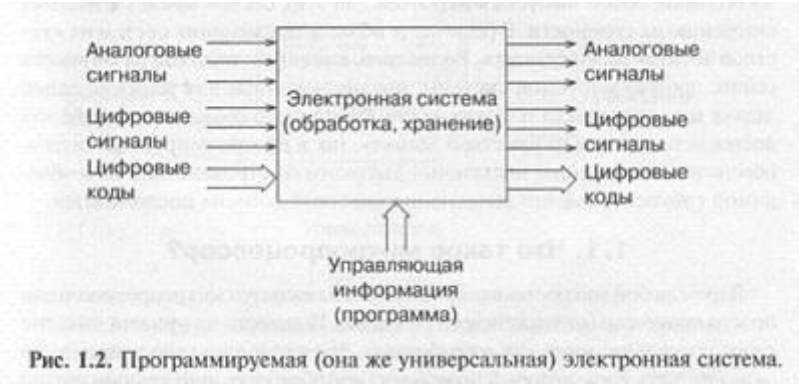


Рис. 1.2. Программируемая (она же универсальная) электронная система.

Поэтому сложность универсальной системы должна быть такой, чтобы обеспечивать решение самой трудной задачи, а при решении простой задачи система будет работать далеко не в полную силу, будет использовать не все свои ресурсы. И чем проще решаемая задача, тем больше избыточность, и тем менее оправданной становится универсальность. Избыточность ведет к увеличению стоимости системы, снижению ее надежности, увеличению потребляемой мощности и т.д.

1.2 Обзор микроконтроллеров и микропроцессоров

Для проэктирования микропроцессорных систем существует множество различных отвлетвлений разработки микропроцессоров. Ныне микропроцессоры являются устройствами шестого поколения (начиная с 2000 года). В основном отличия между микропроцессорами можно различать по особенностям реализации архитектуры. В основном для построения микропроцессорных систем используют микроконтроллеры с RISC – архитектурой. В последнее время в отдельный тип архитектуры пробиваются 64-розрядные микропроцессоры (Athlon 64 AMD, Itanium 2 Intel, PA 7100 Hewlett-Packard и другие). Для построенния систем направленных на паралельное вычисления иногда используют процессоры с особой архитектурой (транспьютеры, SMP архитектура, Кластерная архитектура).

Самые популярные микроконтроллеры на данный момент, это микроконтроллеры фирмы Atmega (AVR) и PIC.

AVR: Настоящая революция в мире микроконтроллеров произошла в 1996 году, когда корпорация Atmel представила свое семейство чипов на новом прогрессивном ядре AVR. Микроконтроллеры AVR имеют более развитую систему команд, насчитывающую до 133 инструкций, производительность, приближающуюся к 1 MIPS/МГц, Flash ПЗУ программ с возможностью внутрисхемного перепрограммирования. Многие чипы имеют функцию самопрограммирования. AVR-архитектура оптимизирована под язык высокого уровня Си. Кроме того, все кристаллы семейства совместимы "снизу вверх".

Огромную роль сыграла доступность программного обеспечения и средств поддержки разработки. У Atmel много бесплатно распространяемых программных продуктов. Хорошо известно, что развитые средства поддержки разработок при освоении и знакомстве с любым микроконтроллерным семейством играют не менее значимую роль, чем сами кристаллы. Фирма Atmel уделяет этому вопросу большое внимание.

PIC: PIC — микроконтроллеры Гарвардской архитектуры, производимые американской компанией Microchip Technology Inc. Название PIC является сокращением от Peripheral Interface Controller, что означает «периферийный интерфейсный контроллер». Название объясняется тем, что изначально PIC предназначались для расширения возможностей ввода-вывода 16-битных микропроцессоров CP1600.

В номенклатуре Microchip Technology Inc. представлен широкий спектр 8-и, 16-и и 32-битных микроконтроллеров и цифровых сигнальных контроллеров под маркой PIC. Отличительной особенностью PIC-контроллеров является хорошая преемственность различных семейств. Это и программная совместимость (единая бесплатная среда разработки MPLAB IDE), и совместимость по выводам, по периферии, по напряжениям питания, по средствам разработки, по библиотекам и стекам наиболее популярных коммуникационных протоколов. Номенклатура насчитывает

более 500 различных контроллеров со всевозможными вариациями периферии, памяти, количеством выводов, производительностью, диапазонами питания и температуры и т. д.

ARM: Архитектура ARM (ранее Advanced RISC Machine — усовершенствованная RISC-машина, предшественник — Acorn RISC Machine) — 32-битная архитектура набора команд с сокращённым набором команд, разрабатываемая ARM Limited. Данные процессоры имеют низкое энергопотребление, поэтому находят широкое применение во встраиваемых системах и доминируют на рынке мобильных устройств, для которых важно низкое энергопотребление.

В 2007 году около 98 процентов из более чем миллиарда мобильных телефонов, продаваемых ежегодно, были оснащены по крайней мере одним процессором ARM. По состоянию на 2009 на процессоры ARM приходится до 90% всех встроенных 32-разрядных RISC процессоров. Процессоры ARM широко используются в потребительской электронике — в том числе КПК, мобильных телефонах, цифровых носителях и плеерах, портативных игровых консолях, калькуляторах и компьютерных периферийных устройствах, таких как жесткие диски или маршрутизаторы.

1.3 Обзор микропроцессора Atmega8515

ATmega8515 – это 8-разрядный микроконтроллер с внутрисхемно программируемой флэш-памятью емкостью 8 кбайт

Отличительные особенности:

Высокоэффективный, экономичный 8-разрядный AVR микроконтроллер

RISC архитектура

- Мощная система команд с 130 инструкциями, большинство из которых выполняются за один машинный цикл
- 32 восьмиразрядных рабочих регистров общего назначения
- Полностью статическое функционирование

– Производительность до 16 млн. оп./ сек. при задающей частоте 16 МГц

– Встроенное умножение за 2 цикла

Энергонезависимые память программ и данных

– 8 кбайт внутрисхемно программируемой флэш-памяти с возможностью самозаписи Долговечность: 10,000 циклов «запись-стирание»

– Возможность создания сектора предварительной загрузки с раздельными битами защиты Возможность внутрисхемного программирования программой во встроенном секторе начальной загрузки Возможность считывания во время записи

– 512 байт ЭППЗУ (EEPROM) Долговечность: 100,000 циклов «запись-стирание»

– 512 байт внутреннего статического ОЗУ

– Возможность организации внешней области памяти размером до 64 кбайт

– Программирование битов защиты программного обеспечения

Периферийные устройства

– Один 8-разрядный таймер-счетчик с отдельным предделителем и режимом компаратора

– Один 16-разрядный таймер-счетчик с отдельным предделителем, режимом компаратора и режимом захвата фронтов

– Три канала ШИМ (широтно-импульсная модуляция)

– Программируемый последовательный УСАПП (устройство синхронной или асинхронной приемопередачи)

– Последовательный интерфейс SPI с режимами главный и подчиненный

– Программируемый сторожевой таймер с отдельным встроенным генератором

– Встроенный аналоговый компаратор

Специальные функции микроконтроллера

- Сброс при подаче питания и программируемый супервизор питания
- Встроенный калиброванный RC-генератор
- Внутренние и внешние источники запросов на прерывание
- Три режима управления энергопотреблением: холостой ход (Idle), пониженное потребление (Power-down) и дежурный (Standby)

Ввод-вывод и корпуса

- 35 программируемых линий ввода-вывода
- 40-выв. PDIP, 44-выв. TQFP, 44-выв. PLCC и 44-выв. MLF

Напряжение питания

- 2.7 - 5.5В для ATmega8515L
- 4.5 - 5.5В дляATmega8515

Рабочая частота

- 0 - 8 МГц для ATmega8515L
- 0 - 16 МГц для ATmega8515

ATmega8515 — экономичный 8-разрядный микроконтроллер, основанный на усиленной AVR RISC архитектуре. ATmega8515 обеспечивает производительность 1 млн. оп. в сек на 1 МГц синхронизации за счет выполнения большинства инструкций за один машинный цикл и позволяет оптимизировать потребление энергии за счет изменения частоты синхронизации.

AVR ядро объединяет богатый набор инструкций с 32 рабочими регистрами общего назначения. Все 32 регистра непосредственно подключены к АЛУ (арифметико-логическое устройство), что позволяет указывать два регистра в одной инструкции и выполнить ее за один цикл. Данная архитектура обладает большей эффективностью кода и в 10 раз большей производительностью по сравнению с CISC микроконтроллерами.

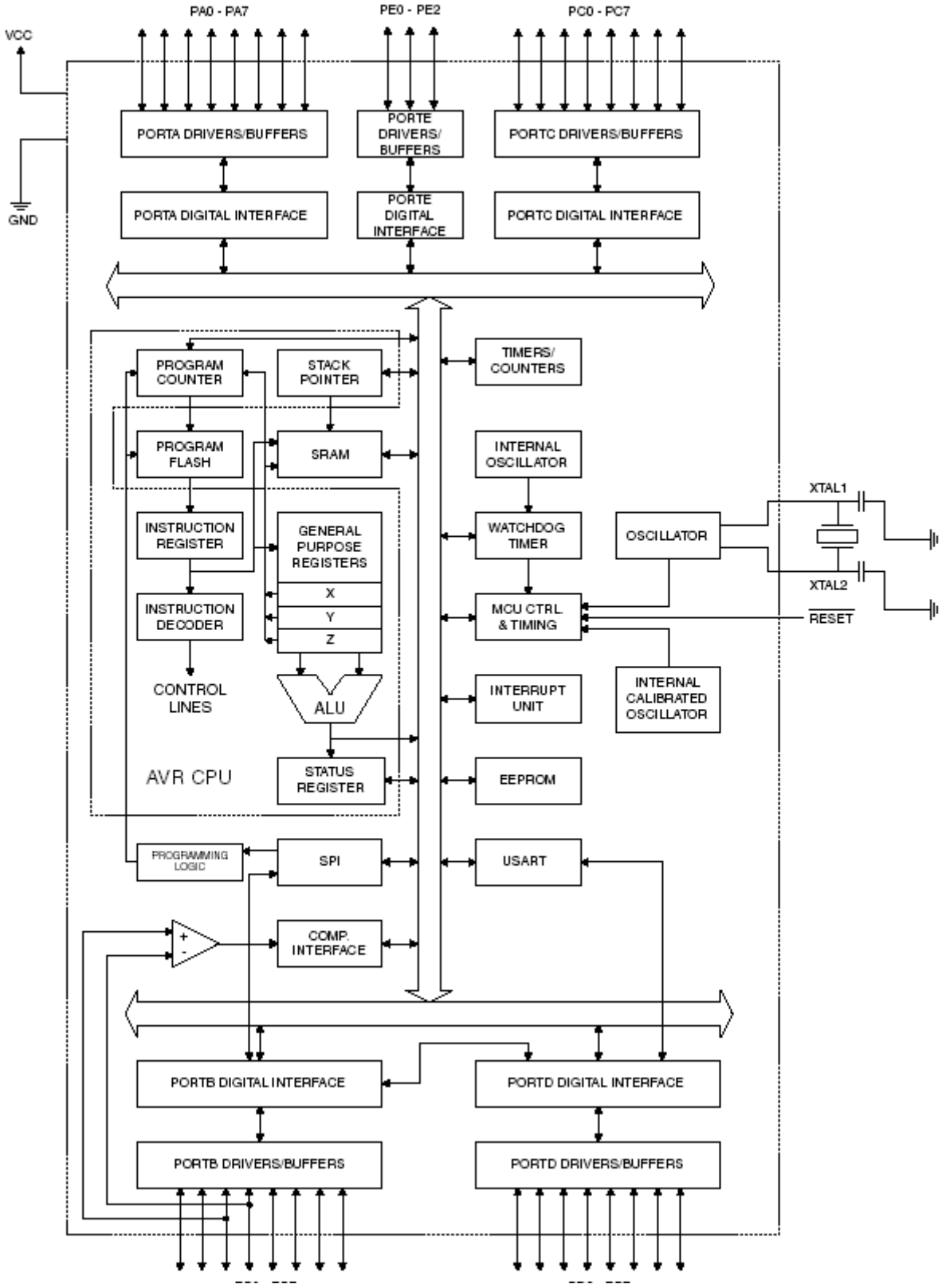


Рис 1.3 Блок- схема ATmega8515

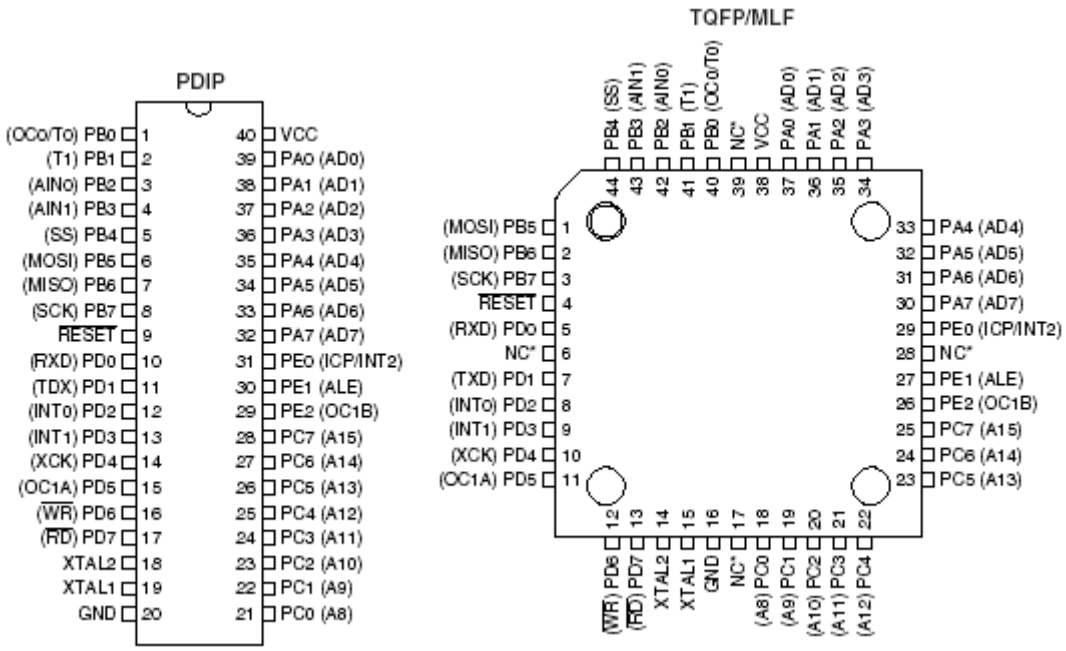


Рис 1.4 Расположение выводов ATmega8515

ATmega8515 обладает следующими возможностями: 8 кбайт внутрисхемно программируемой флэш-памяти с возможностью чтения во время записи, 512 байт ЭППЗУ, 512 байт статического ОЗУ, внешний интерфейс памяти, 35 линий ввода-вывода, 32 рабочих регистров общего назначения, два универсальных таймера-счетчика с режимами компаратора, внутренние и внешние запросы на прерывание, последовательный программируемый УСАПП, программируемый сторожевой таймер с внутренним генератором, последовательный порт SPI и три программно настраиваемых режима управления энергопотреблением. Режим холостого хода (Idle) останавливает ЦПУ, но оставляет в работе статическое ОЗУ, таймеры-счетчики, порт SPI и систему прерываний. Режим пониженного потребления (Power-down) сохраняет содержимое регистров, но останавливает генератор, выключает все встроенные функции до появления следующего запроса на прерывание или аппаратного сброса. В дежурном режиме (Standby) генератор на кварцевом резонаторе запущен, а остальная часть отключена. Данный режим позволяет реализовать быстрый запуск в комбинации с малым потреблением.

Устройство выпускается по разработанной Atmel технологии энергонезависимой памяти высокой емкости. Встроенная ISP флэш-память может внутрисхемно перепрограммироваться через последовательный интерфейс SPI, обычным программатором энергонезависимой памяти или запущенной программой в секторе начальной загрузки AVR ядра. Программа в секторе начальной загрузки может использовать любой интерфейс для записи программы. Программа в секторе начальной загрузки выполняется даже при обновлении флэш-памяти приложения, обеспечивая действительную возможность чтения во время записи. За счет комбинирования 8-разрядного RISC ЦПУ с внутрисхемно самопрограммируемой флэш-памятью на одном кристалле, позволило ATmega8515 быть мощным микроконтроллером, обеспечивающего высокую универсальность и обладающего низкой стоимостью, что делает его применение идеальным для построения встроенных систем управления.

ATmega8515 поддерживается полным набором инструментальных и программных средств для разработки приложений, в т.ч.: Си-компиляторы, макроассемблеры, программные отладчики/симуляторы, внутрисхемные эмуляторы, оценочные наборы.

Раздел 2. Проектирование микропроцессорной системы

2.1 Проектирование Микро ЭВМ

Процессорное ядро

Как процессорное ядро используется микроконтроллер ATmega8515. Упрощённая структурная схема приведена на рис. 2.1. На схеме обозначены:

- УУ — устройство управления;
- PC (program counter) — счётчик команд;
- SP (stack pointer) — указатель вершины стека;
- SREG (status register) — регистр слова состояния;
- R0...R31 — регистры общего назначения;
- АЛУ — арифметико-логическое устройство;
- ПП — память программ;
- ПД — память данных;
- КПП — контроллер приоритетных прерываний;
- EEPROM — энергонезависимая память;
- PORTA...PORTE — порты ввода-вывода;
- СШ — системная шина.

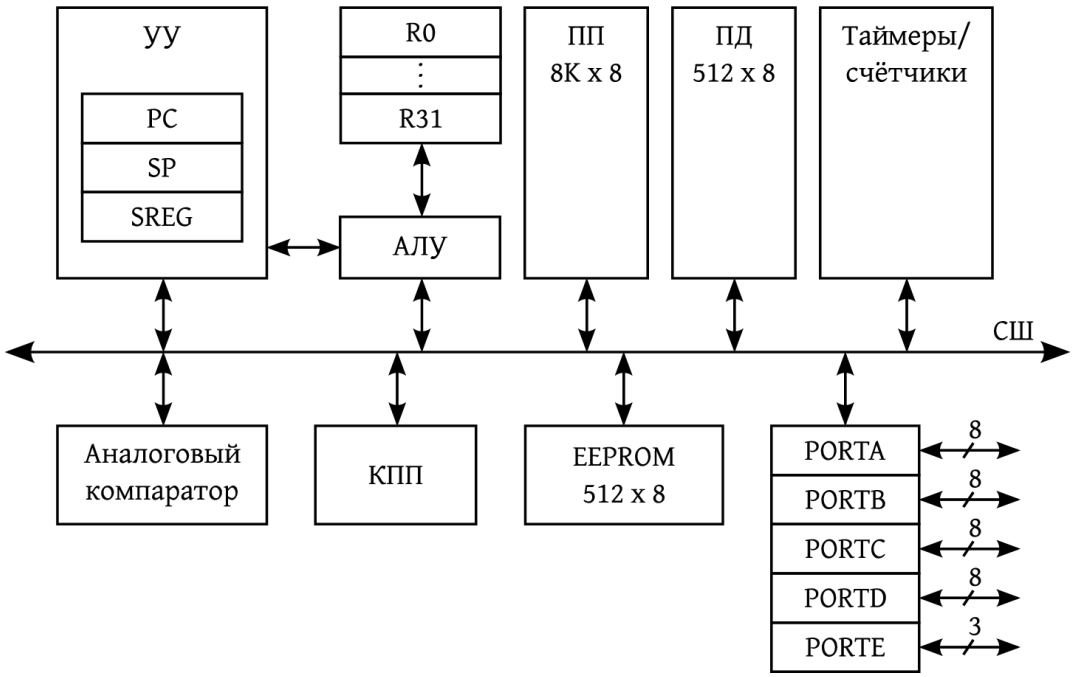


Рис. 2.1 Структурная схема ATmega8515

Любой AVR процессор имеет 32 8-битных регистра общего назначения, имеющих названия от R0 до R31. Некоторые команды принимают операнды, состоящие из пары регистров, рассматривая два 8-битных регистра как один 16-битный. Пара регистров обозначается Rr:Rd. Младший из пары регистров содержит младшие биты числа, т. е. число имеет порядок байт little endian.

Старшие 3 пары регистров используются как 16-битные указатели адреса в командах косвенной адресации. Для этих пар назначены специальные имена X, Y, Z (рис. 2.2)

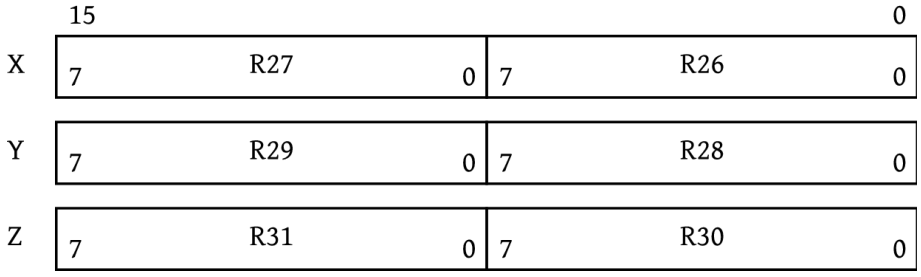


Рис. 2.2 Указатели адреса — регистры X, Y, Z

Регистры общего назначения включены в адресное пространство памяти данных по адресам $00...$1F.

В единое адресное пространство памяти данных отображаются регистры общего назначения, служебные регистры, внутренняя и внешняя память данных. Карта распределения адресного пространства памяти данных приведена на рис. 2.3. На карте обозначены следующие адреса:

- $0000…$001F — регистры общего назначения R0…R31;
- $0020…$005F — 64 порта ввода/вывода (среди них: регистр слова состояния, указатель стека, регистры таймеров, порты микроконтроллера);
- $0060…$025F — резидентная память данных объёмом 512 x 8;
- $0260…$FFFF — внешняя память данных (если подключена).

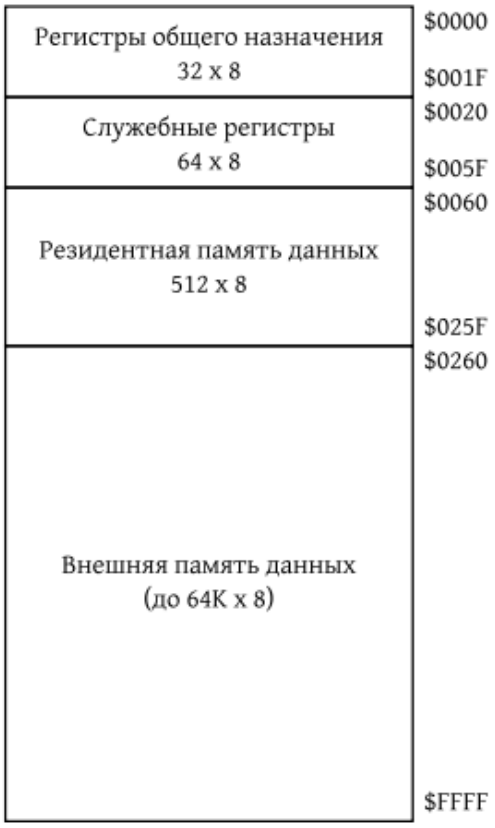


Рис. 2.3 Карта адресного пространства памяти данных

Порты ввода/вывода предназначены для управления режимами работы различных узлов микроконтроллера (порты микроконтроллера, таймеры, контроллер прерываний, настройка внешней памяти данных, АЦП,

UART и др.) Всего в микроконтроллере 64 порта ввода/вывода и они имеют номера $00...$3F.

В микроконтроллере предусмотрено два способа работы с портами ввода/вывода. Первый способ предусматривает использование специальных команд IN и OUT. В качестве операндов этим командам передаётся непосредственно номер порта (то есть, число $00...$3F). Кроме того, предусмотрены команды CBI и SBI для побитовой работы с портами ввода/вывода $00...$1F.

Поскольку порты ввода/вывода отображаются в адресное пространство данных, то к ним можно получить доступ и при помощи стандартных команд работы с памятью данных (LD и ST). Чтобы вычислить адрес порта ввода/вывода в памяти, необходимо к адресу порта ввода/вывода необходимо добавить $20 (см. карту распределения памяти данных). Таким образом, обращение к портам ввода-вывода возможно по адресам $20...$5F памяти данных.

Регистр слова состояния SREG (рис. 2.4) содержит флаги, устанавливаемые согласно результату выполнения последней инструкции, а также флаги управляющие работой микроконтроллера.

SREG	I	T	H	S	V	N	Z	C
	7	6	5	4	3	2	1	0

Рисунок 2.4 Порт ввода/вывода SREG — регистр слова состояния

Биты порта ввода/вывода SREG имеют следующее назначение:

Бит 0. C — флаг переноса из старшего разряда при выполнении арифметико-логических команд или операций сдвига.

Бит 1. Z — флаг нулевого результата арифметико-логической операции.

Бит 2. N — флаг отрицательного результата арифметико-логической операции. Совпадает со значением старшего бита результата.

Бит 3. V — флаг переполнения, устанавливается в случае перехода с $FF в $00 в ходе последней арифметической операции.

Бит 4. S = N V — флаг знака. Позволяет проверить, была ли операция выполнена с отдельной обработкой знака или с беззнаковыми операндами.

Бит 5. H — флаг переноса между тетрадами (т. е., из 3-го разряда в 4-й). Используется для выполнения двоично-десятичной коррекции.

Бит 6. T — пользовательский флаг. Не имеет служебного назначения и может использоваться пользователем по своему усмотрению. Изменяется только специальными командами изменения этого флага (BLD, BST).

Бит 7. I — флаг глобального разрешения прерываний. Если флаг установлен — разрешение/запрет прерываний определяется настройками соответствующих узлов микроконтроллера. Если флаг сброшен — все прерывания запрещены вне зависимости от других настроек. Аппаратно сбрасывается при появлении прерывания и устанавливается командой возврата из прерывания.

В описании системы команд указано, какие именно флаги изменяются каждой командой. При переходе на обработчик прерывания регистр слова состояния автоматически не сохраняется, поэтому подпрограмма-обработчик прерывания должна сохранять и восстанавливать его значение.

Микроконтроллер имеет четыре 8-битных двунаправленных порта (A, B, C, D) и один 3-битный (E). Порты обозначаются PORTx, где x — имя порта (A, B, C, D, E). Каждый порт настраивается тремя портами

ввода/вывода, каждый бит которых соответствует одному пину порта микроконтроллера.

- DDRx — порт направления. Для каждого пина порта может быть отдельно задано направление: 0 — ввод, 1 — вывод.
- PORTx — порт данных. В режиме вывода используется для вывода данных. В режиме ввода: 0 — высокоомное состояние, 1 — подключение резистора-подтяжки.
- PINx — порт входных данных. Содержит текущее значение пинов порта, вне зависимости от режима. Предназначен только для чтения.

Чтобы избежать временной нестабильности при изменении значения пина порта извне, на каждом пине установлен синхронизирующий триггер. Из-за этого новое значение порта устанавливается с задержкой 0.5–1.5 машинных цикла. Временные диаграммы приведены.

В микропроцессор встроен централизованный контроллер приоритетных прерываний. По умолчанию таблица векторов прерываний расположена по младшим адресам памяти программ. Производитель рекомендует по адресу вектора прерывания размещать инструкцию RJMP. Чем больше вектор прерывания — тем ниже приоритет. Описание векторов прерываний приведено в таблице 2.1.

Таблица 2.1 Векторы прерываний в ATmega8515

Вектор	Адрес	Название	Описание
1	$000	RESET	Сброс
2	$001	INT0	Внешнее прерывание 0
3	$002	INT1	Внешнее прерывание 1
4	$003	TIMER1 CAPT	Внешнее событие. Увеличивает таймер/счётчик 1 в режиме подсчета внешних событий.
5	$004	TIMER1	Значение таймера/счётчика 1 совпало с

		COMPA	ожидаемым значением A
6	$005	TIMER1 COMPB	Значение таймера/счётчика 1 совпало с ожидаемым значением B
7	$006	TIMER1 OVF	Переполнение таймера/счётчика 1
8	$007	TIMER0 OVF	Переполнение таймера/счётчика 0
14	$00D	INT2	Внешнее прерывание 2
15	$00E	TIMER0 COMP	Значение таймера/счётчика 0 совпало с ожидаемым значением

Прерывания можно глобально запретить, установив в 0 бит I регистра слова состояния SREG. Данный флаг запрета имеет преимущество перед всеми другими флагами разрешения прерываний.

Внешние прерывания разрешаются установкой флагов INT0, INT1, INT2 порта ввода/вывода GICR в 1.

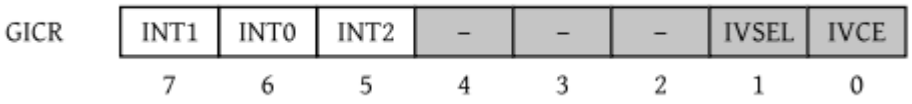


Рис. 2.5 Порт ввода/вывода GICR — регистр глобального управления прерываниями

Для внешних прерываний можно задать вид входного сигнала, при котором будет зафиксирован запрос прерывания. Прерывания INT0 и INT1 имеют 4 режима, которые задаются 2 битами ISCn1:ISCn0 порта ввода/вывода MCUCR, где n — номер внешнего прерывания. Возможные комбинации значений приведены в таблице 2.5. Если настроить фиксацию запроса прерывания по низкому уровню сигнала, то запрос прерывания будет фиксироваться до тех пор, пока сигнал не примет высокий уровень.

Подключение памяти программ и памяти данных

На рис. 2.6 показана схема подключения внешней памяти данных и памяти программ. При подключении памяти данных используются альтернативные функции портов микроконтроллера. Следует заметить, что на рисунке показано подключение максимально возможного объёма — 64 Кб. При подключении внешней памяти меньших размеров несколько старших разрядов шины адреса будут не подключены.

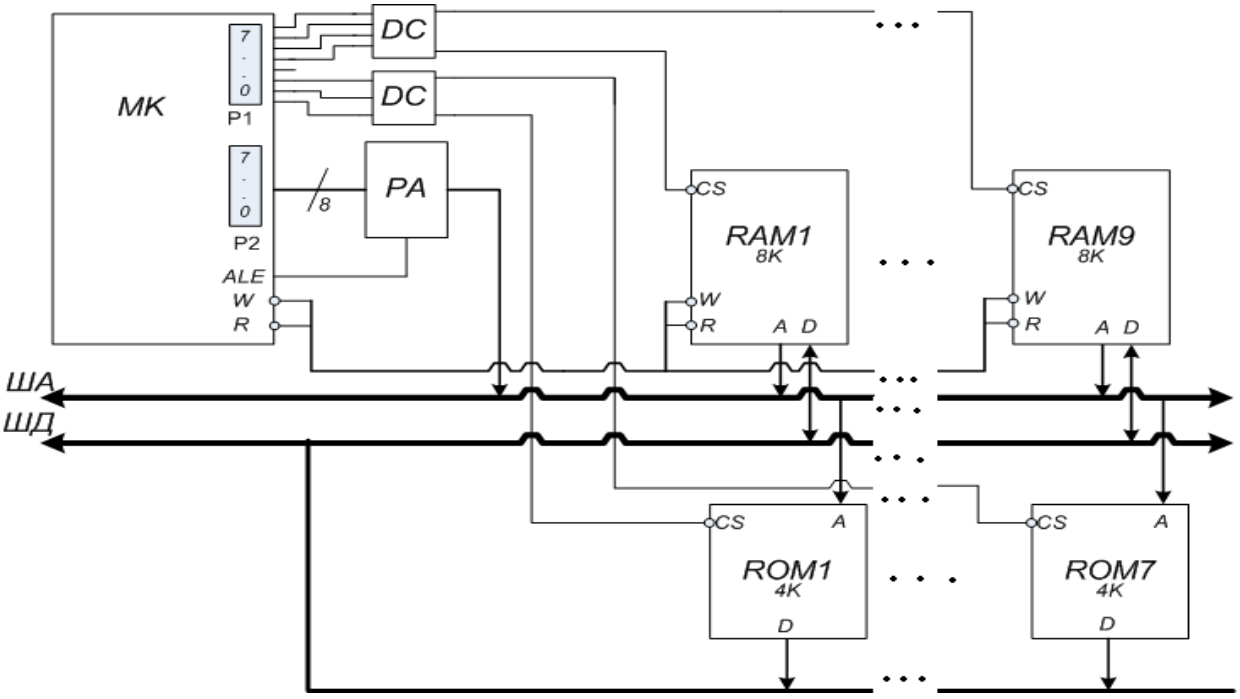


Рис. 2.6. Структурная схема подключения памяти данных и памяти программ

Централизованый контроллер прерывания

Данный режим используется для разгрузки процессора при обмене массивами данных между ОП и ВУ.

Инициатором обмена является процессор, который выполняет инициализацию контроллера прямого доступа к памяти и запускает его.

В дальнейшем два активных устройства (процессор и КПДП) поочередно захватывают шину, за счет чего осуществляется параллельная работа этих устройств.

КПДП имеет ряд адресов в адресном пространстве ВУ. (Рис. 2.7)

Внешние устройства

Подключение внешних устройств изображено на структурной схеме микро ЭВМ.

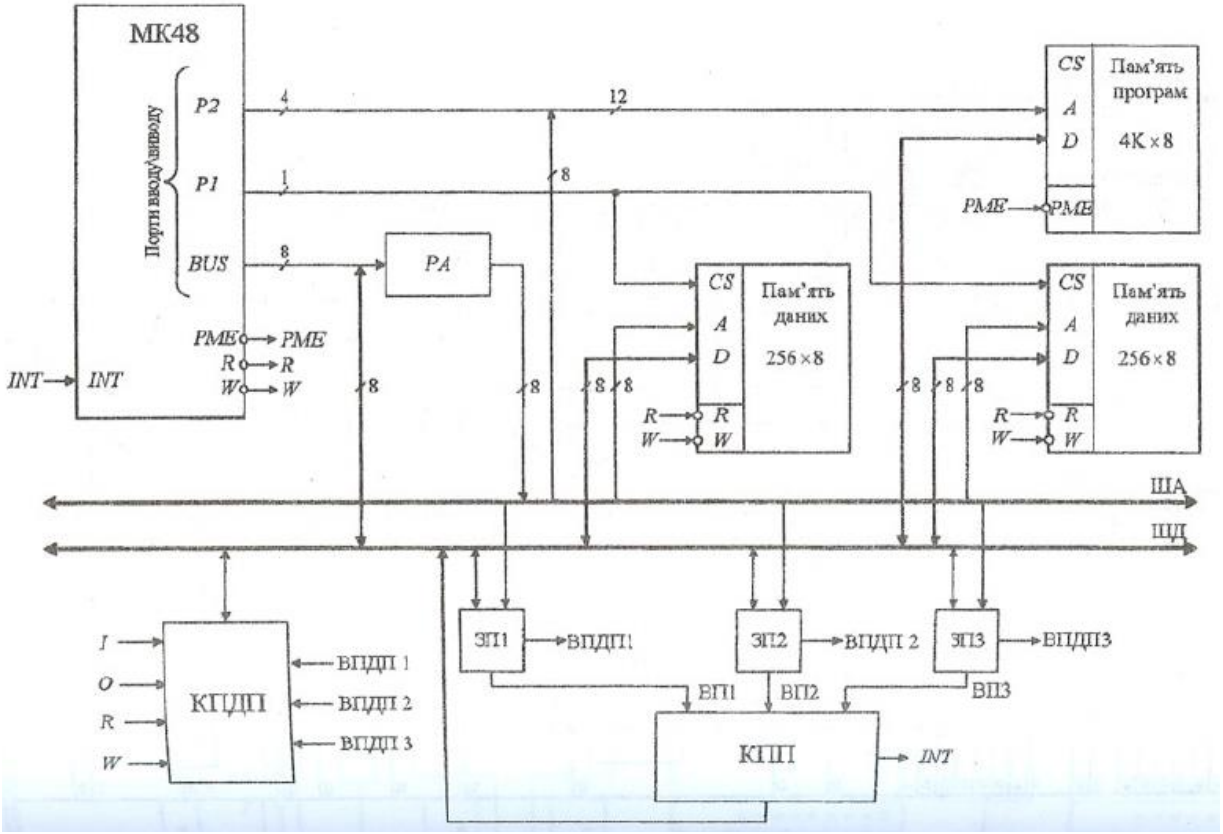


Рис. 2.7. Структурная схема централизованного КПДП

2.2 Проектирование Арифметико – Логического Блока

Структурная схема устройства для выполнения операции умножения третьим способом приведена на рис. 2.8. Для проектирования устройства используются традиционные 8-ми розрядные элементы, такие как регистр, сумматор и счётчик.

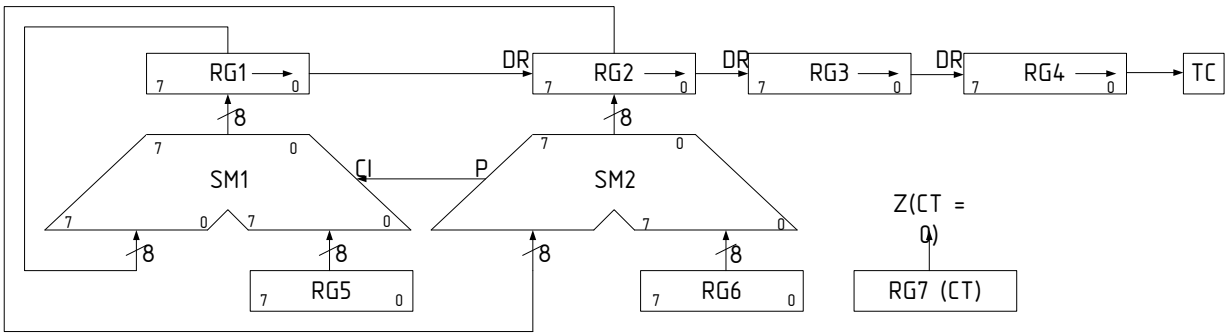


Рис. 2.8. Структурная схема устройства умножения

При умножении первым способом сума частичных результатов сдвигается вправо. Время умножения определяется формулой

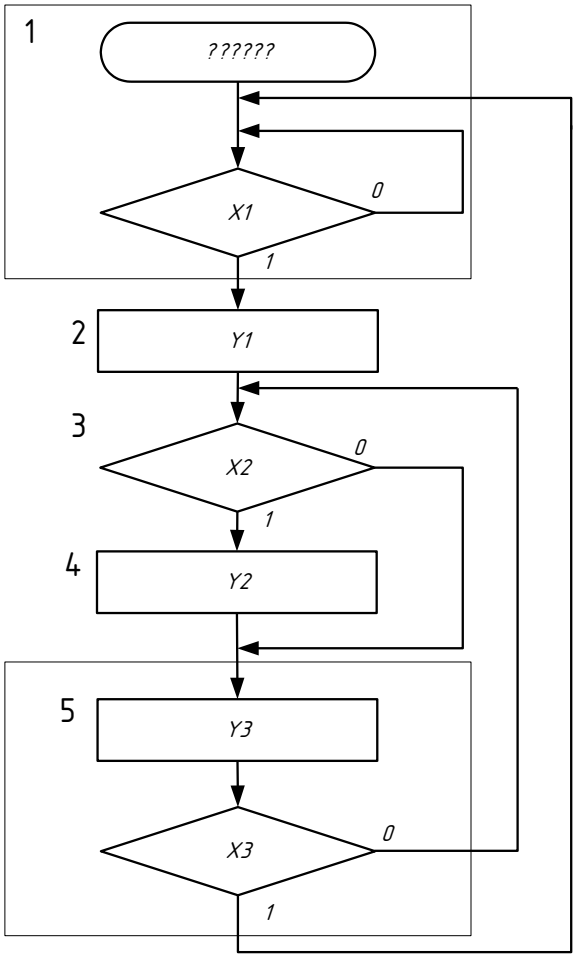
$$t_м = n(t_n + t_з)$$

, где n – розмерность розрядной сетки; t_n – время суммирования; $t_з$ – время сдвига; Результат умножения сохраняется в виде

$$Z = ((..((0 + Yx_n)2^{-1} + Yx_{n-1})2^{-1} + ... + Yx_i)2^{-1} + ... + Yx_1)2^{-1}$$

Отсюда получается, что вычисление сумы частичных результатовна i-тому (i=$\overline{1,n}$) цикле сводится к виду: $Z_i = (Z_{i-1} + Yx_{n-i+1})2^{-1}$

На рис. 2.9 изображен содержательный микроалгоритм умножения



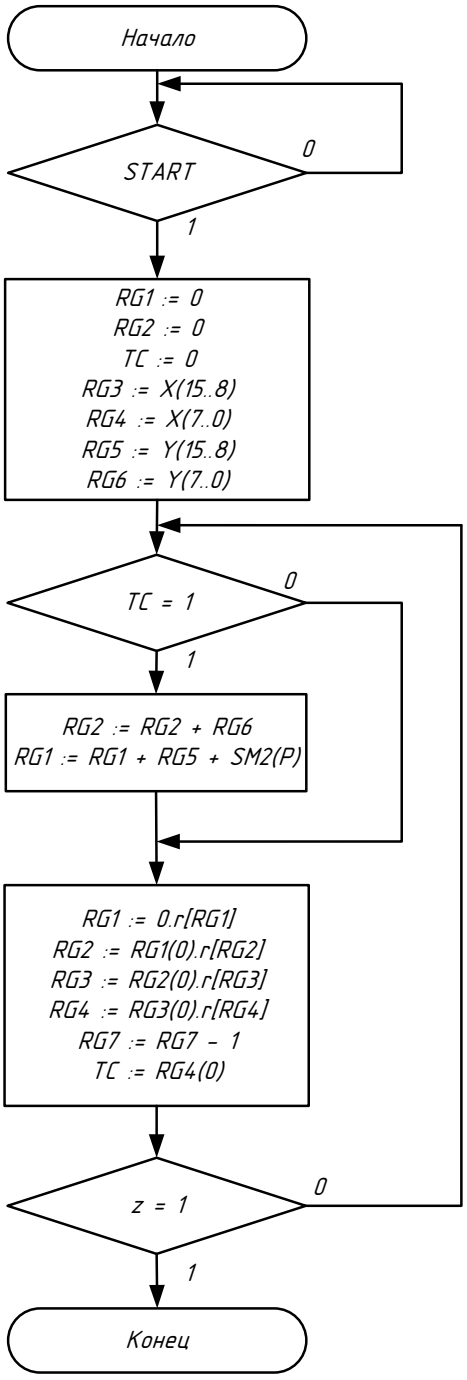


Рис. 2.9. Содержательный и закодированный микроалгоритм

Коды микроалгоритма приведены в таблице 2.1

Таблица 2.1 Кодирование сигналов операционной схемы

Система микроопераций	Вход(ы) регистра(ов)	Код микрооперации
RG1 := 0 RG2 := 0 TC := 0 RG3 := X(15..8) RG4 := X(7..0)	R, W	Y1

RG5 := Y(15..8) RG6 := Y(7..0) RG7 := 16		
RG2 := RG2 + RG6 RG1 := RG1 + RG5 + SM2(P)	W	Y2
RG1 := 0.r[RG1] RG2 := RG1(0).r[RG2] RG3 := RG2(0).r[RG3] RG4 := RG3(0).r[RG4] RG7 := RG7 - 1 TC := RG4(0)	>, -1	Y3
START		X1
TC = 1		X2
z = 1		X3

При проектировании устройств основаных на БМУ, стоит уделить внимание формату микрокоманды, который выдаётся из памяти регистра микрокоманд. Микрокоманда делится на четыре зоны: β1, β2, β3, β4. β1 отвечает за хранение адреса следующей микрокоманды, который хранится в памяти. β2 – это часть микрокоманды содержащая в себе управляющие операционной схемой сигналы. β3 – содержит в себе число отвечающее за время ожидания системы, например если какая-то микрокоманда выполняется дольше остальных. β4 – содержит в себе бит, отвечающий за обнаружение ошибки. Ошибка обнаруживается проверкой микрокоманды на парность или не парность.

Вычислим длину перечисленных зон микрокоманды.

β1: $n_a =]\log_2 64[= 6$, $n_K = 6/2 - 1 = 2$, $n_m =]\log_2 3[= 2$, $n_{\beta 1} = 5$

β2: $n_{\beta 2} = 3$

β3: $n_{\beta 3} =]\log_2 7[+ 1 = 4$

На рис. 2.10 изображён формат микрокоманды и его размещение в матричной памяти. В таблице 2.2 представлена карта настройки БМУ.

Данная схема используется в проектируемой микропроцессорной системе как арифметико-логическое устройство.

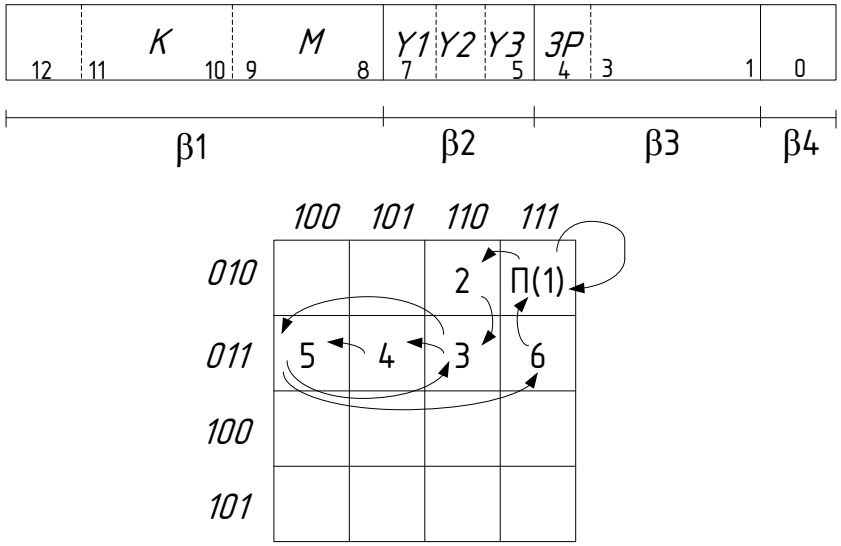


Рис. 2.10. Формат команды и размещение команд в памяти.

Таблица 2.2 Карта настройки БМУ

№ МК	Адрес МК		β1			β2			β3	β4
	Стб.	Ряд	V	K	M	Y1	Y2	Y3		
1	010	111	0	11	01	0	0	0	1111	0
2	010	110	1	01	11	1	0	0	1111	1
3	011	110	0	10	10	0	0	0	1111	1
4	011	101	0	10	00	0	1	0	0000	1
5	011	100	0	11	10	0	0	1	1111	0
6	011	111	1	10	11	0	0	0	1111	0

2.3 Проектирование Блока Деления

Структурная схема устройства для выполнения операции деления вторым способом приведена на рис. 2.11. Для проектирования устройства используются традиционные 8-ми розрядные элементы, такие как регистр и сумматор.

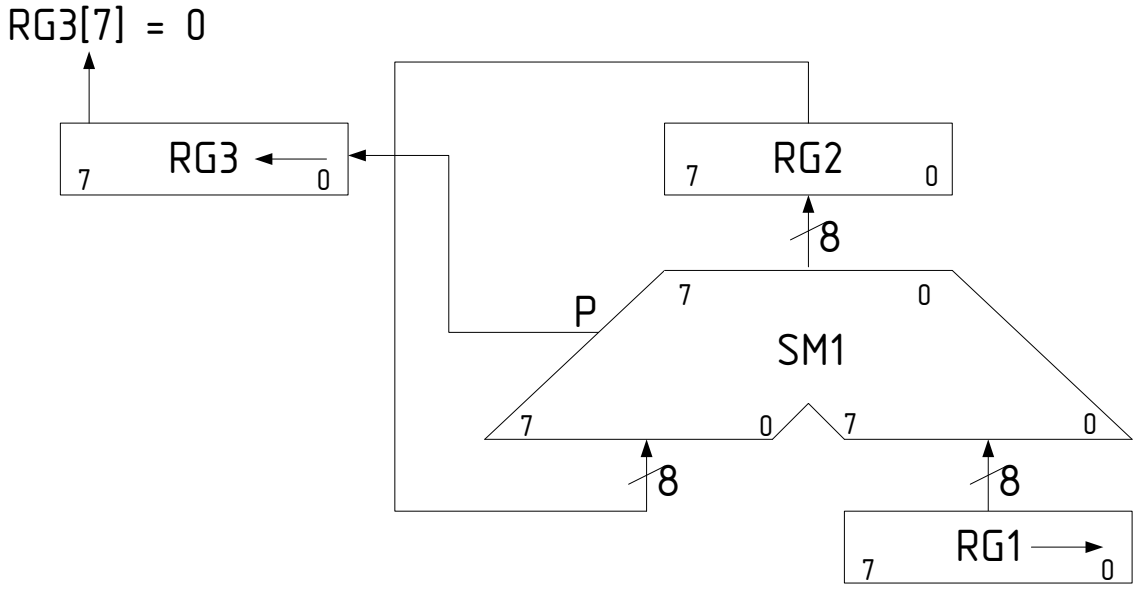


Рис. 2.11. Структурная схема устройства деления.

Максимальное время получения результата деления: $t = t_д + t_з$

, где $t_д$ – длительность микрооперации суммирования, а $t_з$ – длительность выполнения микрооперации сдвига.

Пусть мы имеем n-розрядное число результата, тогда

$$Z_i = Z_{-1}Z_{-2}Z_{-3}Z_{-4}...Z_{-n}$$

Тогда мы будем иметь, что на каждом i-том цикле виполняется неравность: $Z_i \le \frac{X}{Y} < Z_i + 2^{-i}$ => $Z_i Y \le X < Z_i Y + 2^{-i} Y$ => $0 \le (X - Z_i Y)2^i < Y$ отсюда выражение $(X - Z_i Y)2^i$ приравнивается к R_i, тоесть: $R_i = (X - Z_i Y)2^i ; 0 \le R_i < Y$

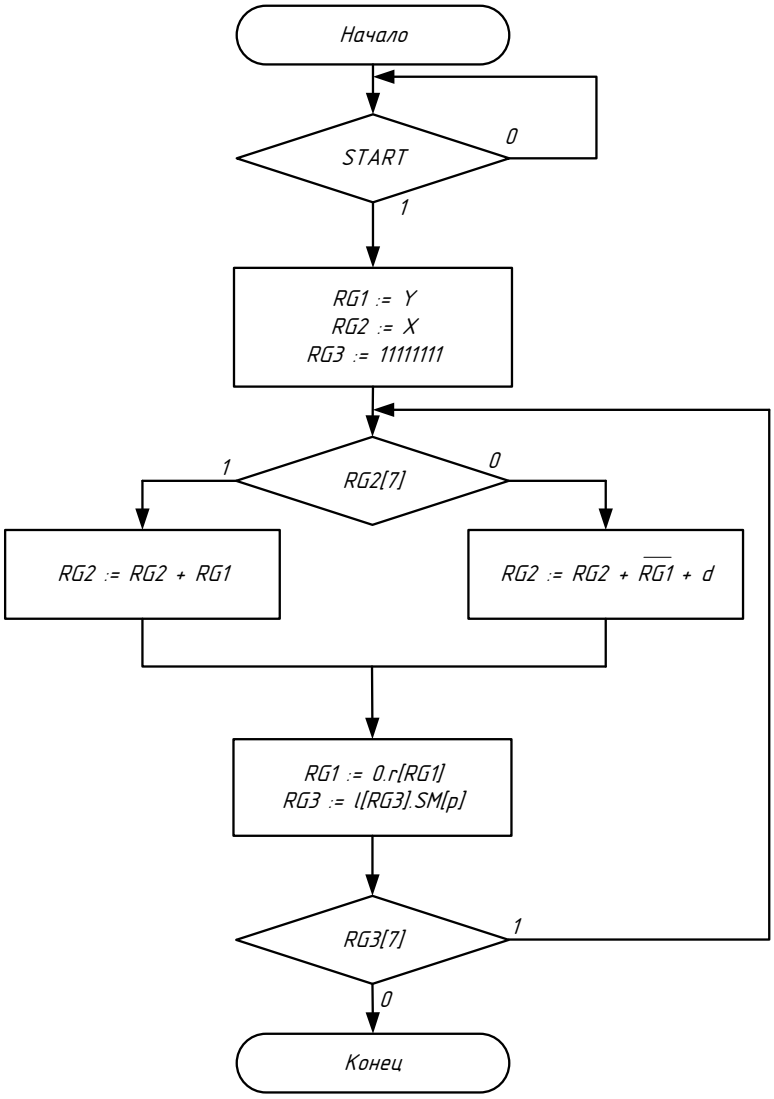


Рис. 2.12. Содержательный микроалгоритм

Для реализации операционного блока был использован метод синтезирования автомата Мура. На основе содержательного микроалгоритма, изображенного на рис. 2.12 был построен закодированный микроалгоритм, который определяет граф афтомата Мура.

В графе автомата Мура выходные сигналы изображаются в вершинах графа, а условные переходы присваивают дугам между вершинами. Так же, для кодирования, необходимо было выполнять очень важное условие, чтобы в спроектированном блоке не было «гонок». Каждое следующее состояние между вершинами должно отличатся лишь в одном бите. Граф автомата изображен на рис 2.14.

Автомат реализован с помощью D-триггера.

Таблица 2.3 Кодирование сигналов операционной схемы

Система микроопераций	Вход(ы) регистра(ов)	Код микрооперации
RG1 := Y RG2 := X RG3 := 11111111	W	Y1
RG2 := RG2 + RG1	W	Y2
RG2 := RG2 + !RG1 + d	d, W	Y3
RG1 := 0.r[RG1] RG3 := I[RG3].SM[p]	>, <, P	Y4
START		X1
RG2[7]		X2
RG3[7]		X3

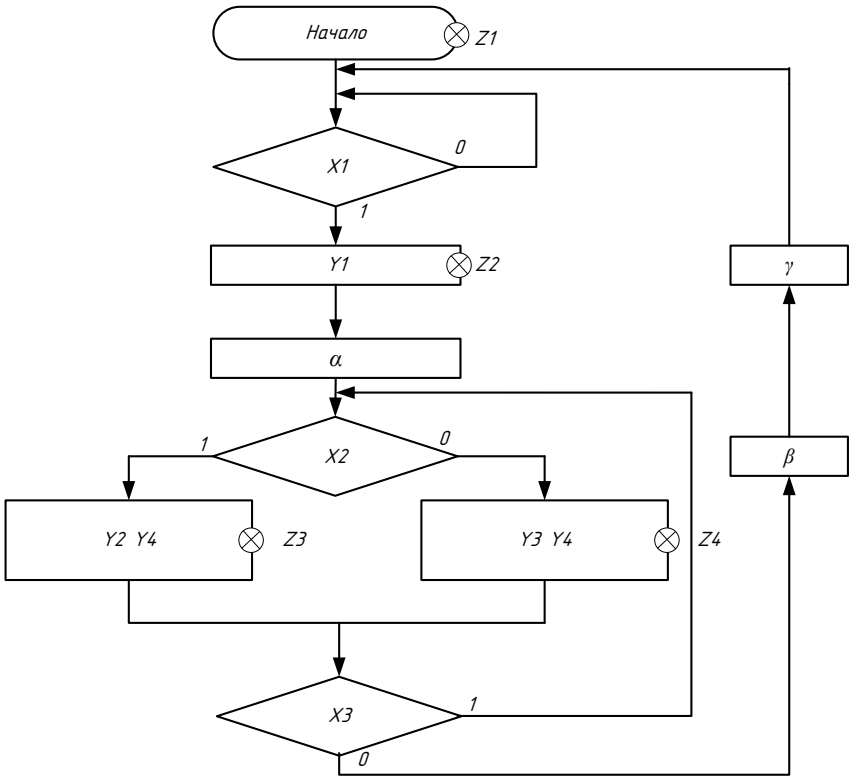


Рис. 2.13. Закодированный микроалгоритм

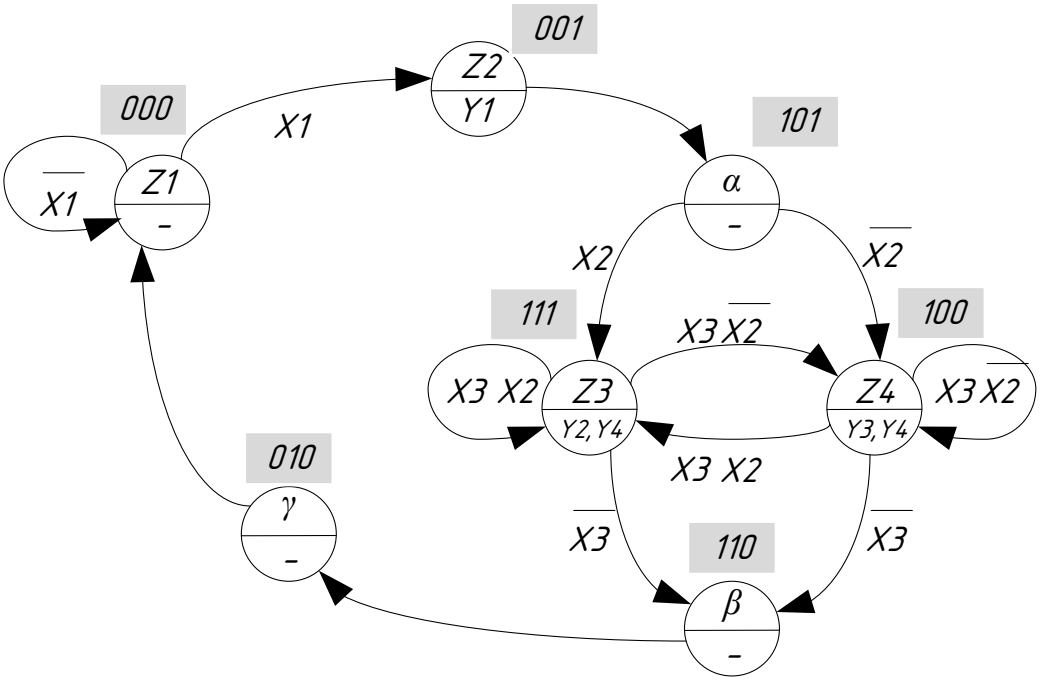


Рис. 2.14. Граф автомата Мура.

Таблица 2.4 Карта состояний и переходов между ними графа автомата.

ПС	Код ПС			СП	Код СП			Сигналы				Упр.сигналы			Тригер		
								Y1	Y2	Y3	Y4	X1	X2	X3			
Z1	0	0	0	Z1	0	0	0	0	0	0	0	0	–	–	0	0	0
Z1	0	0	0	Z2	0	0	1	0	0	0	0	1	–	–	0	0	1
Z2	0	0	1	α	1	0	1	1	0	0	0	–	–	–	1	0	1
α	1	0	1	Z3	1	1	1	0	0	0	0	–	1	–	1	1	1
α	1	0	1	Z4	1	0	0	0	0	0	0	–	0	–	1	0	0
Z3	1	1	1	Z3	1	1	1	0	1	0	1	–	1	1	1	1	1
Z3	1	1	1	Z4	1	0	0	0	1	0	1	–	0	1	1	0	0
Z3	1	1	1	β	1	1	0	0	1	0	1	–	–	0	1	1	0
Z4	1	0	0	Z4	1	0	0	0	1	1	0	–	0	1	1	0	0
Z4	1	0	0	Z3	1	1	1	0	1	1	0	–	1	1	1	1	1
Z4	1	0	0	β	1	1	0	0	1	1	0	–	–	0	1	1	0
β	1	1	0	γ	0	1	0	0	0	0	0	–	–	–	0	1	0
γ	0	1	0	Z1	0	0	0	0	0	0	0	–	–	–	0	0	0

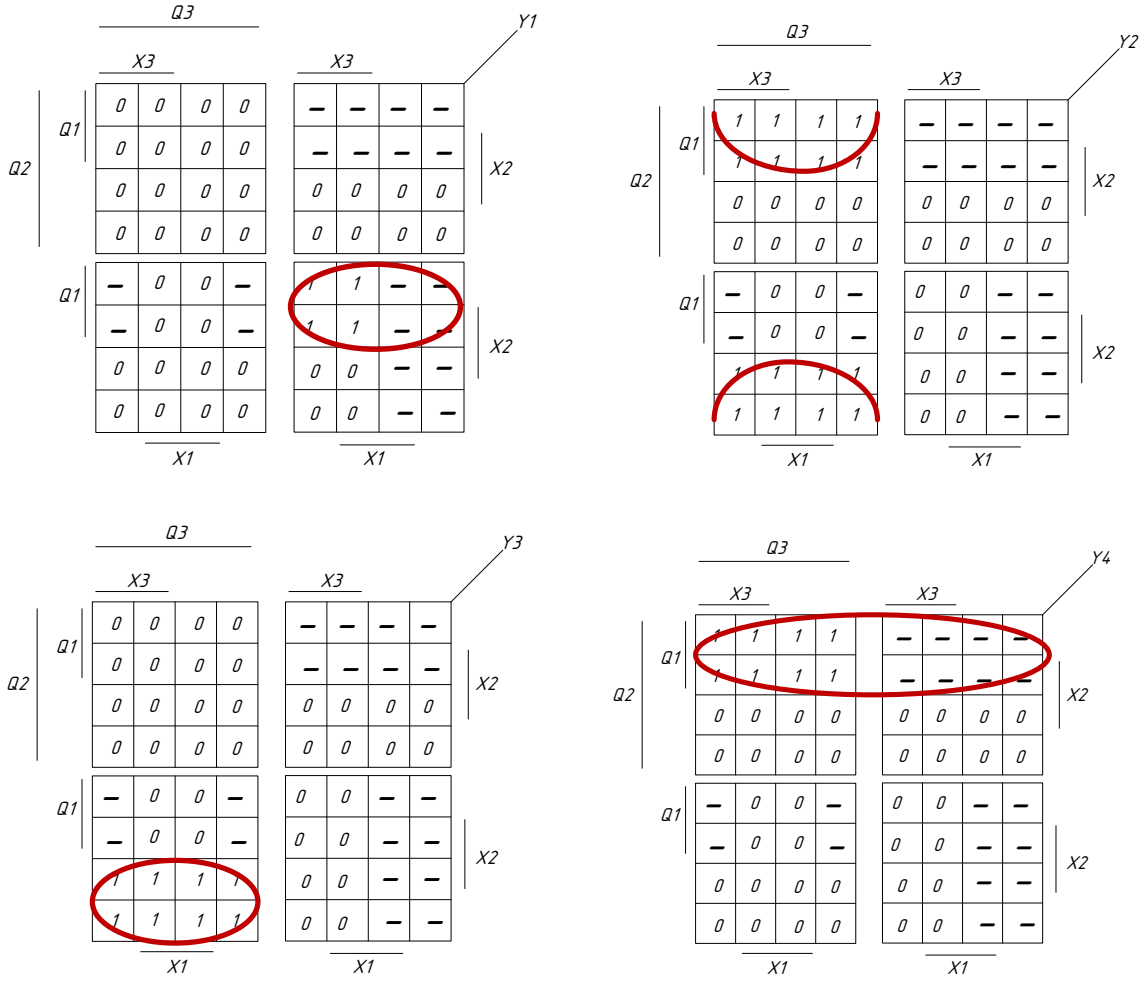


Рис. 2.14. Минимизация управляющих сигналов

В результате минимизации выполненной на рис. 2.15 и на рис 2.14 мы получаем основные функции возбуждения тригеров и выполнения ункциональной схемы.

$$y_1 = \overline{Q_3}\,\overline{Q_2}Q_1,\ y_2 = Q_3Q_2Q_1,\ y_3 = Q_3\overline{Q_2}\,\overline{Q_1},\ y_4 = Q_2Q_1$$

$$D_3 = Q_1 \vee Q_3\overline{Q_2},\ D_2 = Q_3x_2 \vee Q_3\overline{Q_1}\,\overline{x_3} \vee Q_3Q_2\overline{Q_1} \vee Q_3Q_2\overline{x_3},$$

$$D_1 = Q_3\overline{Q_2}x_3x_2 \vee \overline{Q_3}\,\overline{Q_2}\,\overline{Q_1}x_3 \vee Q_1x_3x_2 \vee \overline{Q_2}Q_1x_2 \vee Q_1Q_3$$

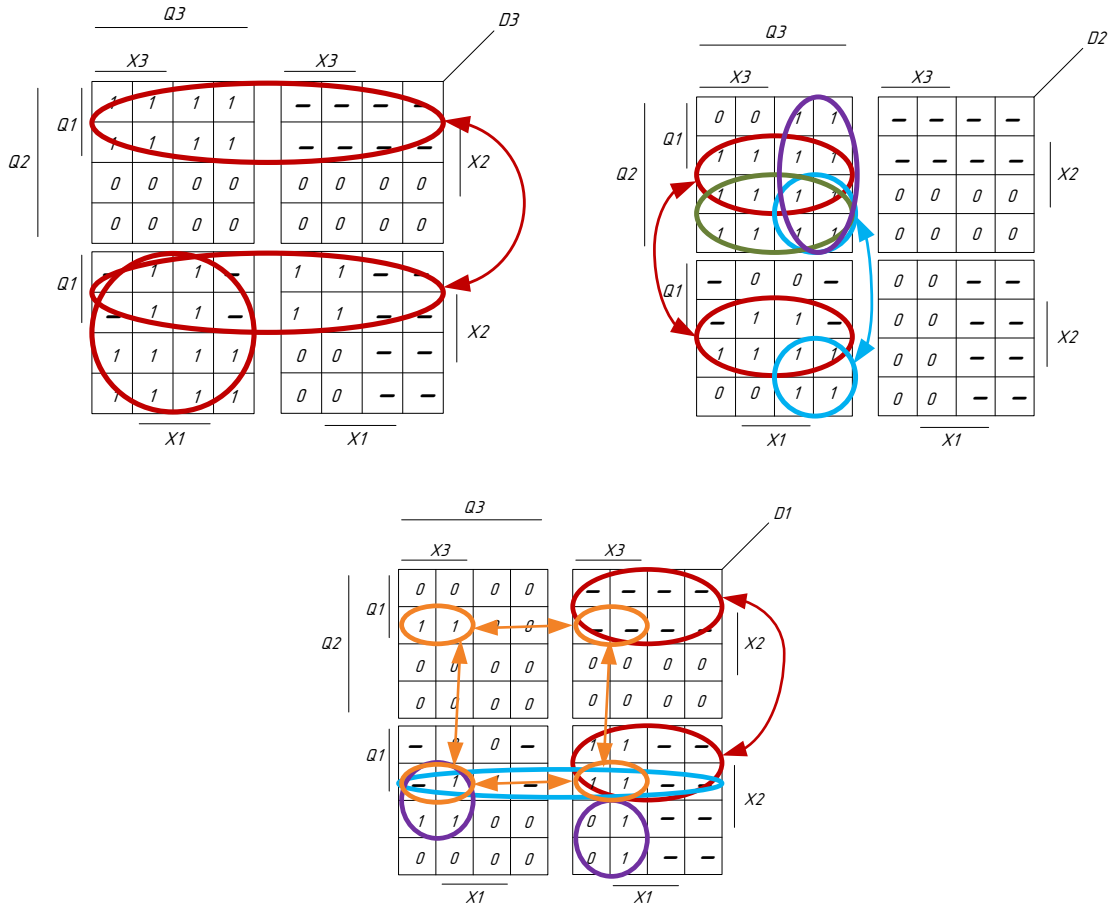


Рис. 2.15. Минимизация функций возбуждения D-тригера

2.4 Проектирование Блока вычисления квадратного корня

Структурная схема устройства для выполнения операции вычисления квадратного приведена на рис. 2.16. Для проектирования устройства используются традиционные 8-ми розрядные элементы.

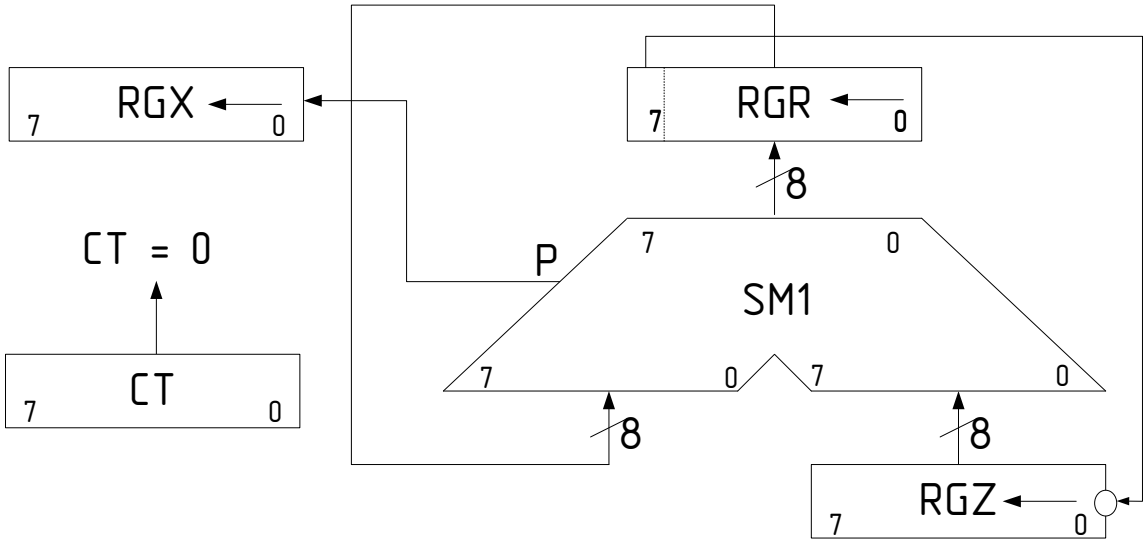
При вычислении квадратного корня, каждый розряд результата должен отвечать неравностям $Z_i \le \sqrt{X} < Z_i + 2^{-i}$ => $Z_i^{\,2} \le X < Z_i^{\,2} + 2^{-i+1} Z_i + 2^{-2i}$ => $0 \le 2^{-i-1}(X - Z_i^{\,2}) < Z_i + 2^{-i-1}$

Пусть $R_i = 2^{-i-1}(X - Z_i^{\,2})$, тогда $0 \le R_i < Z_i + 2^{-i-1}$

Для реализации операционного блока был использован метод синтезирования автомата Мили. На основе содержательного микроалгоритма, изображенного на рис. 2.16 был построен закодированный микроалгоритм (рис 2.17), который определяет граф афтомата Мура.

В графе автомата Мура выходные сигналы и условные переходы присваивают дугам между вершинами. Так же, для кодирования, необходимо было выполнять очень важное условие, чтобы в спроектированном блоке не было «гонок». Каждое следующее состояние между вершинами должно отличатся лишь в одном бите. Граф автомата изображен на рис 2.18.

Автомат реализован с помощью JK-тригера.



2.15. Структурная схема устройства вычисления квадратного корня.

Таблица 2.5 Кодирование сигналов операционной схемы

Система микроопераций	Вход(ы) регистра(ов)	Код микрооперации
RGX := X RGZ := 0 RGR := 0	W, R	Y1
RGR := RGR + !RGZ.11	d, W	Y2
RGR := RGR + RGZ.11	W	Y3
RGX:=l2[RGX].00 RGR := l2[RGR].RGX[7,6] RGZ := l1[RGZ].!RGR[7] CT := CT - 1	>, <, d, dec	Y4
START		X1
RGR[7]		X2
CT = 0		X3

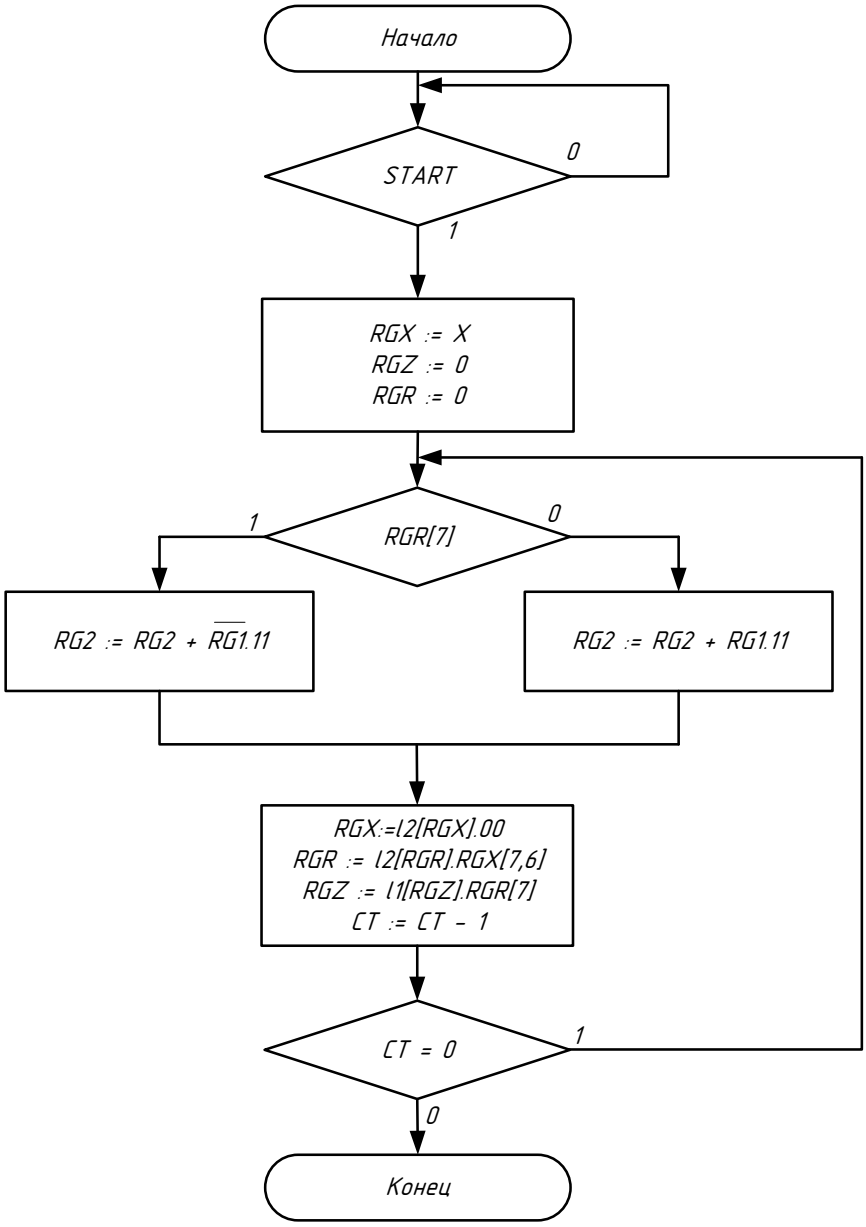


Рис. 2.16. Содержательный микроалгоритм

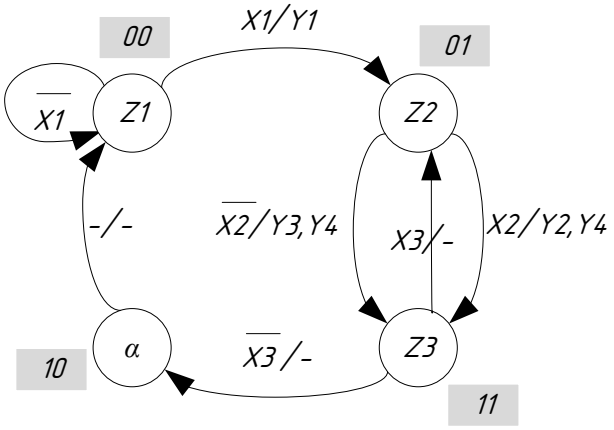


Рис. 2.18. Граф автомата Мили.

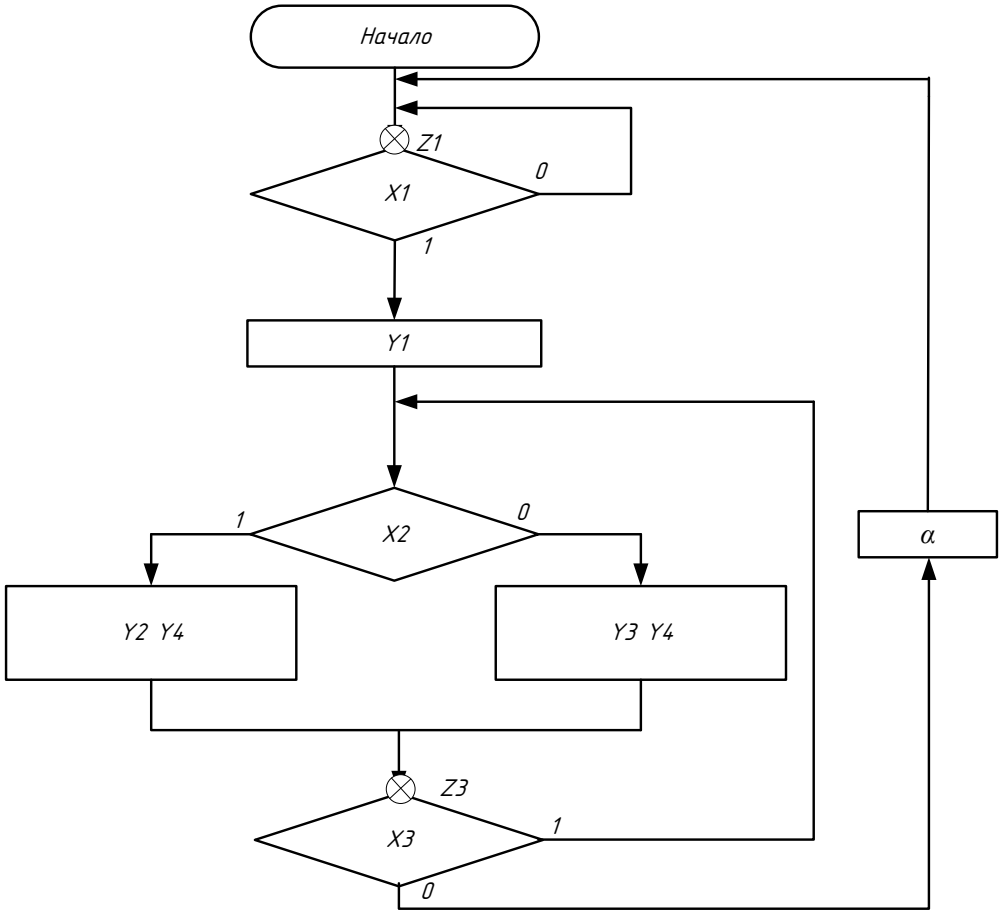


Рис. 2.17. Закодированный микроалгоритм

Таблица 2.6 Карта состояний и переходов между ними графа автомата.

ПС	Код ПС		СП	Код СП		Сигналы				Упр.сигналы			Тригер			
						Y1	Y2	Y3	Y4	X1	X2	X3				
Z1	0	0	Z2	0	1	1	0	0	0	1	–	–	0	–	1	–
Z1	0	0	Z1	0	0	0	0	0	0	0	–	–	0	–	0	–
Z2	0	1	Z3	1	1	0	1	0	1	–	1	–	1	–	–	0
Z2	0	1	Z3	1	1	0	0	1	1	–	0	–	1	–	–	0
Z3	1	1	Z2	0	1	0	0	0	0	–	–	1	–	1	–	0
Z3	1	1	α	1	0	0	0	0	0	–	–	0	–	0	–	1
α	1	0	Z1	0	0	0	0	0	0	–	–	–	–	1	0	–

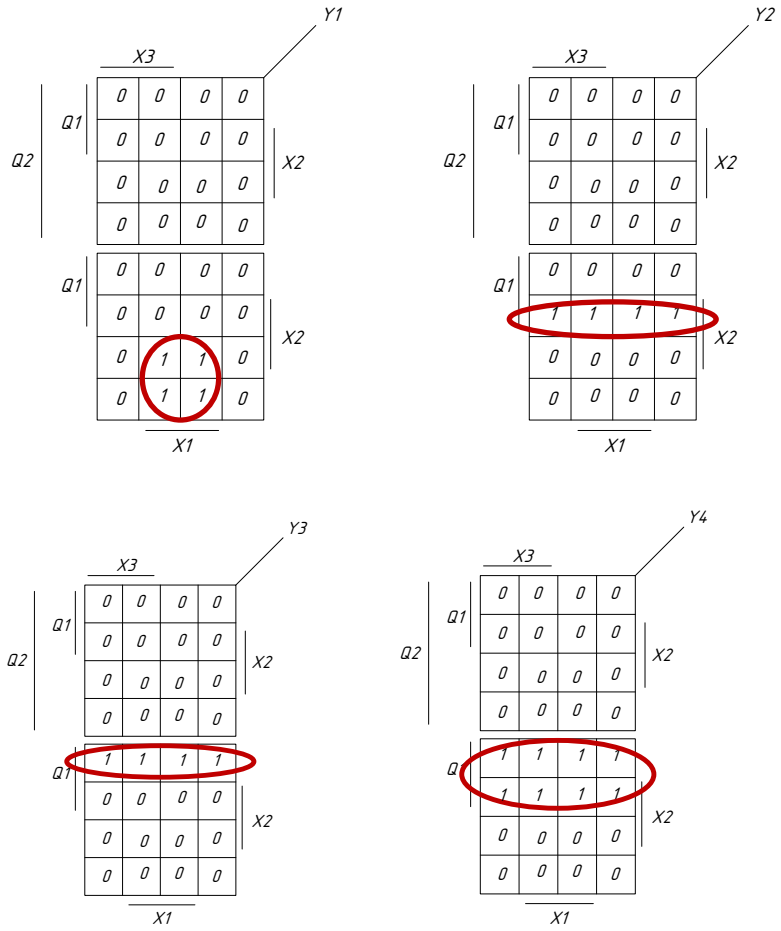


Рис. 2.19. Минимизация управляющих сигналов

В результате минимизации выполненной на рис. 2.19 и на рис 2.18 мы получаем основные функции возбуждения тригеров и выполнения функциональной схемы.

$$y_1 = \overline{Q_2}\,\overline{Q_1}x_1,\ y_2 = \overline{Q_2}Q_1x_2,\ y_3 = \overline{Q_2}Q_1\overline{x_2},\ y_4 = \overline{Q_2}Q_1$$

$$J_1 = Q_1,\ K_1 = x_3 \vee \overline{Q_1},\ J_2 = \overline{Q_2}x_1,\ K_2 = Q_2\overline{x_3}$$

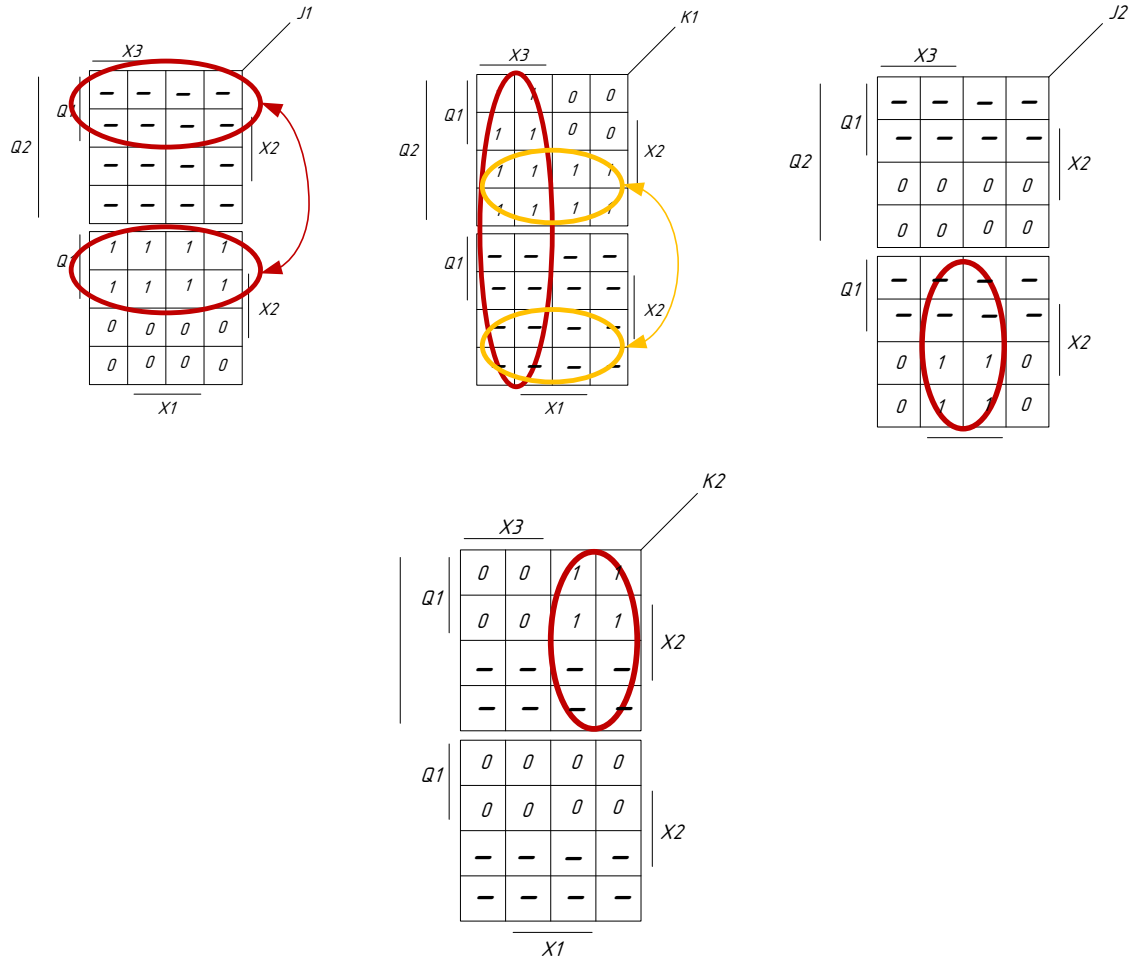


Рис. 2.18. Минимизация функций возбуждения JK-тригера.

2.5 Проектирование устройства превращения чисел

Алгоритм превращения двоичных чисел в двоично-десятичные числа в коде «8421» состоит из следующих этапов:

1. Коды всех N тетрад сдвигаются влево на один розряд с переходом двоических розрядов из каждой младшей тетрады в соседнюю старшую тетраду. Розряды двоичного числа переходят в младшую тетраду.

2. Когда из тетрады в соседнюю старшую переходит единица, или код в тетраде становится больше чем 9, то необходима коррекция, которая представляет из себя добавление к коду тетрады числа 6 с распространением переносов.

3. Цифры двоично-десятичной системы формируются в процессе сдвига в тетрадах, начиная со старших розрядов. Превращение заканчивается когда все двоические розряды (включая нули) переходят в двоично-десятичные тетрады.

Операционная схема устройства для перевода чисел из двоичного в десятичный изображена на рис.2.19. Устройство содержит регистр RGR для хранения входного двоично десятичного числа, 2 регистра (RG1 и RG2) для формирования тетрад двоично-десятичного результата, регистр RGC для хранения значений коррекции, мультиплексор MX, логические элементы и комбинационная схема КС.

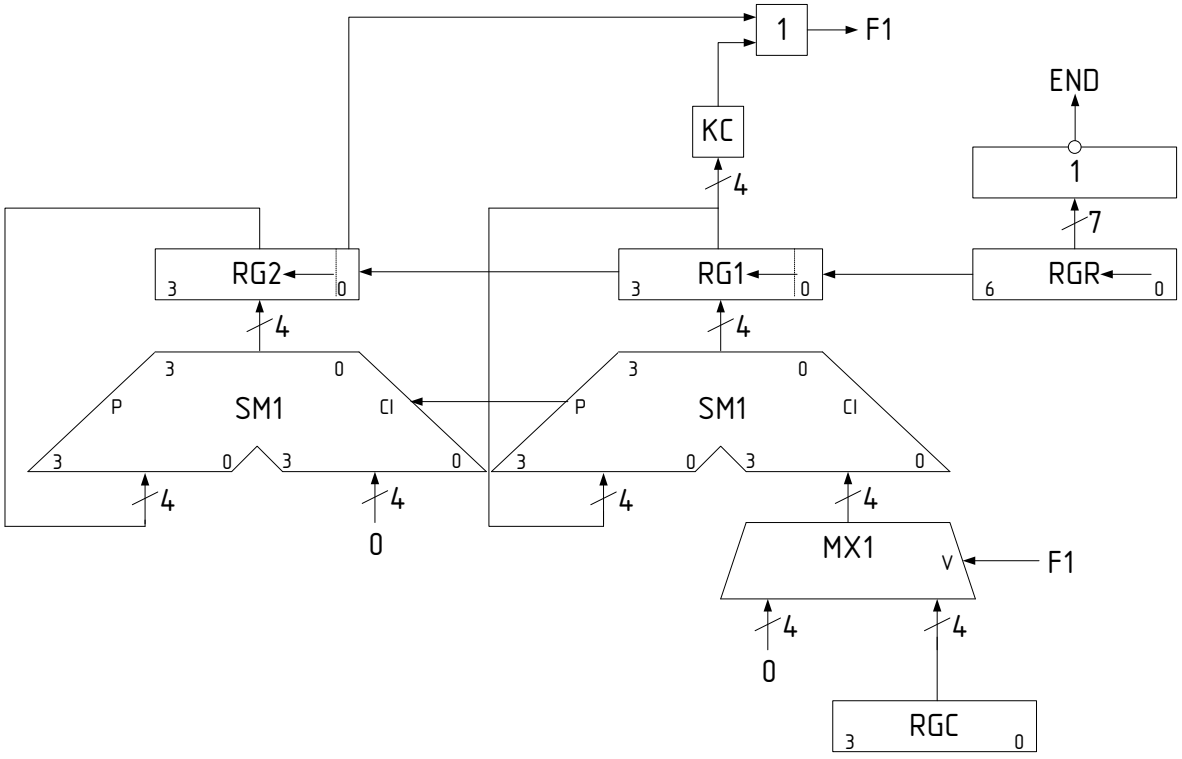


Рис. 2.19. Операционная схема устройства для перевода чисел из двоичной системы исчисления в десятичную методом «сдвига-коррекции».

К управляющим входам мультиплексора подключается вход F1 который генерируется КС. КС работает таким образом, что если в старшую тетраду переходит единичка или значение в тетраде больше нуля, то мультиплексор прибавляет число 6 (коррекция). Таблица

истиности комбинационной схемы изображена в таблице 2.7, а минимизация и результирующая схема изображена на рис. 2.20.

Таблица 2.7 Таблица истиности КС.

Розряды регистра RGT				Функция
Q4	Q3	Q2	Q1	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

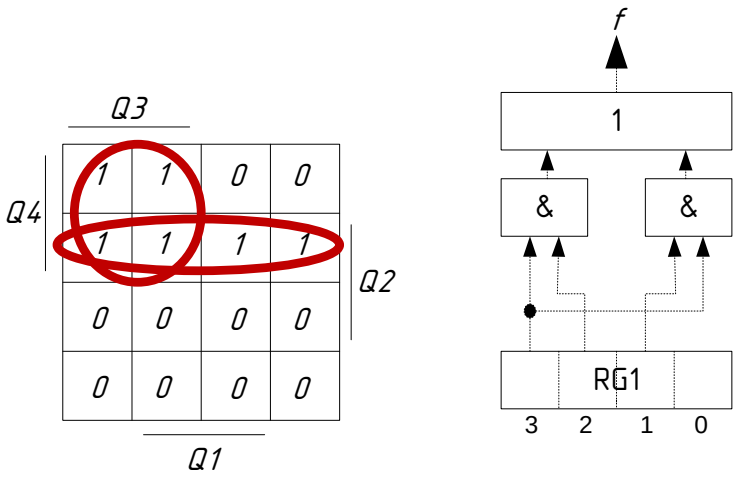


Рис. 2.20. Комбинационная схема и диаграма Вейча.

К управляющим входам мультиплексора подключается вход F1 который генерируется КС. КС работает таким образом, что если в старшую тетраду перейдёт единичка, или в регистре который хранит

значение тетраду, это значение будет больше единицы, то сигнал F1 вызовит коррекцию +6.

Для спроектированного устройства построим автомат Мура с использованием временных функций. Алгоритмы выполнения операции превращения чисел изображены на рис. 2.21, а граф автомата изображен на рис 2.22.

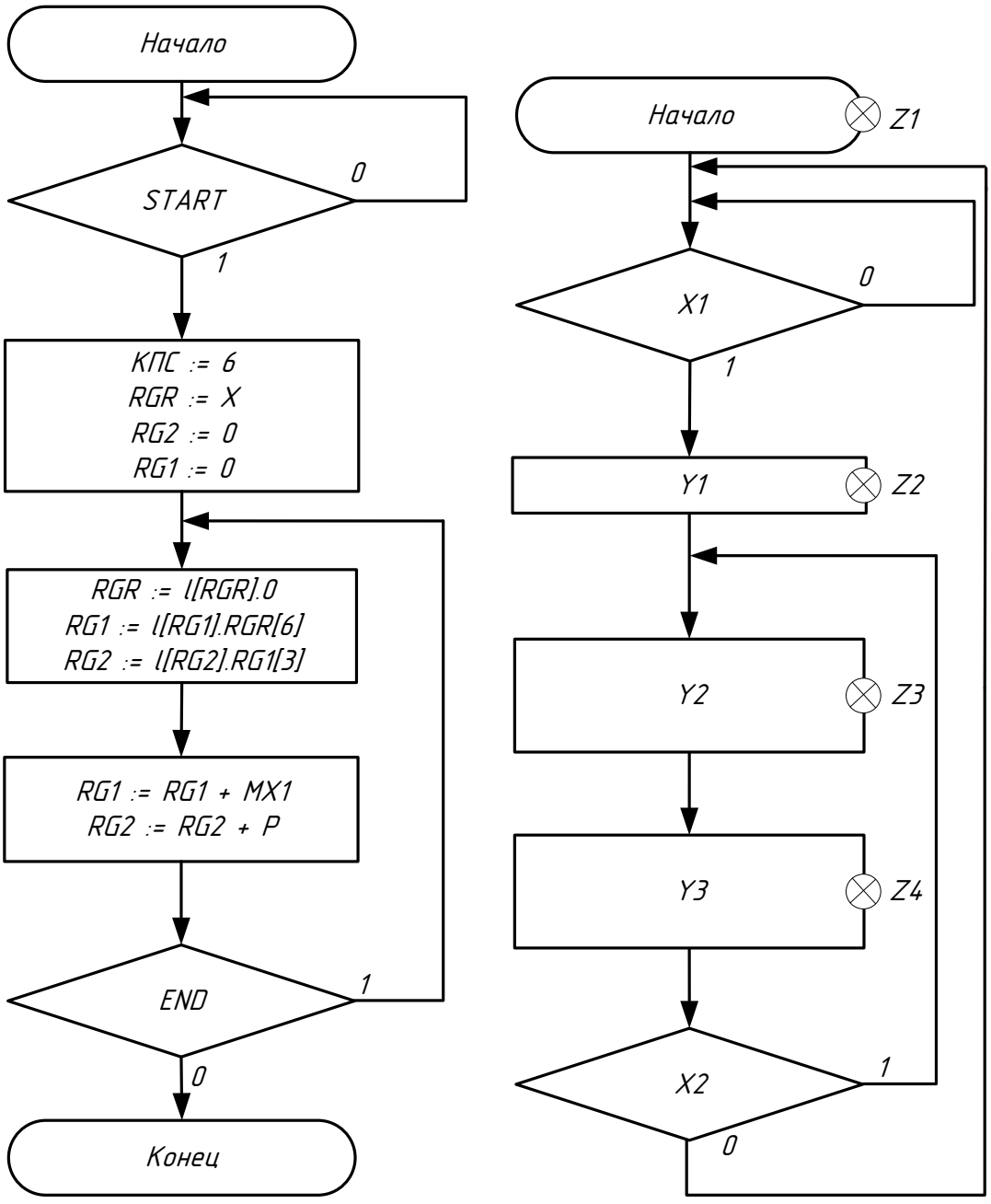


Рис. 2.21. Содержательный и закодированный микроалгоритм

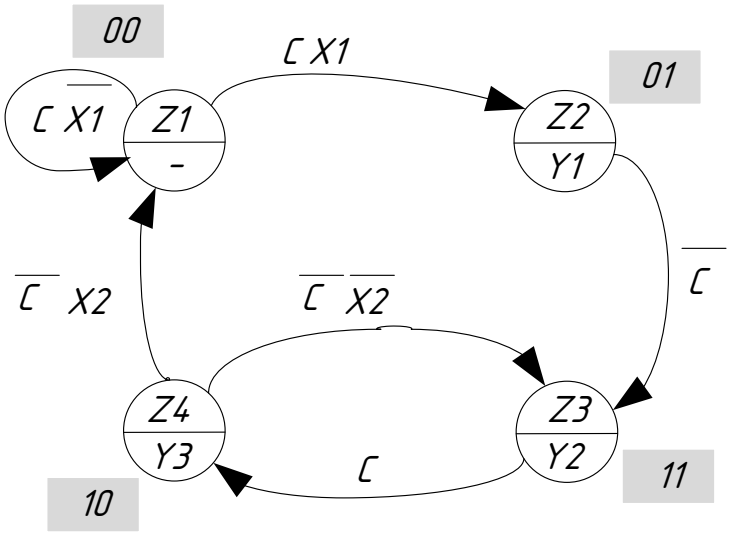


Рис. 2.22. Граф автомата

Таблица 2.8 Таблица структурного синтеза автомата использующего временные функции

ПС	Код ПС		СП	Код СП		Сигналы			Лог.условия			Времен.функции			
						Y1	Y2	Y3	C	X1	X2	F2	φ2	F1	φ1
Z1	0	0	Z2	0	1	0	0	0	1	1	-	0	0	1	0
Z1	0	0	Z1	0	0	0	0	0	1	0	-	0	0	0	0
Z2	0	1	Z3	1	1	1	0	0	0	-	-	1	0	0	0
Z3	1	1	Z4	1	0	0	1	0	1	-	-	0	0	0	1
Z4	1	0	Z3	1	1	0	0	1	0	-	0	0	0	1	0
Z4	1	0	Z1	0	0	0	0	1	0	-	1	0	1	0	0

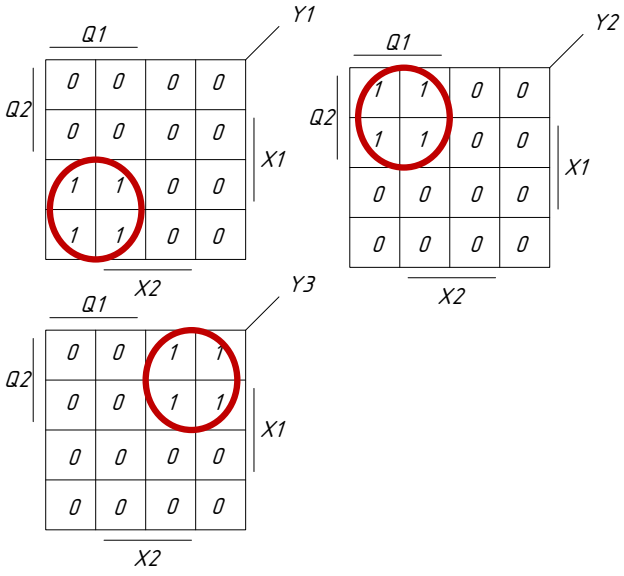


Рис. 2.23. Минимизация управляющих сигналов

В результате минимизации выполненной на рис. 2.23 и используя таблицу 2.8 мы получаем временные функции и сигналы управления функциональной схемы:

$$y_1 = \overline{Q_2}Q_1,\ y_2 = Q_2Q_1,\ y_3 = Q_2\overline{Q_1}$$

$$F_2 = \overline{Q_2}Q_1\overline{C},\ F_1 = \overline{Q_2}\,\overline{Q_1}Cx_1 \vee Q_2\overline{Q_1}\,\overline{C}\,\overline{x_2},\ \varphi_2 = Q_2\overline{Q_1}\,\overline{C}x_2,\ \varphi_1 = Q_2Q_1C,$$

$$Q_2^{t+1} = F_2 \vee Q_2^t\overline{\varphi_2} = \overline{Q_2}Q_1\overline{C} \vee Q_2^t\overline{Q_2\overline{Q_1}\,\overline{C}x_2},$$

$$Q_1^{t+1} = F_1 \vee Q_1^t\overline{\varphi_1} = \overline{Q_2}\,\overline{Q_1}Cx_1 \vee Q_2\overline{Q_1}\,\overline{C}\,\overline{x_2} \vee Q_1^t\overline{Q_2Q_1C}.$$

2.6 Проектирование устройства с плавающей запятой

Для проектирования устройства для выполнения вычислений чисел с плавающей запятой будет выполнен синтез некоторых устройств, готовых для погружения в ПЛИС. Основной алгоритм работы с числами с плавающей запятой (C = A – B) следующий:

1. Сравнивание порядков A, B. Находится самый максимальный порядок из двух чисел, и устанавливается как порядок результата. А меньший порядок увеличивается (при этом сдвигается мантисса числа вправо) до тех пор, пока не станет равным максимальному.
2. Выполняется вычисление мантисс. C = A – B.

Формат числа записанного в формате описанном в стандарте IEEE 754 изображен на рис 2.24:

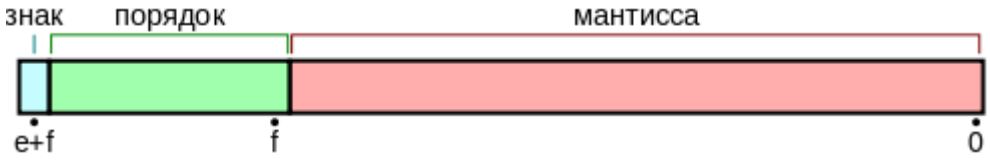


Рис 2.24. Формат числа с плавающей запятой.

Где:

f – размер мантиссы. В нашем варианте равно 5. Следует учесть, что число подается в нормализованном виде, поэтому старший бит числа, это всегда 1. Исходя из этих соображений старший бит не хранят

e – размер порядка. В нашем варинате равно 4.

Структура устройства изображена на рис 2.25, а его реализация представлена в приложении Е.

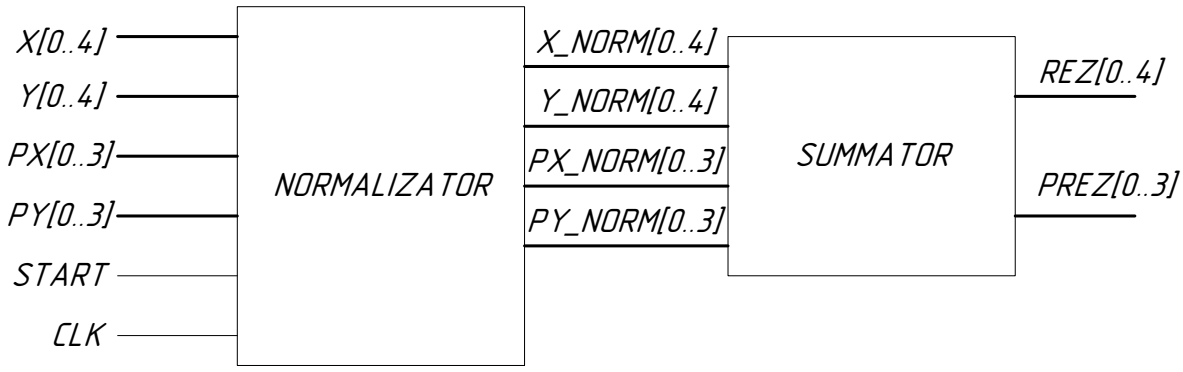


Рис 2.25. Структурная схема устройства работы с плавающей запятой

В данном случае на вход устройства отвечающего за нормализацию, входит числа X и Y (их мантиссы и порядки). Устройство сравнивает порядки и выполняет нормализацию, в том случае, если на вход START поступает сигнал равный «1». На выход устройство выдаёт нормализованные порядки и мантиссы. А сумматор уже выполняет операцию разности.

Раздел 3. Разработка микропрограмного обеспечения

3.1 Разработка стандартных микроопераций

В таблице 3.1 представлена система команд, которые на текущий момент являются стандартными и содержатся в любой ЭОМ.

Таблица 3.1 Система стандартных команд

Мнемоника	КОП	Описание команды
ADD Rx, Ry	00001	Двуадресная команда сложения. Выполняет действие Rx := Rx + Ry
SUB Rx, Ry	00010	Двуадресная команда разности. Выполняет действие Rx := Rx – Ry
ADDC Rx, C	00011	Двуадресная команда сложения с константой. Выполняет действие Rx := Rx + C
SUBC Rx, C	00100	Двуадресная команда разности с константой. Выполняет действие Rx := Rx – C
IN port	00101	Одноадресная команда ввода с порта в аккумулятор (R15)
OUT port	00110	Одноадресная команда вывода с аккумулятора (R15) в порт
JUMP met	00111	Безусловній переход на метку по адресу met
JMPZ met	01000	Прыжок по условию, если Z = 1, на адрес met
JMPC met	01001	Прыжок по условию, если C = 1, на адрес met
MOVC C	01010	Загрузить в регистр R15 константу R15 := C
MOV Rx, Ry	01011	Загрузить в регистр Rx содержимое регистра Ry. Rx := Ry
MOVA Rx	01100	Загрузить в регистр Rx содержимое аккумулятора (R15)

Реализация микрокоманд представлена в приложении Г, а блок-схема алгоритма в приложении А. Данная ЭОМ предоставляет пользователю два банка регистров по 16 регистров (R0 – R15). В разработанной системе микрокоманд и реализованном микропрограмном

обработчике для пользователя предоставлены регистры R5 – R15. R15 играет роль аккумулятора, а остальные регистры играют важную роль в обработке микрокоманд. Регистр R0 буфферный, используется для распаковки микрокоманды. Регистр R1 играет роль Счётчика Команд, и в нём содержится указатель на адрес микрокоманды в памяти. Регистр R1 – Регистр микрокоманды. В нём лежит текущая загруженная микрокоманда. В регистрах R3, R4 лежат адреса операндов (регистров).

Формат команди	*Код операції*	*Тип адресації*	*Адреса в ОП*
15	*14..11*	*10*	*9..0*

	Тип адресации	*Обозначение*
0	*Пряма*	*П*
1	*Непряма*	*К*

	Формат команди
0	*Одноадресна*
1	*Двоадресна*

Рис 3.1 Формат одноадресной команды.

Формат команди	*Код операції*	*ТА 1*	*РОН 1*	*ТА 2*	*РОН 2*
15	*14..10*	*9, 8*	*7..5*	*4, 3*	*2..0*

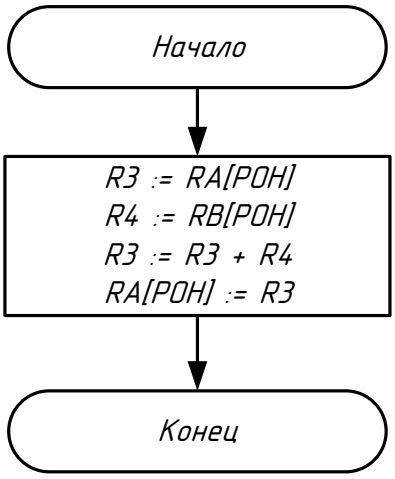
	Тип адресації	*Позначення*
00	*Пряма регистрова*	*ПР*
01	*Непряма регистрова*	*КР*
10	*Автоінкрементна*	*АІ*
11	*Автодекрементна*	*АД*

	Формат команди
0	*Одноадресна*
1	*Двоадресна*

Рис 3.2 Формат двухадресной команды.

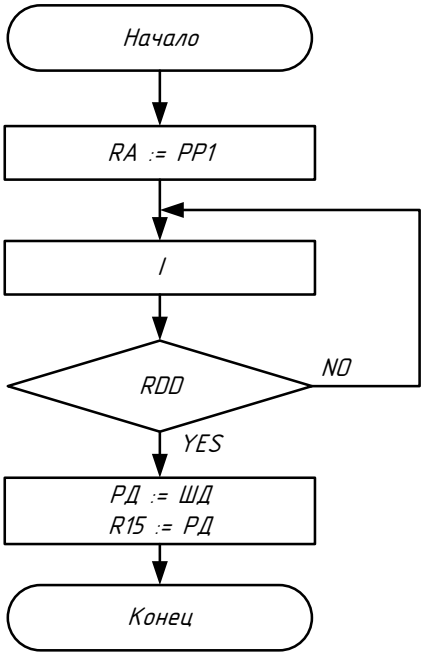
Для примера рассмотрим блок-схемы и обработку разработанных команд, таких так ADD Rx, Ry, IN port, OUT port, JUMP met и JMPC met.

Блок-схемы алгоритмов и листинг микрокоманд изображено на рисунках 3.3, 3.4, 3.5, 3.6 и 3.7 соответственно




```
ADDD
{XOR R0, R0;}
{OR R0, RA, z;}
{ADD R3, R3, R4, z;}
writeles1
{EWH; OEY; XOR nil, R0, R0;}
{EWL; OEY; OR nil, R0, R0;}
{CJP rdm, writeles1; W; OR nil, R3, z; OEY;}
{CJP NZ, next;}
```

Рис 3.3 Блок-схема алгоритма и листинг микрокоманды ADD Rx, Ry.




```
IN
{OR nil, R3, z; EWL; OEY;}
PC
{I; CJP rdd, PC; OR R15, BUS_D, z;}
{CJP NZ, next;}
```

Рис 3.4 Блок-схема алгоритма и листинг микрокоманды IN port.

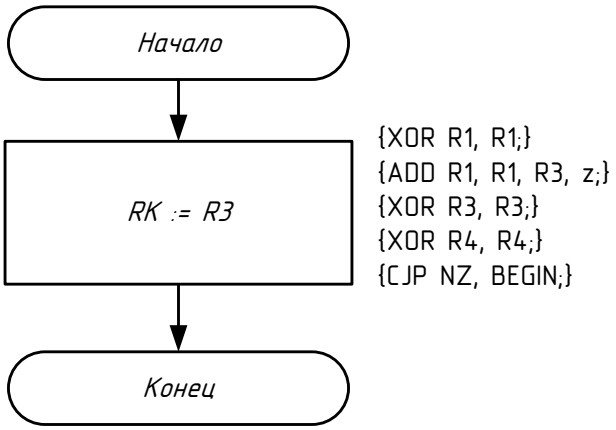


Рис 3.5 Блок-схема алгоритма и листинг микрокоманды JUMP met.

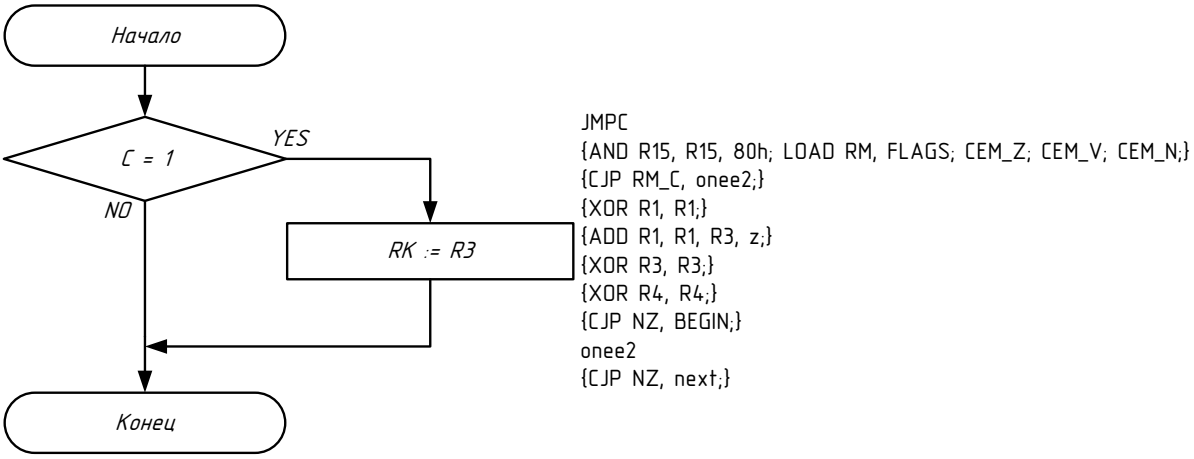


Рис 3.6 Блок-схема алгоритма и листинг микрокоманды JMPC met.

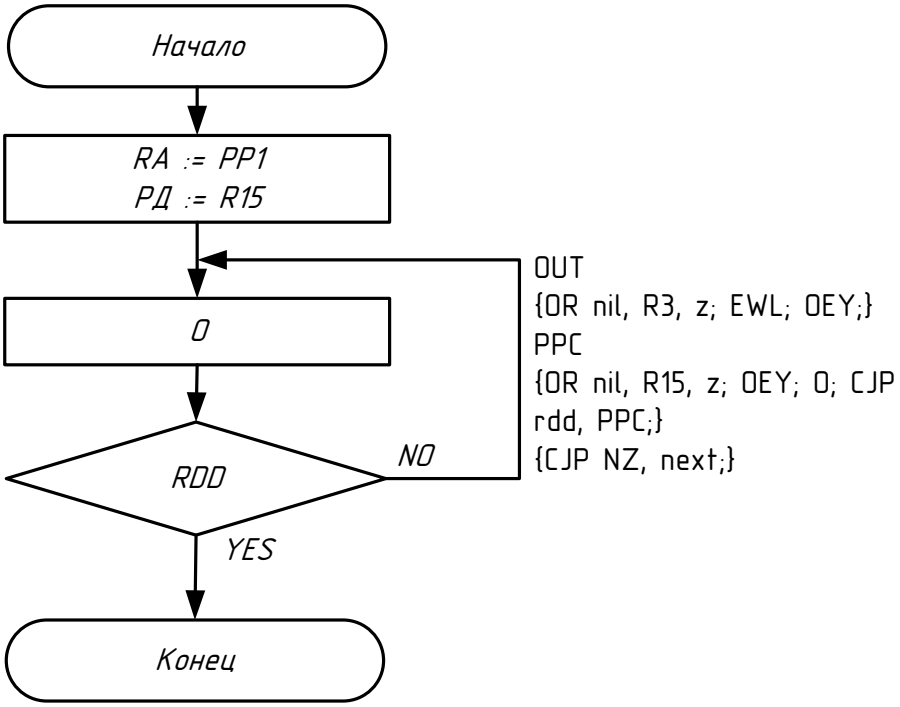


Рис 3.7 Блок-схема алгоритма и листинг микрокоманды OUT port.

3.2 Оценка достаточности разработанных микроопераций

Для того, чтобы определить достаточность разработанных микроопераций, необходимо рассмотреть функцию, которую нам необходимо реализовать:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots + (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!} + \ldots$$

Разработаем блок-схему алгоритма вычисления данной функции. Блок-схема алгоритма вычисления функции изображена на рис 3.8.

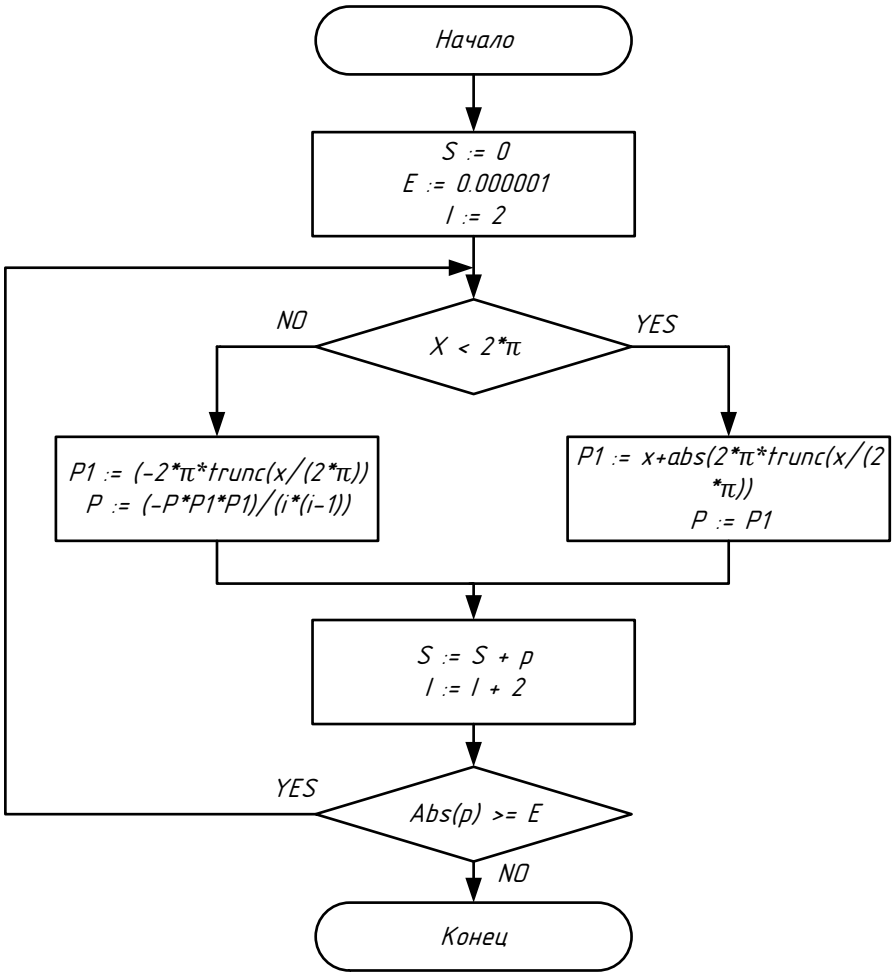


Рис 3.8 Блок-схема алгоритма вычисления функции sin(x).

Исходя из блок-схемы алгоритма вычисления функции становится понятно, что разработанного в пункте 3.1 базиса микроопераций не

достаточно для того, чтобы решить данную задачу. Решение данной задачи требует разработку дополнительных микроопераций и микроопераций для работы с числами с плавающей запятой.

3.3 Разработка необходимых микроопераций

Для реализации алгоритма необходимо реализовать работу с плавающей запятой и дополнительные микрооперации. Все дополнительные разработанные микрооперации представлены в таблице 3.2.

Таблица 3.2 Система специализированных команд

Мнемоника	КОП	Описание команды
Дополнительные микрокоманды		
CLR Rx	01101	Очищение содержимого Rx. Выполняет действие Rx := 0 или Rx := Rx xor Rx
DEC Rx	01110	Декремент содержимого Rx. Выполняет действие Rx := Rx – 1
Микрокоманды для работы с числами с плавающей запятой		
FMUL Rx, Ry	01111	Двуадресная команда умножения чисел с плавающей запятой. Выполняет действие Rx := Rx * Ry
FADD Rx, Ry	10000	Двуадресная команда суммирования чисел с плавающей запятой. Выполняет действие Rx := Rx + Ry
FDIV Rx, Ry	10001	Двуадресная команда деления чисел с плавающей запятой. Выполняет действие Rx := Rx / Ry
TRUNC Rx	10010	Выделение целой части числа лежащего в Rx
FABS Rx	10011	Вычисление модуля значения числа в Rx
FCMP Rx, Ry	10100	Сравнение чисел в Rx и Ry (установка C)
FLOAD Rx,m, p	10101	Загрузка числа с плавающей запятой в Rx, где m – мантисса, а p – порядок.
FMOV Rx, Ry	10110	Загрузить в регистр Rx содержимое регистра Ry. Rx := Ry

Разработанная система команд хоть и не является универсальной, однако может использовается для решения других вычислительных задач. Разработанная программа вычисления функции sin(x) представлена в приложении В.

Раздел 4. Разработка программного обеспечения

4.1 Разработка программного обеспечения

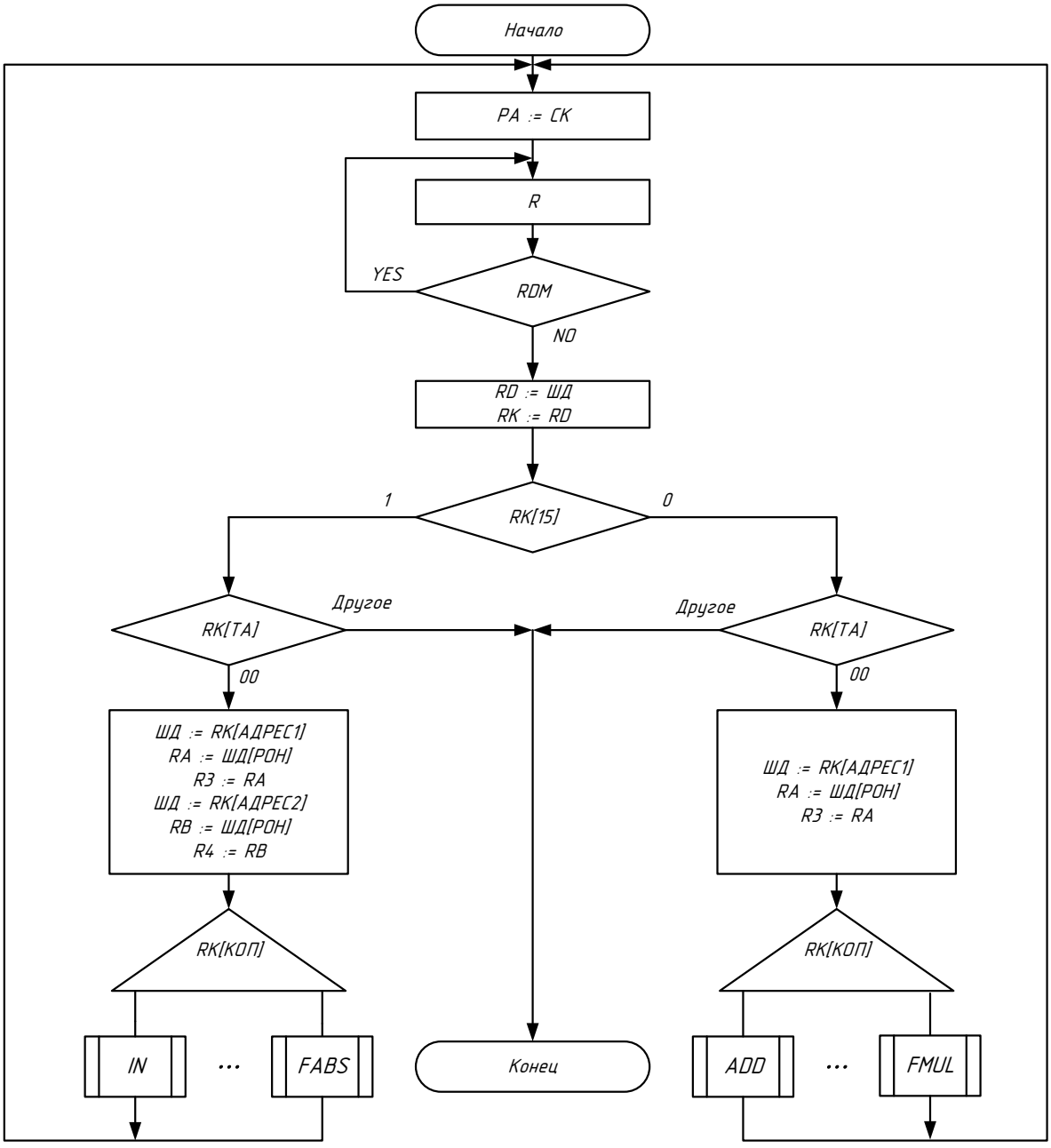
Программное обеспечение ЭОМ должно содержать доступный для пользователя интерфейс для работы. Тоесть имеется в виду, что пользователь должен иметь возможность воспользовается любым из спроектированных блоков микропроцессорной системы. Листинг разработанной программы представлен в приложении Д, а блок-схема в приложении Б.

4.2 Принцип работы спроектированного устройства

В зависимости от команды пользователя система посылает сигнал с порта Р1 на дешифратор, который в зависимости от сигнала с Р1 посылает сигнал на выбранный блок. Каждый разработанный блок работает в режиме ожидания сигнала, который, после того, как с дешифратора прийдёт сигнал, запускается на выполнение, а после завершения возвращается в режим ожидания. После того, как посылается сигнал на дешифратор, через порт Р2 отправляется операнд X, а через порт Р3 отправляется операнд У, если функциональный блок его требует. Важно заметить, что существует проблема синхронизации, когда управляющий автомат сработает быстрее, чем микропроцессор подаст необходимую информацию.

В микропроцессорной системе вшита внутренняя микропрограмма вычисления синуса. Структура спроектированной ЭОМ изображено на ИАЛЦ 462637.003 Э1.

Приложение А. Блок-схема алгоритма обработчика микрокоманд



Приложение Б. Блок-схема алгоритма основной программы

Приложение В. Листинг обработчика микрокоманд

```
link l3:CT
link l1:rdm
link l2:rdd
link ewh:16
link m:z,z,z,z,z,z,z,z,14,13,12,11 \mask of operation code

accept dev[1]:i, 22h, 24h, 3, 12
accept dev[2]:o, 0A2h, 0A4h, 3, 12
accept dev_buf[1]:0F45Bh

accept r1:2  \counter of command
accept r2:0  \registr command
\r3 - pervui operand
\r4 - vtoroj operand
\r0 - zaregeno

{CJP NZ, BEGIN;}
org 000000000001% {CJP NZ, ADDD;}
org 000000000010% {CJP NZ, SUBD;}
org 000000000011% {CJP NZ, ADDC;}
org 000000000100% {CJP NZ, SUBC;}
org 000000000101% {CJP NZ, IN;}
org 000000000110% {CJP NZ, OUT;}
org 000000000111% {CJP NZ, JUMP;}
org 000000001000% {CJP NZ, JMPZ;}
org 000000001001% {CJP NZ, JMPC;}
org 000000001010% {CJP NZ, MOVC;}
org 000000001011% {CJP NZ, MOV;}
org 000000001100% {CJP NZ, MOVA;}
org 000000001101% {CJP NZ, CLR;}
org 000000001110% {CJP NZ, DEC;}
org 000000001111% {CJP NZ, FMUL;}

BEGIN
\1) choosing the command\
{EWH; OEY; XOR nil, R1, R1;} \null page
{EWL; OEY; OR nil, R1, z;}   \into EWL adress of command
{R; OR R2, BUS_D, z;} \reading the command from data

\2) raspakovka ka commnad\
\format adressu
{ADD R0, R0, R2, z;}
{AND R0, R0, 8000h; LOAD RM, FLAGS; CEM_C; CEM_V; CEM_N;}
{CJP not RM_Z, twoadress;}  \if twoadress command => exit
{XOR R0, R0;}
{CJP NZ, odnoadress;}

twoadress
\adressation type
{ADD R0, R0, R2, z;}
```

```
{AND R0, R0, 00000001100011000%; LOAD RM, FLAGS; CEM_C; CEM_V; CEM_N;}
{CJP not RM_Z, exitt;} \if non-straight adressation => exit
{XOR R0, R0;}

\loading operands
{ADD R0, R0, R2, z;}
{AND R0, R0, 111%;}
{EWA; OEY; OR nil, R0, R0; LOAD RA;}
{XOR R0, R0;}
{ADD R0, R0, R2, z;}
{AND R0, R0, 11100000%;}
{OR SRL, R0, R0, z;}
{OR SRL, R0, R0, z;}
{OR SRL, R0, R0, z;}
{OR SRL, R0, R0, z;}
{OR SRL, R0, R0, z;}
{EWB; OEY; OR nil, R0, R0; LOAD RB;}
{OR R3, RA, z;}
{OEY; OR RQ, RB, z;}
{OR R4, RQ, z;}

\proverka na kod OP, esli ravno, do\
{ADD R0, R0, R2, z;}
{AND RQ, R0, 0111100000000000%;}
{JMAP; OR RQ, z; OEY;} \if adr then sum

odnoadress
\adressation type
{ADD R0, R0, R2, z;}
{AND R0, R0, 400h; LOAD RM, FLAGS; CEM_C; CEM_V; CEM_N;}
{CJP not RM_Z, exitt;} \if non-straight adressation => exit
{XOR R0, R0;}

\3) loading the operand
{XOR R0, R0;}
{ADD R0, R0, R2, z;}
{AND R0, R0, 0000001111111111%;}
{EWH; OEY; XOR nil, R0, R0;} \v R0 lejit adres operanda
{EWL; OEY; OR nil, R0, z;}    \mi ego gruzim v R3 (operand)
{R; OR R3, BUS_D, z;}

\proverka na kod OP, esli ravno, do\
{ADD R0, R0, R2, z;}
{AND RQ, R0, 0111100000000000%;}
{JMAP; OR RQ, z; OEY;}


\OBRABOTKA KOMAND

ADDD
{XOR R0, R0;}
{OR R0, RA, z;}
```

```
{ADD R3, R3, R4, z;}
writeles1
{EWH; OEY; XOR nil, R0, R0;}
{EWL; OEY; OR nil, R0, R0;}
{CJP rdm, writeles1; W; OR nil, R3, z; OEY;}
{CJP NZ, next;}

SUBD
{XOR R0, R0;}
{OR R0, RA, z;}
{SUB R3, R3, R4, z;}
writeles2
{EWH; OEY; XOR nil, R0, R0;}
{EWL; OEY; OR nil, R0, R0;}
{CJP rdm, writeles2; W; OR nil, R3, z; OEY;}
{CJP NZ, next;}

ADDC
{XOR R0, R0;}
{OR R0, RA, z;}
{ADD R0, R0, R2, z;}
{AND R0, R0, 111%;}
{ADD R3, R3, R0, z;}
writeles3
{EWH; OEY; XOR nil, R0, R0;}
{EWL; OEY; OR nil, R0, R0;}
{CJP rdm, writeles3; W; OR nil, R3, z; OEY;}
{CJP NZ, next;}

SUBC
{XOR R0, R0;}
{OR R0, RA, z;}
{ADD R0, R0, R2, z;}
{AND R0, R0, 111%;}
{SUB R3, R3, R0, z;}
writeles4
{EWH; OEY; XOR nil, R0, R0;}
{EWL; OEY; OR nil, R0, R0;}
{CJP rdm, writeles4; W; OR nil, R3, z; OEY;}
{CJP NZ, next;}

IN
{OR nil, R3, z; EWL; OEY;}
PC
{I; CJP rdd, PC; OR R15, BUS_D, z;}
{CJP NZ, next;}

OUT
{OR nil, R3, z; EWL; OEY;}
PPC
{OR nil, R15, z; OEY; O; CJP rdd, PPC;}
{CJP NZ, next;}
```

```
JUMP
{XOR R1, R1;}
{ADD R1, R1, R3, z;}
{XOR R3, R3;}
{XOR R4, R4;}
{CJP NZ, BEGIN;}

JMPZ
{AND R15, R15, 80h; LOAD RM, FLAGS; CEM_C; CEM_V; CEM_N;}
{CJP RM_Z, onee1;}
{XOR R1, R1;}
{ADD R1, R1, R3, z;}
{XOR R3, R3;}
{XOR R4, R4;}
{CJP NZ, BEGIN;}
onee1
{CJP NZ, next;}

JMPC
{AND R15, R15, 80h; LOAD RM, FLAGS; CEM_Z; CEM_V; CEM_N;}
{CJP RM_C, onee2;}
{XOR R1, R1;}
{ADD R1, R1, R3, z;}
{XOR R3, R3;}
{XOR R4, R4;}
{CJP NZ, BEGIN;}
onee2
{CJP NZ, next;}

MOVC
{XOR R15, R15;}
{XOR R0, R0;}
{ADD R0, R0, R2, z;}
{AND R0, R0, 0000001111111111%;}
{ADD R15, R15, R0, z;}
{CJP NZ, next;}

MOV
{XOR R0, R0;}
{OR R0, RA, z;}
{XOR R3, R3;}
{ADD R3, R3, R4, z;}
writeles6
{EWH; OEY; XOR nil, R0, R0;}
{EWL; OEY; OR nil, R0, R0;}
{CJP rdm, writeles6; W; OR nil, R3, z; OEY;}
{CJP NZ, next;}

MOVA
{XOR R0, R0;}
{OR R0, RA, z;}
```

```
{ADD R0, R0, R2, z;}
{AND R0, R0, 11100000%;}
{OR SRL, R0, R0, z;}
{OR SRL, R0, R0, z;}
{OR SRL, R0, R0, z;}
{OR SRL, R0, R0, z;}
{OR SRL, R0, R0, z;}
{ADD R4, R4, R15, z;}
writeles7
{EWH; OEY; XOR nil, R0, R0;}
{EWL; OEY; OR nil, R0, R0;}
{CJP rdm, writeles7; W; OR nil, R4, z; OEY;}
{CJP NZ, next;}

CLR
{XOR R0, R0;}
{OR R0, RA, z;}
{XOR R3, R3;}
writeles8
{EWH; OEY; XOR nil, R0, R0;}
{EWL; OEY; OR nil, R0, R0;}
{CJP rdm, writeles8; W; OR nil, R3, z; OEY;}
{CJP NZ, next;}

DEC
{XOR R0, R0;}
{OR R0, RA, z;}
{SUB R3, R3, z, z; LOAD RM, FLAGS; CEM_C;}
writeles8
{EWH; OEY; XOR nil, R0, R0;}
{EWL; OEY; OR nil, R0, R0;}
{CJP rdm, writeles8; W; OR nil, R3, z; OEY;}
{CJP NZ, next;}

next
{ADD R1, R1, 2h, z;}
{XOR R3, R3;}
{XOR R4, R4;}
{CJP NZ, BEGIN;}

exitt
{}
```

Приложение Г. Листинг подпрограммы вычисления sin(x)

```
--R8 => S, R9 => E, R10 => I, R11 => X, R12 => 2*Pi, R5 => P, R13 => P1
CLR R8              --S := 0
LOAD R9, -5, 1      --E := 0.00001
MOVC 2
MOVA R10            --I:=2
IN Port2
MOV R11             --R11:=Port2:=X
LOAD R12,1,628      --R12:=2*Pi
nextone:
FCMP R11, R12 --сравнить X и 2*Pu, если X меньше, то C = 1
JMPC anoth
       FMOV R13, R12       -- R13:=2*Pi
       MOVC -1
       MOV R14             --R14:=-1
       FMUL R13, R14       --R13 := -2*Pi
       FMOV R14, R12       --R14 := 2* Pi
       FMOV R15, R11--R15 := X
       FDIV R15, R14 --R15 := X / 2 * Pi
       TRUNC R15           --R15 := Целая часть от X / 2 * Pi
       FMUL R13, R14       --R13 := -2*Pi * trunk(X / 2 * Pi) == P1
       FMUL R13, R13--R13 := P1*P1
       MOV R15, R10 --R15 := I
       MOV R14, R15 --R14 := I
       DEC R15             --R15 := I – 1
       FMUL R14, R15       --R14 := I*(I-1)
       FDIV R13, R14 --R13 := P1*P1/I*(I-1)
       MOVC -1
       MOVA R14            --R14 := -1
       FMUL R13, R14       --R13 := -P1*P1/I*(I-1)
       FMUL R5, R13 --R5 == P := -P*P1*P1/I*(I-1)
       JUMP next
anoth: FMOV R13, R12       -- R13:=2*Pi
       FMOV R14, R12       --R14 :=2* Pi
       FMOV R15, R11--R15 := X
       FDIV R15, R14 --R15 := X / 2 * Pi
       TRUNC R15           --R15 := Целая часть от X / 2 * Pi
       FMUL R14, R13       --R14 := 2*Pi * trunk(X / 2 * Pi) == P1
       FABS F14            --R14 := ABS(P1)
       FMOV R13, R11--R13 := X
       FADD R13, R14       --R13 := X + ABS(P1)
next:
FMOV R5, R13        -- P := X + ABS(P1)
FADD R8, R5         -- S := S + P
ADDC R10, 2         -- I := I + 2
FMOV R6, R5         --R6 := P
FABS R6                   --R6 := ABS(P)
FCMP R6, R9         --Сравниваение с експонентой
JMPC end
JUMP nextone
end:
```

Приложение Д. Листинг основной программы

```
.include "8515def.inc"

.CSEG
.org 0x00   ;reset
rjmp beg
.org 0x06     ; t1 overflow
rjmp t1ovf

.equ KB1 = 0x90FE
.equ KB2 = 0x90FC
.equ KB3 = 0x90FB

#define X1 R20
#define X2 R21
#define Y1 R22
#define Y2 R23
#define CT R24
#define SET R25

beg:
LDI CT, 0x06
LDI SET, 0xFF
; === Разрешаем прерывания ===
sei
; ============================
        ldi r16, low(RAMEND)
        out spl, r16
        ldi r16, high(RAMEND)
        out sph, r16

        ldi     r24, 0b11000000
        out     DDRB, r24
        out     MCUCR, r24          ;
; === инициализируем и запускаем таймер 1 ===
        ldi r16, 0b10000000   ; разрешаем только прерывание по
        out TIMSK, r16        ; переполнению таймера 1

        ; указываем источник и предделитель, тем самым запуская
        ldi r16, 0b00000000   ; режимы сравнения не нужны
        out TCCR1A, r16

        ldi r16, 0b00000101   ; указываем источник и предделитель
        out TCCR1B, r16
; ========================================

forever:
rjmp forever

t1ovf:
```

```
//====ПРОВЕРКА НАЖАТИЯ КЛАВИШ====
LDS R16, KB1
//======КНОПКА 1====== (ВЫЗОВ ПОДПРОГРАММЫ SINX)
SBRC R16, 0         //пропустить следующий если нажато
RJMP NextB11    //если нет, прыгаем на след.
       RCALL sin_x
       RJMP End    // конец обработчика кнопки
NextB11:

//======КНОПКА 4====== (ВЫЗОВ РАБОТЫ УСТРОЙСТВА АЛУ (УМНОЖЕНИЕ))
SBRC R16, 1
RJMP NextB12
       LDI R16, 1 //на дешифратор
       LDI R17, 16
       OUT PORTC, R16 //на дешифратор
       OUT PORTB, X1
       OUT PORTB, X2
       OUT PORTB, Y1
       OUT PORTB, Y2
       OUT PORTB, R17
       IN R16, PORTB //результат умножения
       IN R17, PORTB
       RJMP End
NextB12:

//======КНОПКА 7====== (ВЫЗОВ РАБОТЫ УСТРОЙСТВА ДЕЛЕНИЯ)
SBRC R16, 2         //пропустить следующий если нажато
RJMP NextB13        //если нет, прыгаем на конец
       LDI R16, 2 //на дешифратор
       OUT PORTC, R16 //на дешифратор
       OUT PORTB, Y1
       OUT PORTB, X1
       OUT PORTB, SET
       IN R16, PORTB //результат деления
       RJMP End    // конец обработчика кнопки
NextB13:

//======КНОПКА *====== (ВЫЗОВ РАБОТЫ УСТРОЙСТВА КОРНЯ)
SBRC R16, 3         //пропустить следующий если нажато
RJMP NextB14        //если нет, прыгаем на конец
       LDI R16, 3 //на дешифратор
       OUT PORTC, R16 //на дешифратор
       OUT PORTB, X1
       OUT PORTB, SET
       IN R16, PORTB //результат корня
       RJMP End    // конец обработчика кнопки
NextB14:
LDS R16, KB2

//======КНОПКА 2====== (ВЫЗОВ РАБОТЫ УСТРОЙСТВА 2 -> 10)
SBRC R16, 0         //пропустить следующий если нажато
RJMP NextB21    //если нет, прыгаем на след.
```

```
            LDI R16, 4 //на дешифратор
            OUT PORTC, R16 //на дешифратор
            OUT PORTB, X1
            OUT PORTB, CT
            IN R16, PORTB //результат 2->10 числа
            RJMP End    // конец обработчика кнопки
        NextB21:

        //======КНОПКА 5======(ВЫЗОВ РАБОТЫ УСТРОЙСТВА С ПЛАВАЮЩЕЙ ЗАПЯТОЙ)
        SBRC R16, 1         //пропустить следующий если нажато
        RJMP NextB21    //если нет, прыгаем на след.
            LDI R16, 5 //на дешифратор
            OUT PORTC, R16 //на дешифратор
            OUT PORTB, X1
            OUT PORTB, Y1
            OUT PORTB, X2
            OUT PORTB, Y2
            IN R16, PORTB //результат числа (мантисса)
            IN R17, PORTB //результат числа (порядок)
            RJMP End    // конец обработчика кнопки
        NextB21:


        End:
        LDI R16, 0xFF //частота задается таймером
        OUT TCNT1H, R16
        CLR R16
        OUT TCNT1L, R16
reti
```

Приложение Е. Листинг VHDL-проекта устройства для выполнения вычисления с плавающей запятой

```
-- hds interface_start
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

---------------- НОРМАЛИЗАТОР ------------------
ENTITY normalize IS
   PORT(
      SIG_in_A  : IN     std_logic_vector (5 DOWNTO 0);
      EXP_in_A  : IN     std_logic_vector (4 DOWNTO 0);
      SIG_in_B : IN    std_logic_vector (5 DOWNTO 0);
      EXP_in_B : IN    std_logic_vector (4 DOWNTO 0);
      SIG_in_REZ_A : OUT    std_logic_vector (5 DOWNTO 0);
      EXP_in_REZ_A : OUT    std_logic_vector (4 DOWNTO 0);
      SIG_in_REZ_B : OUT    std_logic_vector (5 DOWNTO 0);
      EXP_in_REZ_B : OUT    std_logic_vector (4 DOWNTO 0);
   );

-- Declarations
END normalize ;


-- hds interface_end
ARCHITECTURE normalize OF normalize IS
BEGIN

PROCESS(SIG_in_A, EXP_in_A, SIG_in_B, EXP_in_B)
BEGIN
      IF std_match(EXP_in_A, EXP_in_B) THEN
            SIG_in_REZ_A <= SIG_in_A;
            SIG_in_REZ_B <= SIG_in_B;
            EXP_in_REZ_A <= EXP_in_A;
            EXP_in_REZ_B <= EXP_in_B;
      ELSE
           for i in 1 to n loop

         END loop;
     END IF;
END PROCESS;

END normalize ;

---------------- Сумматор ------------------
ENTITY summator IS
   GENERIC(
      n: integer := 5
   );
```

```
   PORT(
      SIG_in_A  : IN     std_logic_vector (n DOWNTO 0);
      SIG_in_B : IN    std_logic_vector (n DOWNTO 0);
      SIG_out: OUT    std_logic_vector (n DOWNTO 0);
   );

-- Declarations
END summator;

-- hds interface_end
ARCHITECTURE summator OF summator IS
  variable carry : bit;
  variable vsum : std_logic_vector (n DOWNTO 0);
BEGIN

PROCESS(SIG_in_A, SIG_in_B)
BEGIN
    for i in 1 to n loop
        vsum(i) := (SIG_in_A(i) xnor SIG_in_B(i)) xor carry;

        carry := ((SIG_in_A(i) and SIG_in_B(i)) or (carry and (SIG_in_A(i) or SIG_in_B(i)));
    end loop;
    SIG_out <= vsum;
END PROCESS;

END summator;

----------------ВСЁ ВМЕСТЕ---------------
ENTITY FPadd IS
   PORT(
      START : IN     std_logic;
      SIG_in_A  : IN     std_logic_vector (5 DOWNTO 0);
      EXP_in_A  : IN     std_logic_vector (4 DOWNTO 0);
      SIG_in_B : IN    std_logic_vector (5 DOWNTO 0);
      EXP_in_B : IN    std_logic_vector (4 DOWNTO 0);
      SIG_NORM_A : OUT    std_logic_vector (5 DOWNTO 0);
      EXP_NORM_A  : OUT    std_logic_vector (4 DOWNTO 0);
      SIG_NORM_B : OUT    std_logic_vector (5 DOWNTO 0);
      EXP_NORM_B : OUT    std_logic_vector (4 DOWNTO 0);
      clk     : IN     std_logic;
      SIG_REZ : OUT    std_logic_vector (5 DOWNTO 0);
      EXP_REZ : OUT    std_logic_vector (4 DOWNTO 0);
   );

-- Declarations
END FPadd;

-- hds interface_end
ARCHITECTURE FPadd OF FPadd IS
BEGIN
    K1: ENTITY normalize PORT MAP(SIG_in_A, EXP_in_A, SIG_in_B, EXP_in_B, SIG_NORM_A,
EXP_NORM_A, SIG_NORM_B, EXP_NORM_B);
```

```
    K2: ENTITY summator PORT_MAP(SIG_NORM_A, SIG_NORM_B, SIG_REZ);
PROCESS
BEGIN
    IF (START = '1') THEN
          EXP_REZ <= EXP_NORM_B;
    ENDIF;
END PROCESS;

END FPadd;
```

Выводы

В результате выполнения курсового проекта была спроектирована специализированная микроЭВМ, реализующая специальную систему команд, ориентированную на выполнение арифметических операций над целыми числами.

Разработанная ЭВМ позволяет организовать работу в многопрограммном режиме. Наличие достаточного объема памяти и ее эффективная организация являются необходимыми условиями для функционирования микроЭВМ в многопрограммном режиме.

Список используемой литературы

1. Жабин В.И. Ткаченко В.В. "Однокристальные и микропрограммируемые ЭВМ"
2. Жабин В.И., Макаров В.В., Ткаченко В.В., Зайцев А.А. "Архитектура однокристальных ЭВМ"
3. Конспекты лекций "Архитектура компьютеров"