Федеральное государственное бюджетное образовательное учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Курс лекций

дисциплины базовой части математического и естественнонаучного цикла Б.1

«Микропроцессорные системы ч.2»

(7 семестр)

Направление подготовки 09.03.01 Информатика и вычислительная техника

Профили: Автоматизированные системы обработки информации и управления, Вычислительные машины, комплексы, системы и сети

Авторский коллектив:

Доцент кафедры ВМСС, к.т.н. _____ С.Н. Михалин

«01» сентября 2017 г.

МИКРОКОНТРОЛЛЕРЫ

Микроконтроллер (МК) можно определить как самостоятельную микропроцессорную систему (МПС), которая содержит ядро, вспомогательные схемы и устройства ввода-вывода, размещенные в одном корпусе и предназначенная для управления различными электронными устройствами. Как следует из определения, основным назначением микроконтроллеров является программное управление объектами и связь с внешними устройствами (это преобразование интерфейса, функции обработки аналоговых сигналов, мониторинг показателей и т.п.). На рис.1 представлена обобщенная структура МК. Можно видеть, что МК содержат те же узлы, что и микропроцессорные системы.

Рис. 1 – Общая структура микроконтроллера

«Ядро» – базовое устройство внутренней вычислительной системы. Ядро определяет систему команд, шинный интерфейс, архитектуру памяти, т.е. главные отличия вычислительных систем друг от друга. Ядро МК может быть одинаковым, а изготовители – разными.

«Семейство» – группа микросхем, имеющих одно ядро, у которых примерно одинаковый набор программных и периферийных функций.

«Серия» или «линейка» – это фирменный бренд (реклама), например, серия «*Classic*», серия «*tinyAVR*», линейка «*MegaPIC*». Встречаются и обобщенные названия по типу «линейка 16-битных МК общего назначения».

«Модель» – несколько микросхем одного семейства, различающиеся между собой второстепенными цифрами (буквами) в названии, что определяет разный температурный диапазон, тактовую частоту, вариант корпуса, диапазон питающего напряжения и т.п.

Как и микропроцессоры (МП) развитие архитектуры и изменение состава периферии различных МК происходит быстро. Например, только одна компания *Microchip* за 1997 год выпустила свыше 40 видов МК, а клонов распространенного *Intel* 8051 на сегодняшний день существует более 200 модификаций, выпускаемых двумя десятками компаний. Практическая потребность в МК (множество задач, решаемых с помощью МК) побуждает многих производителей выпускать целые серии различных МК. Среди таких компаний можно назвать *Atmel*, *Dallas Semiconductor*, *Intel*, *Analog Devices*, *Infineon Technologies*, *Hitachi Semiconductor*, *Microchip Technology Inc.*, *Mitsubishi Electronics*, *National Semoconductor*, *Philips Semiconductors*, *STMicroelectronics*, *Texas Instruments*, *Toshiba*, *Zilog* и др. Отметим, что и в СССР в 1979 году была разработана однокристальная 16-разрядная ЭВМ К1801ВЕ1, микроархитектура которой называлась «Электроника НЦ». В связи с этими обстоятельствами нерационально изучать конкретный вид МК, но имеет смысл в первую очередь проанализировать общие подходы к их построению и применению, а затем рассмотреть наиболее популярные и перспективные серии.

Все МК можно разделить по назначению на три большие во многом пересекающиеся группы:

- встраиваемые 8-разрядные МК (*embedded*) – дешевые, простые в применении, содержащие минимальное количество компонентов для реализации самых простых задач (требующих малого потребления, невысокой производительности и небольших объемов памяти);
- 16- и 32-разрядные МК – устройства, на кристалле которых интегрированы специфические блоки: такие как монитор отладки, аналоговый ввод-вывод, развитая система прерываний, наличие внешней полноценной шины для подключения внешней памяти, процессорных устройств и т.п. (для управления несколькими устройствами, простейшей обработки НЧ сигналов);
- цифровые процессоры *DSP* – 32-х разрядные и более МК с высокой производительностью и специализацией системы команд под стандартные процессы обработки данных (фильтрация, БПФ, сверточное кодирование, преобразование кодов и т.п.), могут содержать несколько вычислительных модулей и средства поддержки вычислений с плавающей точкой.

В свою очередь МК каждой этой группы можно разделить по принципу построения:

- МК с Гарвардской архитектурой (разделены память программ и данных; наличие регистров общего назначения);
- МК с Принстонской (Фон Неймана) архитектурой (память программ и данных едина, регистры имеют предписанное назначение, например: аккумулятор);
- МК с *RISC* ядром (сокращенная система команд);
- МК с *CISC* ядром (полная система команд).

Разные архитектуры каждого МК, построенные с тем или иным ядром, обладают как преимуществами, так и недостатками. Однако есть МК, в которых разработчики сделали попытку соединить достоинства разных архитектур. Поэтому решение одной и той же задачи, реализуемой на разных МК, должно быть оптимизировано под конкретную архитектуру и систему команд конкретного МК.

С учетом сказанного рассмотрим обобщенные структурные схемы основных узлов применяемых в МК (таймеры, подсистемы прерываний, параллельный и последовательный интерфейсы, организация памяти, принципы построения МК и т.п.). Затем рассмотрим конкретные реализации этих идей в семействах МК *Intel* 8051 (как классический *CISC* МК с гарвардской архитектурой), *Motorola 68HC*05*xx* (*CISC* МК с принстонской архитектурой), *Atmel AVR AT*90*S*xxxx (как классический и простейший *RISC* МК с гарвардской архитектурой).

АППАРАТНЫЕ СРЕДСТВА МИКРОКОНТРОЛЛЕРОВ

Запуск (сброс в начальное состояние) МК осуществляется подачей питания или активным уровнем сигнала на входе *Reset*. После включения питания МК должен начать работу после окончания переходных процессов, связанных с установлением нормального уровня питающего напряжения и нормальной частоты и формы тактирующих импульсов. Для обеспечения задержки между появлением питания и началом работы применяют *RC*-цепочку, включенную к входу *Reset*. Часто такая схема задержки уже встроена в МК.

Тактирование системы осуществляется от внешнего или встроенного генератора – рис.2. Для работы последнего обычно требуется включение внешнего элемента – кварцевого резонатора (*Q*1). Однако если при работе МК не требуется измерений или отсчета интервалов точного времени, то вместо кварца допускается включение *RC*-цепи.

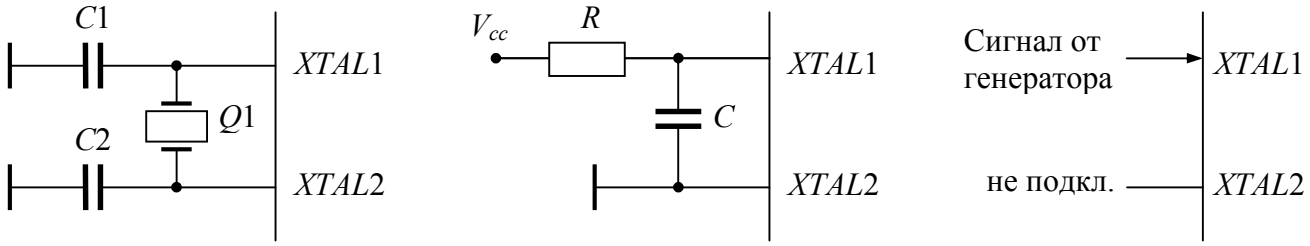

Рис. 2 – Схемы тактирования МК

Командный цикл МК состоит из нескольких тактов синхронизации, которые необходимы для выполнения команд. Некоторые команды могут требовать нескольких командных циклов, что свойственно *CISC* архитектурам.

Арифметико-логическое устройство (АЛУ) представляет собой аппаратный блок, реализующий операции сложения, вычитания (умножения, деления), логических операций (*and*, *or*, *xor*, *not*) и сдвигов влево и вправо. АЛУ не используется при чтении или записи данных или команд, оно служит только для обработки данных. АЛУ обрабатывает два операнда и сохраняет результат в третьем. Но физическая реализация (источники и приемник данных) зависит от конкретной модели МК.

Сторожевой таймер – это аппаратная схема, позволяющая исключать из работы МК или процессора ситуации его зависания, блокировки и т.п. Устройство вызывает сброс системы, если через фиксированный момент времени (обычно от нескольких десятков миллисекунд до нескольких секунд) содержимое определенного регистра не будет обновлено. Проще говоря, запускается аппаратный таймер, считающий к нулю, происходит проверка управляющего слова и, если записи в управляющее слово за время счета таймера не произошло, то генерируется сигнал сброса МК, иначе счетчик таймера перезагружается и начинает счет сначала.

Прерывание – это аппаратная реализация отклика синхронной системы на асинхронное событие. Любой МК, получив запрос на прерывание, может реагировать на него одним из трех способов:

а) игнорировать запрос (маскирование прерывания) и при необходимости реализацией поллинга выполнение прерывания позже – например, после завершения текущей задачи;
б) быстро среагировать на запрос, сообщив внешнему устройству (которое выставило прерывание), что МК занят и процедура обработки поступившего прерывания будет выполнена позже – например, после завершения текущей передачи данных;
в) быстро среагировать и обслужить запрос полностью, а затем вернуться к прерванной задаче.

Обработчик прерываний обычно выполняет следующую последовательность действий: сохранение контекста регистров, посылка управляющей комбинации подсистеме прерываний и внешнему устройству, обработка данных, восстановление контекста регистров, возврат к прерванной программе. В МК передача управления обработчику и возврат управления прерванной программе реализуются аппаратно, но в любом случае для этих целей применяется стек, глубина которого ограничивает количество вложенных прерываний.

Таймеры – аппаратный счетчик со схемой предварительного деления частоты, с регистрами управления, статуса, начальной загрузки и защелки (позволяющей читать значение счетчика «на лету»), а также со схемами коммутации входного и выходного сигналов. Обычно один МК содержит несколько таймеров (необязательно одинаковых). На рис.3 представлена обобщенная схема таймера.

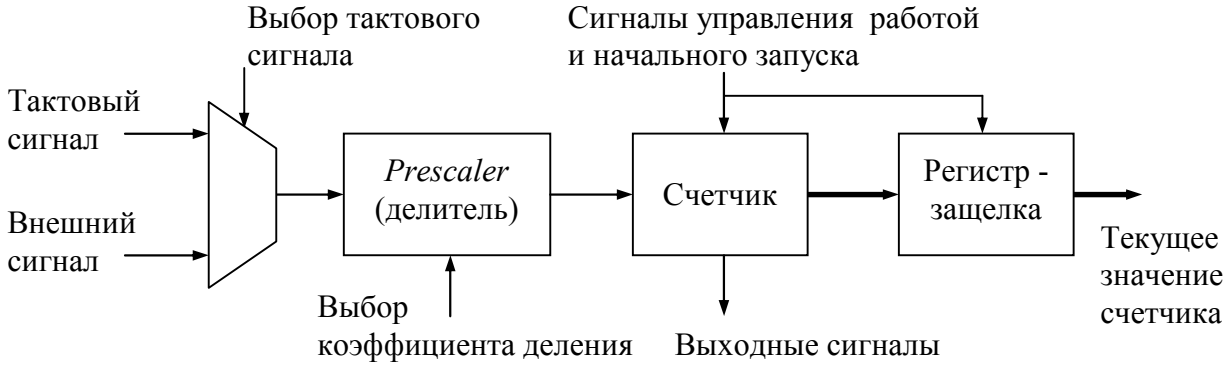

Рис. 3 – Обобщенная структурная схема таймера

Часто в МК таймеры используются для вывода сигналов с широтно-импульсной модуляцией (ШИМ или *Pulse Width Modulated*). Для реализации режима ШИМ к структуре рис.3 необходимо добавить два регистра и два компаратора – рис.4.

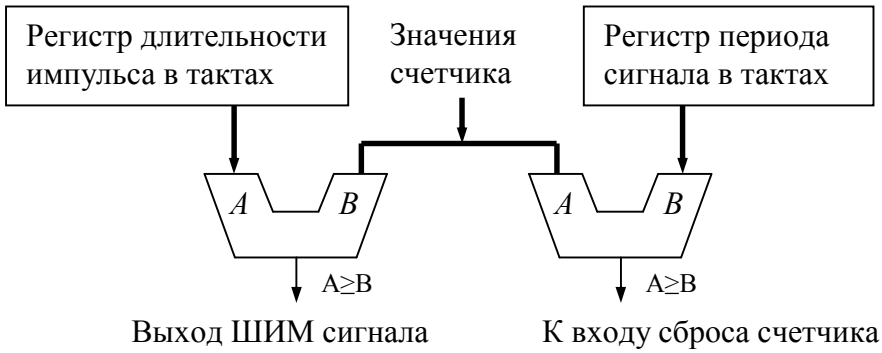

Рис. 4 – Дополнение структурной схемы таймера

Параллельный ввод-вывод данных является основным интерфейсом МК – порт. Во многих МК для сокращения внешних выводов микросхемы порты ввода/вывода сочетаются с другими функциями, а выбор назначения такого контакта осуществляется программным способом. Типовая схема организации линии (одного бита) порта приведена на рис.5.

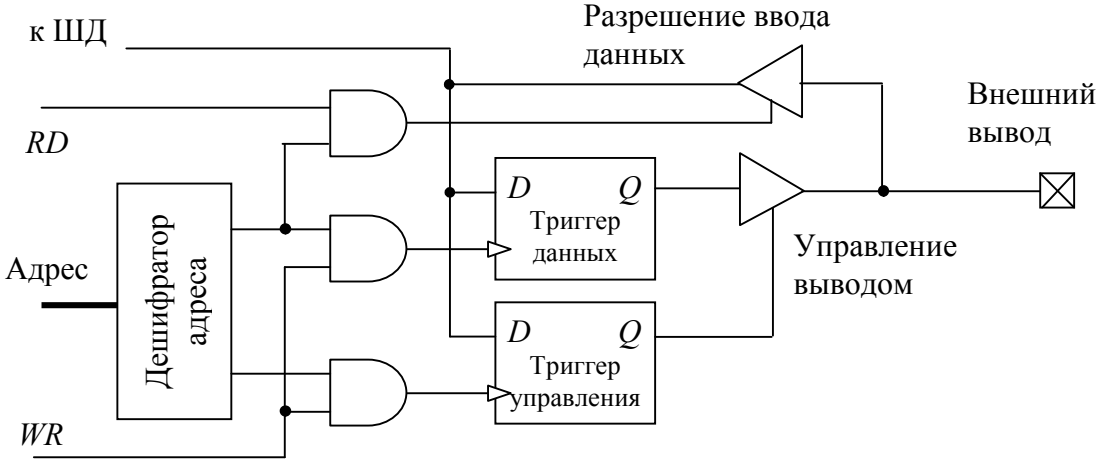

Рис. 5 – Типовая схема линии порта ввода-вывода

Последовательный ввод-вывод – наиболее распространенный вид связи для реализации обмена между несколькими МК или между МК и различными устройствами. Существует множество протоколов передачи данных последовательным кодом, которые имеют различное назначение (обмен с другими контроллерами, управление чем-либо, обмен с датчиками): *Microwire*, *SPI*, *CAN*, *I2C* и т.п.

Асинхронный обмен *SCI* (*Serial Communication Interface*) предполагает пакетную передачу данных, т.е. аналогичен интерфейсу *RS*-232 (*COM* порт *x*86). Часто для увеличения помехоустойчивости передачи применяют кодирование как сообщения сигнала так и логических уровней (например: лог. 1 это перепад с высокого напряжения на низкое, а ноль – наоборот или логические уровни кодируются серией перепадов). Примеры: локальная вычислительная сеть (протокол *TCP*/*IP*) – манчестерский код, *USB* – дифференциальная передача с кодированием лог. «0» как перепад напряжения и самосинхронизацией.

Синхронный обмен предполагает непрерывную передачу данных, характерную потоковому или конвейерному методу обработки данных.

В частности, для подключения *Flash*-памяти, медленных АЦП применяют: *SPI*, а для обмена между несколькими МК: *I2C* и *CAN*, которые подразумевают обмен по схеме *master-slave* и являются комбинацией асинхронного и синхронного способов передачи данных.

Краткие сведения о *SPI* и *Microwire*

Последовательный периферийный интерфейс *SPI* (*Serial Peripheral Interface*) обеспечивает высокоскоростной синхронный дуплексный обмен данными между МК и периферийными устройствами или между несколькими МК (по дальности: в рамках платы, блока и т.п.). В общем случае, один из многих МК или устройств должен быть ведущим (*Master*), остальные – ведомыми (*Slave*). Для обеспечения обмена данными между устройствами используются четыре линии: *SCK* (*Serial clock*), *MOSI* (*Master Output data*, *Slave Input*), *MISO* (*Master Input data, Slave Output*), *SS* (*Slave Select*) используется ведущим для выбора одного из ведомых устройств (при обмене типа "точка-многоточка"). Тактовые сигналы по линии *SCK* всегда генерирует ведущий МК. Данные от ведущего передаются по линии *MOSI*, прием данных осуществляется по линии *MISO*. Если ведомым устройством является МК, то его вывод, соответствующий линии *SS*, должен быть настроен как вход. Протокол *SPI* обеспечивает:

- скорость передачи достигает 3 Мбит/сек.;
- разрядность данных в пакете составляет 8 бит;
- передатчик может приостанавливать передачу данных;
- данные могут передаваться блоками (страницами), которые объединяют множество байтов.

Формат пакета *SPI* включает: старт-бит, 8-разрядную команду, необязательный 16-разрядный адрес и 8-разрядные данные. В протоколе применяется симметричное тактирование: тактовые сигналы представляют собой меандр. Выходные данные выдаются на линию не позже чем за 30 нс до переднего фронта тактового сигнала, а считывание происходит за 30 нс до заднего фронта. На рис.6 показана временная диаграмма обмена данными по интерфейсу *SPI* для случая передачи байта (восемь периодов сигнала *SCK*), *CPOL*=0 – управляющий бит определяет полярность сигналов *SCK*, *CPHA*=0 – фаза синхронизации: по переднему фронту в цикле синхронизации будет выполняться выборка данных, а по заднему фронту – установка данных, в цикле обмена *SS*=0.

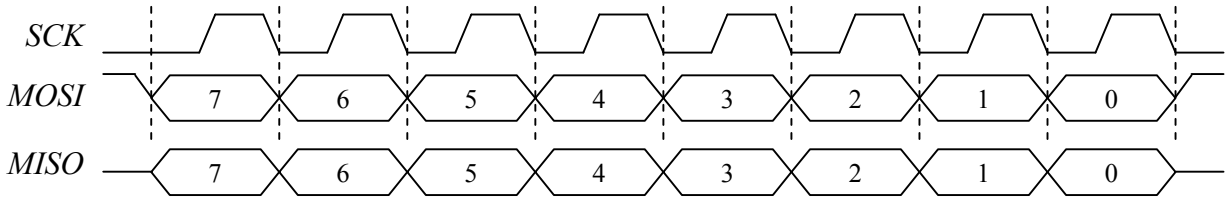

Рис. 6 – Временная диаграмма обмена данными по интерфейсу *SPI*

На рис.7 представлены схемы соединения устройств (*slave*) с МК (*master*) по шине *SPI* (вариант кольцевого соединения поддерживается не всеми устройствами).

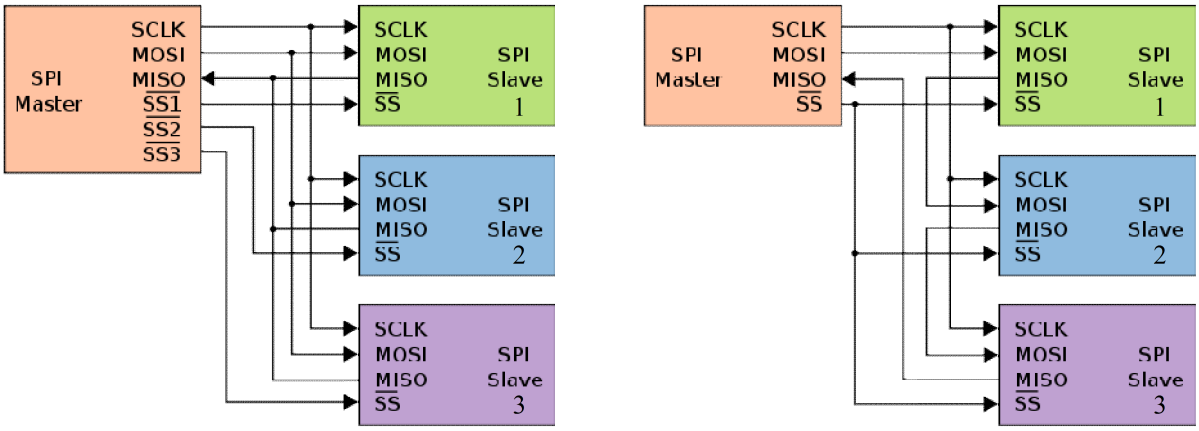

Рис.7 – Схемы соединения по шине *SPI*. Слева – радиальная; справа – кольцевая.

Протокол *Microwire* почти не отличается от *SPI* и обеспечивает передачу данных со скоростью до 1 Мбит/сек. Структура пакета состоит из старт-бита, 8-разрядной команды, необязательного 16-разрядного адреса и 16-разрядных данных. Передаваемые данные должны выдаваться на линию за 100 нс до поступления переднего фронта тактового сигнала. Чтение данных должно происходить за 100 нс до заднего фронта тактового сигнала.

Основными недостатками интерфейсов является:

- нет подтверждения приема данных со стороны ведомого устройства;
- нет определенного стандартом механизма обнаружения ошибок;
- ведомое устройство не может управлять потоком данных.

Краткие сведения о *I2C*

Шина *I2C* (*Inter-Integrated Circuit*) была разработана компанией *Philips* в конце 70-х и позволяет разделять ресурсы между несколькими устройствами (до 112 шт., 16 адресов зарезервированы). Это двунаправленная (полудуплексная) синхронная шина с последовательной передачей данных (стандартизована в 1992 г.).

Шина содержит две сигнальные линии: *SCL*-синхросигналы (*serial clock*) и *SDA*-данные (*serial data*), выполненные по схеме с открытым коллектором. Поэтому обе линии подтягивают к высокому уровню (через внешние резисторы к цепи питания) и такое состоянии линий является пассивным (*idle* состояние). Обмен информацией осуществляется байтами. Каждый байт передается в течение 9 тактов – периодов сигнала синхронизации на линии *SCL*.

Чтобы инициировать передачу данных ведущее устройство устанавливает низкий уровень сначала на линии данных *SDA*, а затем на линии *SCL* (стартовое состояние – рис.8). В процессе пересылки данных такое состояние шины является нерабочим, т.к. прием данных осуществляется при высоком уровне синхроимпульсов на линии *SCL*. Данные передаются синхронно начиная со старшего бита. После передачи данных ведущее устройство переводит линию в плавающее состояние, ожидая подтверждения приема данных от ведомого устройства (рис.8). Таким подтверждением (*ACK*) является установка низкого уровня на линии *SDA*. Затем пересылается следующий байт или шина переводится в пассивное состояние (как показано на рис.8). Высокий уровень на линии *SDA* в девятом периоде свидетельствует о возникшей ошибке в приемнике (отсутствие подтверждения: *NAK*). Следует отметить, что передатчиком может быть как ведущее, так и ведомое устройства и, следовательно, получатель информации всегда должен формировать подтверждение *ACK*.

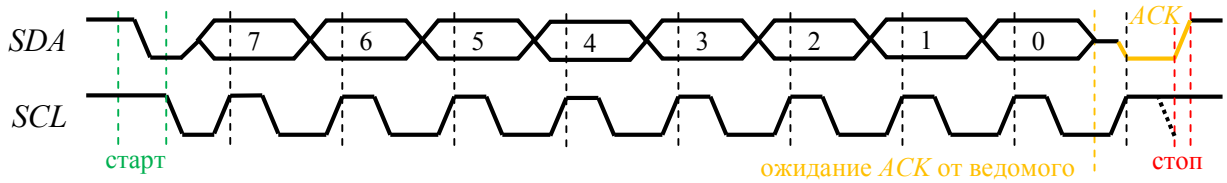

Рис. 8 – Временная диаграмма обмена данными по шине *I2C*

Максимальная скорость обмена по этому протоколу составляет 400 Кбит/сек (стандартно – до 100 Кбит/сек) при длине линии связи до 2 м (суммарная емкость не более 400 пФ), существуют «модификации» увеличивающие скорость передачи до 3.4 Мбит/сек.

Пакет состоит из адреса приемника (7 бит – 128 устройств), бита *R*/*W* указывающего направление передачи данных, бита подтверждения, собственно передаваемых или принимаемых данных (байт) и еще одного бита подтверждения. На рис. 9 показан цикл обмена.

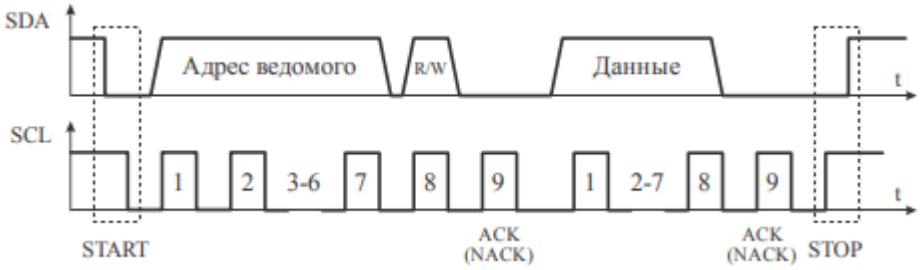

Рис. 9 – Временная диаграмма цикла обмена на шине *I2C*

Все устройства шины делятся на два типа: ведущее (*Master*) и ведомое (*Slave*). Тактовый сигнал *SCL* генерирует только ведущее устройство. Оно может самостоятельно выходить на шину и обращаться по адресу к любому ведомому устройству. При обнаружении собственного адреса и, распознав его, ведомые устройства выполняют предписываемую операцию. Возможен режим работы шины с несколькими ведущими устройствами. На рис. 10 представлена схема соединения по шине *I2C*, резисторы *R* подтягивают потенциалы линий к напряжению питания.

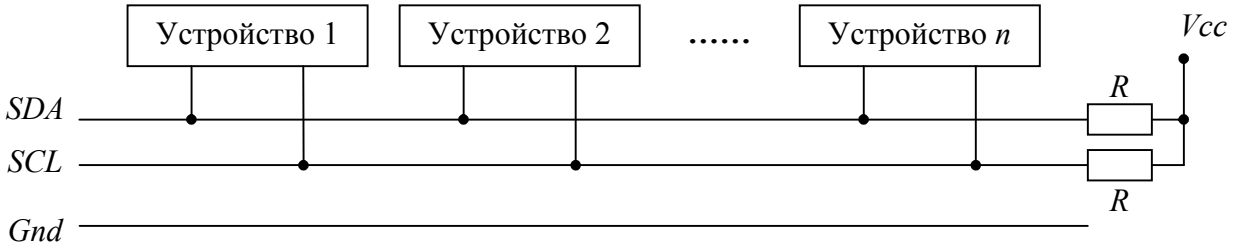

Рис. 10 – Схема соединения по шине *I2C*

Краткие сведения о *CAN*

Последовательный интерфейс *CAN* (*Controller Area Network*) был разработан компанией *Bosch* в 1980-х для связи компьютерных систем (главным образом в автомобилестроении) и удовлетворяет следующим требованиям:

- полудуплексный обмен до 1 Мбит/сек (на расстоянии до 30-40 м);
- нечувствительность к электромагнитным помехам;
- простота применения и небольшое количество разъемных контактов.

Физически протокол *CAN* основан на драйверах *RS*-485, обеспечивающих дифференциальную передачу сигнала, и реализуется с использованием операции «монтажное И» – рис.11.

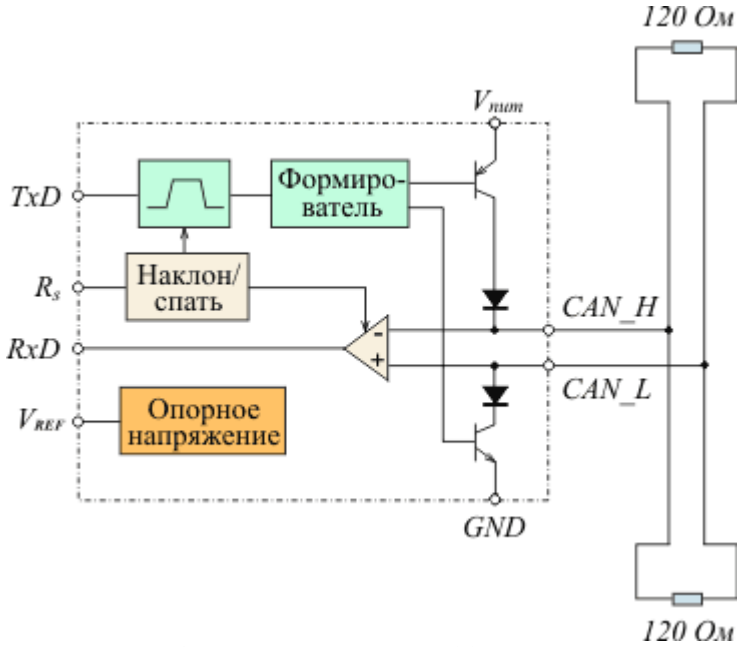

Рис. 11 – Структурная схема *CAN* трансивера

Если базироваться на семиуровневой модели сети *OSI*, то *CAN* описывает передачу данных между узлами на двух нижних уровнях – физическом и канальном. Битовый поток кодируется по методу *NRZ* (*Non Return to Zero* – кодирование без возврата к нулю – простое потенциальное кодирование – рис.12). Для повышения устойчивости синхронизации используется вставка нулевого бита в случае следования подряд пяти единиц или вставка единицы в случае следования подряд пяти нулей (этот процесс называется *Bit Stuffing*).

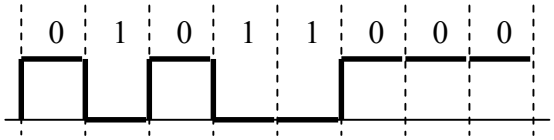

Рис. 12 – Пример *NRZ* кодирования

Кадр (фрейм) протокола включает:

- старт-бит (*SOF - start of frame*) – низкий уровень сигнала;
- идентификатор (11 бит - версия *A* или 29 бит - версия *B*);
- бит, определяющий направление передачи;
- два нулевых бита (резерв);
- длина передаваемого сообщения в байтах (4 бита);
- данные (0-8 байт);
- 15-битный *CRC* кода (формируется циклическим кодом и служит для обнаружения ошибок, возникающих при передаче кадра, вероятность необнаруженной ошибки: 4.7×10^{-11});
- один единичный бит (разделитель);
- 2-битное поле подтверждения готовности: бит подтверждения (*ACK*) и единичный бит;
- семь единичных бит окончания кадра (*EOF - end of frame*).

Отметим, что версия *B*, часто именуемая *FullCAN*, постепенно вытесняет версию *A*, которую называют также *BasicCAN*. Шина стандартизована *ISO* (*ISO* 11898) и *SAE* (*Society of Automotive Engineers*).

Между кадрами имеется промежуток *IFS* (*Inter Frame Spacing*), состоящий из не менее трех битов.

Рис. 13 – Кадр протокола *CAN*

Краткие сведения о 1-*wire*

Фирмой *Dallas Semiconductor* разработан последовательный асинхронный полудуплексный интерфейс, позволяющий обеспечивать низкоскоростной обмен данными по одной линии (вторая линия – общий провод). Сеть устройств, соединенных этим интерфейсом, известна под названием *MicroLAN*. К однопроводной шине могут быть подключены несколько устройств, но только одно из них является ведущим, а все остальные ведомыми. Допускается питание устройств по линии данных.

Все устройства шины самотактируемые, т.е. логика работы основана на измерении и формировании относительных временных интервалов (импульсов различной длительности). Выходной каскад ведущего строится на схеме с открытым стоком. Синхронизация обмена на шине происходит по падающему фронту (переключение с высокого на низкий уровень напряжения), поскольку в схемах с открытым стоком именно падающий фронт менее зависит от емкости нагрузки.

Активная часть временного интервала шины составляет 60 мкс. Для надежности приема данных, чтение осуществляется примерно в середине интервала (фактически выборка может осуществляться спустя 15-60 мкс после синхронизирующего фронта). После окончания активной части временного интервала требуется освобождение линии не менее чем на 1 мкс (в линии должен быть установлен высокий уровень). Логические уровни передаваемых данных кодируются разной длительностью импульса низкого уровня в линии – рис. 14:

- логическая "1" – длительность импульса <15 мкс;
- логический "0" – длительность импульса ≥60 мкс и <120 мкс.

Рис. 14 – Состояние линии при передаче логической "1" слева и "0" справа

Цикл обмена данными на шине начинается с формирования ведущим устройством состояния сброса. Оно заключается в передаче сигнала низкого уровня длительностью 480 мкс (8 активных периодов шины) и ожидания подтверждения от ведомого устройства в течение еще 480 мкс – рис.15 (пунктирной линией обозначен уровень напряжения, обеспечиваемый подтягивающим резистором). В ответ на импульс сброса каждое устройство, подключенное к линии, производит сброс своих внутренних цепей и через 15-60 мкс (после перевода ведомым линии в высокое состояние) выдает низкий уровень сигнала подтверждения в течение 60-240 мкс. Обнаружив этот импульс, ведущий узел передает адрес нужного ему устройства. Все устройства в 1-*wire* сети обладают идентификатором (адресом), который записан в ПЗУ микросхемы *MicroLAN* (это уникальный ее номер, производитель гарантирует невозможность выпуска двух микросхем с одним номером). Все устройства, адрес которых не совпал с

переданным, логически отключаются от шины до следующего импульса сброса. Выбранному устройству передается код операции обмена данными, который определяет направление передачи. По окончании операции ведущий узел генерирует новый импульс сброса и начинается новый цикл обмена.

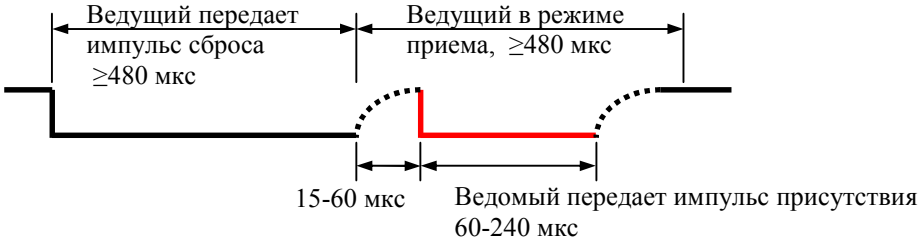

Рис. 15 – Инициализация на шине 1-*wire*

Передача данных в ведомое устройство начинается с установки ведущим устройством на линии низкого уровня длительностью не менее 15 мкс, затем устанавливается уровень передаваемого бита.

Приём данных из ведомого устройства также начинается с установки ведущим устройством в течение 15 мкс на линии низкого уровня, а затем ведущее устройство устанавливает на линии высокий уровень и не более чем через 15 мкс ведомое устройство должно выдать уровень нуля или единицы.

1. Общие сведения о МК *Intel* 8051

В 1980 году *Intel* выпустила МК 8051. Исторически это не первый МК компании, однако, удачный набор периферийных устройств, возможность выбора внешней или внутренней программной памяти и приемлемая цена обеспечили (в свое время) этому МК успех. С точки зрения технологии 8051 являлся для своего времени сложным изделием – кристалл содержит 128 тыс. транзисторов, что в 4 раза превышало количество транзисторов в 16-разрядном МП i8086. На сегодняшний день существует более 10 различных версий классического МК 8051 (87*C*51 – «7» означает наличие *EPROM*), которые совместимы между собой, но отличаются в основном внутренней архитектурой, позволяющей увеличить производительность.

Основными производителями клонов этого семейства МК являются: *Atmel*, *Philips*, *Siemens*, *Dallas*, *AMD*, *Winbond* и другие. В СССР производство его аналогов осуществлялось в Киеве, Воронеже, Минске и Новосибирске (1816ВЕ31/51, 1830ВЕ31/51, 1834ВЕ31, 1850ВЕ31).

МК 8051 имеет гарвардскую архитектуру с возможностью обращения к внешней памяти. На рис.6 представлена упрощенная структура МК, в его состав входят:

- внутренний генератор тактовой частоты *G*;
- арифметико-логическое устройство *ALU*;
- внутренняя память программ *ROM*;
- внутренняя память данных *RAM*;
- четыре восьмиразрядных порта ввода/вывода *P*0 – *P*3;
- интерфейсные устройства *IU* (реализуют специальные функции).

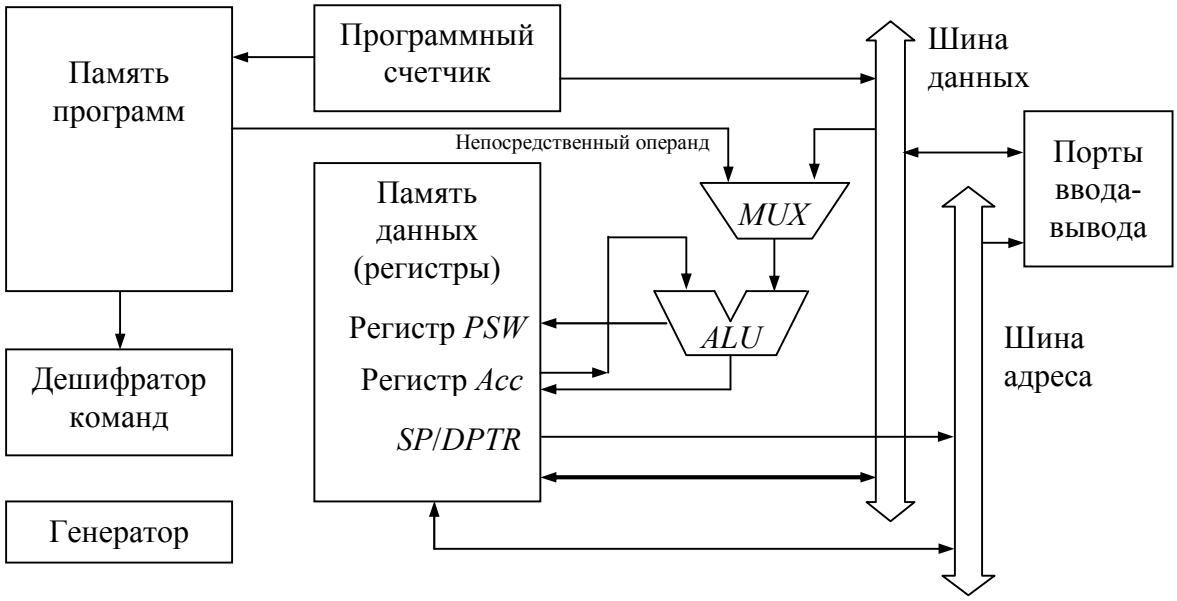

Рис. 1.1 – Упрощенная структурная схема МК 8051

Основные характеристики *CISC* МК 8051:

- восьмиразрядное ядро, оптимизированное для реализации функций управления;
- адресное пространство памяти программ – 64 Кбайт;
- адресное пространство памяти данных – 64 Кбайт;
- внутренняя память программ – 4 Кбайт;
- внутренняя память данных – 128 байт;
- дополнительные возможности по выполнению булевых операций (побитовые операции);
- 32 двунаправленные и индивидуально адресуемые линии ввода-вывода (4 порта);
- два 16-разрядных многофункциональных таймера/счетчика;
- полнодуплексный асинхронный приемопередатчик;
- векторная система прерываний с двумя уровнями приоритета.

Особенностью семейства этих МК является организация памяти – рис.1.2 (в адресном пространстве 080h-0FFh показаны только основные специальные функции).

<table>
<tr><th>Адрес</th><th colspan="2">Прямая адресация</th><th>Косвенная адресация</th></tr>
<tr><td>0FFh</td><td colspan="2"></td><td rowspan="32">Доступная память
(в некоторых моделях может отсутствовать в принципе)</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>0F0h</td><td colspan="2">Регистр «B»</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>0E0h</td><td colspan="2">Аккумулятор (Acc)</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>0D0h</td><td colspan="2">PSW – слово состояния</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>0B8h</td><td colspan="2">IP – регистр приоритетов</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>0B0h</td><td colspan="2">Порт 3</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>0A8h</td><td colspan="2">IE - регистр разрешения IRQ</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>0A0h</td><td colspan="2">Порт 2</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>099h</td><td colspan="2">SBUF – буферный регистр приемопередатчика UART</td></tr>
<tr><td>098h</td><td colspan="2">SCON – регистр управления приемопередатчика UART</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>090h</td><td colspan="2">Порт 1</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>08Dh</td><td>TH1</td><td>Старший байт таймера/счётчика 1</td></tr>
<tr><td>08Ch</td><td>TH0</td><td>Старший байт таймера/счётчика 0</td></tr>
<tr><td>08Bh</td><td>TL1</td><td>Младший байт таймера/счётчика 1</td></tr>
<tr><td>08Ah</td><td>TL0</td><td>Младший байт таймера/счётчика 0</td></tr>
<tr><td>089h</td><td colspan="2">TMOD – регистр режимов таймеров</td></tr>
<tr><td>088h</td><td colspan="2">TCON – регистр управления таймерами</td></tr>
<tr><td>087h</td><td colspan="2">PCON – регистр «мощности»</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>083h</td><td>DPH</td><td rowspan="2">DPTR – регистр косвенной адресации</td></tr>
<tr><td>082h</td><td>DPL</td></tr>
<tr><td>081h</td><td colspan="2">SP – указатель стека</td></tr>
<tr><td>080h</td><td colspan="2">Порт 0</td></tr>
<tr><td>07Fh
030h</td><td colspan="3">ОЗУ (80 байт) – доступная память (обычно здесь размещают стек и переменные)</td></tr>
<tr><td>02Fh
020h</td><td colspan="3">Побитно доступная область (128 бит)</td></tr>
<tr><td>01Fh
000h</td><td colspan="3">4 банка по 8 байт каждый (регистровый файл)</td></tr>
</table>

Рис. 1.2 – Карта памяти МК 8051 (256 первых байт)

Для доступа к первым 256 байтам применяется прямая адресация при этом ячейки памяти можно интерпретировать как отдельные регистры. Для доступа к памяти за пределами 256 байт необходимо применять косвенную адресацию: посредством регистра *DPTR* (*DPH*+*DPL*). При этом служебные регистры с адресами 080h-0FFh не могут адресоваться с помощью косвенной адресации. Эти адреса занимают регистры устройств (таймеры, *UART*, АЦП и т.п.), которые называются регистрами специальных функций (*SFR*). При этом для регистров, адреса которых кратны восьми, возможен побитный доступ (это порты, аккумуляторы, регистры управления).

Регистры-аккумуляторы *Acc* и *B* используются для хранения промежуточных результатов арифметических операций. Регистр *B* используется при умножении и делении.

Указатель стека *SP* может адресовать только 256 байт памяти. После включения МК содержимое *SP* равно 07h. При этом загрузка в стек (*push*) производит инкремент регистра *SP*, тогда как в большинстве других МК указатель стека при этом декрементируется.

Регистр *DPTR* разрядностью 16 бит состоит из двух 8-разрядных регистров *DPH* и *DPL* и может адресовать до 64 Кбайт памяти при косвенной (индексной) адресации.

Слово состояния программы *PSW* содержит флаги результата арифметических операций, флаги управления ходом вычислительного процесса. На рис.1.3 представлен формат регистра состояния и приведено назначение его битов (обратите внимание – нет флага нуля).

CY	*AC*	*F*0	*RS*1	*RS*0	*OV*	*F*1	*P*

CY – флаг переноса, устанавливается когда возникает перенос в старший разряд или заем из него;

AC – флаг полупереноса, устанавливается когда при сложении или вычитании результат операции над младшим полубайтом (тетрадой) влияет на результат операции над старшим полубайтом;

*RS*1, *RS*0 – биты, выбирающие текущий 8-байтовый банк регистров (внутри банка регистрам присвоены имена: *R*0-*R*7), используемых для реализации однобайтных команд, которые содержат меньше байтов и выполняются быстрее (при включении МК или сбросе – устанавливается активным 0-й банк);

*RS*1	*RS*0	банк	адрес
0	0	0	00h - 07h
0	1	1	08h - 0Fh
1	0	2	10h - 17h
1	1	3	18h - 1Fh

OV – флаг арифметического переполнения;

P – флаг четности, устанавливается если результат содержит четное количество единиц;

*F*0, *F*1 – резерв.

Рис. 1.3 – Состав регистра состояния (*PSW*) и назначение его битов

Двунаправленный порт *P*0 может служить для передачи информации по шине данных, связывающей МК с внешней памятью или другими устройствами МПС. Через этот же порт передаётся младший байт 16-разрядного адреса (при работе с внешней памятью). Разделение данных от адреса производится с помощью сигнала *ALE*. Старший байт адреса выводится через двунаправленный порт *P*2.

Двунаправленный порт *P*1 применяется как обычный порт ввода-вывода данных (в более поздних модификациях также может быть многофункциональным).

Двунаправленный порт *P*3 многофункциональный. Его линии используются для ввода-вывода следующих управляющих сигналов:

- бит 0 - последовательный ввод *UART* (приёмник *RxD*);
- бит 1 - последовательный вывод *UART* (передатчик *TxD*);
- бит 2 - вход внешнего прерывания от источника 0 (*INT*0);
- бит 3 - вход внешнего прерывания от источника 1 (*INT*1);
- бит 4 - вход таймера/счётчика0 (*T*0);
- бит 5 - вход таймера/счётчика1 (*T*1);
- бит 6 - сигнал записи данных *WR* во внешнюю память данных;
- бит 7 - сигнал чтения данных *RD* из внешней памяти данных.

1.1 ВИДЫ АДРЕСАЦИИ

Непосредственная адресация не требует обращения к регистрам или памяти данных, т.к. операнд содержится непосредственно в команде (т.е. поступает из памяти программ):
add A,#77; добавить к аккумулятору число (константу) 77.

Адресация к регистровым банкам обеспечивает доступ к одному из 8 байтов, размещенных в текущем банке. Регистры банка называются *R*0-*R*7 (рис.1.2), выбор банка осуществляется битами *RS* слова состояния *PSW* (рис.1.3).

Прямая адресация отличается от регистровой тем, что можно адресовать 256 ячеек. При этом на ячейки памяти с адресами 080h-0FFh отображаются регистры МК (регистры *Special Function Register*). Например: команда mov A,88h загружает в аккумулятор содержимое регистра управления таймером (*TCON*), а не содержимое ячейки памяти с таким адресом.

Косвенно-регистровая адресация осуществляется с помощью регистров *R*0 или *R*1 текущего банка, т.е. значение одного из этих регистров интерпретируется как 8-разрядный адрес для обращения к первым 256 ячейкам памяти. При этом такая адресация не позволяет обращаться к регистрам по адресам 080h-0FFh (соответствующее адресное пространство заполнено ячейками ОЗУ) – говорят: нет отображения регистров. Пример: orl A,@R0 – логическое сложение аккумулятора с байтом по адресу, расположенному в регистре *R*0 (т.е. если *R*0=88h, то обращение будет к ячейке ОЗУ, а не к регистру *TCON* как при прямой адресации).

Косвенно-регистровая адресация со смещением образуется при помощи 16-разрядного индексного регистра *DPTR*. Этот вид адресации удобен для доступа к структурированным переменным образующих массив (отображения регистров нет): mov A,@(DPTR+const).

Для доступа к таблицам (константам) хранящимся в памяти программ можно использовать команды: movc A,@A+DPTR или movc A,@A+PC, которые загружают в аккумулятор значение из ячейки с адресом образуемым содержимым аккумулятора и регистра *DPTR* или *PC* соответственно. Последняя адресует в памяти программ байт, расположенный со смещением относительно адреса текущей команды. Значение смещения содержится в аккумуляторе и вычисляется заранее.

Концептуально, компания *Intel* реализовала три способа адресации, которые позволяют учесть приоритет доступа к определенным массивам данных. В небольшой программе все переменные можно разделить на две группы: те к которым обращение происходит часто и остальные. Поэтому для часто требуемых данных были определены однобайтные команды. Это группа команд работающих в пределах текущего банка регистров (банк это 8 байт). Имеются также двух байтовые команды, обеспечивающие доступ к памяти за пределами текущего банка, но в пределах 256 байт. И существуют трех байтовые команды (всего их 17), обеспечивающие доступ к памяти за пределами 256 байт.

Большинство команд (64 из 111) выполняются за 12 тактов (1 цикл), 45 команд – за 24 такта (2 цикла) и 2 команды – за 48 тактов (4 цикла).

Поэтому при написании программ есть возможность оптимизировать размещение переменных, уменьшая при этом время выполнения программы.

1.2 ПРЕРЫВАНИЯ

Организация прерываний у этого МК достаточно специфична. Рассмотрим карту памяти программ – рис.1.4.

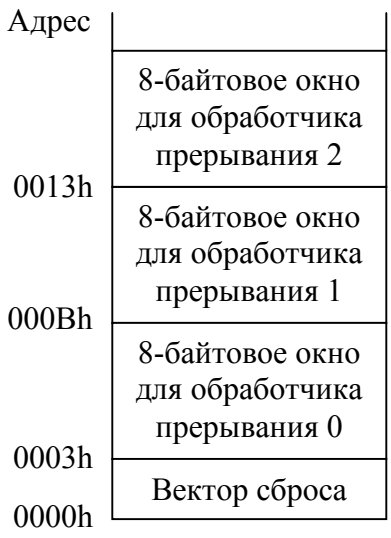

Рис. 1.4 – Карта нижней области памяти программ

По адресу 0 расположен адрес точки программы, с которого процессор начинает выполнение команд при поступлении сигнала сброса или при переполнении сторожевого таймера. При поступлении прерывания с номером 0 ядро МК передает управление на ячейку с адресом 0003h, а при поступлении прерывания 1 – на ячейку с адресом 000Bh и т.д. Таким образом, в МК 8051 обработчик может размещаться непосредственно в «таблице прерываний». Конечно восемь байт это немного, но достаточно, например, для размещения команды передачи управления, либо вызова функции или сброса прерывания, инкрементирования метки реального времени и т.п.

Следует отметить, что при поступлении прерывания МК не сохраняет в стеке регистр флагов. Это связано с тем, что регистр флагов не содержит флага нуля, а код обработчика из арифметических операций обычно содержит только команды инкремента/декремента или установки/сброса битов, которые не влияют на флаги.

В МК могут быть обработаны сигналы от пяти источников прерываний (первые 43 байта памяти программы заняты под «таблицу векторов прерываний»):

- два по переполнению встроенных таймеров/счетчиков;
- два от внешних источников прерывания;
- один от последовательного порта (*UART*).

Прерывания от каждого из указанных источников могут быть независимо друг от друга разрешены или запрещены (замаскированы), причем каждому источнику может быть присвоен приоритет (высокий или низкий). Источник с более высоким приоритетом может прервать программу обслуживания прерывания источника с более низким приоритетом.

Управление системой прерывания обеспечивается с помощью регистров *IE* (адрес 0A8h) и *IP* (адрес 0B8h), которые доступны программно. Назначение бит регистров приведены в нижеследующей таблице.

№ бита	7	6	5	4	3	2	1	0
Регистр *IE*	*EA*	-	-	*ES*	*ET*1	*EX*1	*ET*0	*EX*0
Регистр *IP*	-	-	-	*PS*	*PT*1	*PX*1	*PT*0	*PX*0

*EX*0, *EX*1 – биты разрешения (маски) внешних прерываний *INT*0 и *INT*1;
*ET*0, *ET*1 – биты разрешения (маски) прерываний от таймеров;
ES – бит разрешения (маски) *UART* (маскируемый сигнал является дизъюнкцией сигналов прерывания от приемника и передатчика);
EA – бит разрешения (маски) всех прерываний независимо от битов 0-4 (=0 – запрет);
PS, *PT*1, *PT*0, *PX*1, *PX*0 – биты управления приоритетами соответствующих прерываний (0 – низкий; 1 – высокий приоритеты).

Обработка прерываний от внешних источников осуществляется при поступлении сигналов на входы *INT*1 (*INT*0), причем формирование флага прерывания *IE*1 (*IE*0) в регистре *TCON* (088h) производится либо по уровню внешнего сигнала, либо по спадающему фронту внешнего сигнала. Соответствующий выбор осуществляется с помощью разрядов *IT*1 (*IT*0) регистра *TCON* (1 – обеспечивается прерывание по срезу сигнала, поступающего на вход порта *P*3, иначе – по уровню сигнала). Сброс этих флагов выполняется аппаратно только в том случае,

если прерывание вызвано срезом внешнего сигнала. При потенциальном способе фиксации сигнала прерывания сброс флагов должна осуществлять процедура обслуживания прерывания, воздействуя на источник прерывания с целью снятия им запроса. Разрешение обработки прерывания от внешних источников осуществляется установкой в единицу разрядов *EX*1 (*EX*0) регистра *IE*. Если прерывания разрешены, то обеспечивается переход к программе обслуживания прерывания, адрес которой определяется в соответствии с жестко зафиксированными векторами прерываний – рис.1.5.

Обработка прерываний по переполнению таймера/счетчика производится в том случае, когда разряды *TF*1 (*TF*0) регистра *TCON* (088h) установлены в единицу и прерывания разрешены (разряды *ET*1 (*ET*0) регистра *IE* установлены в единицу). Значения разрядов *TF*1 (*TF*0) автоматически сбрасываются в нуль при входе в подпрограмму обслуживания прерывания.

Формирование флагов *RI* и *TI* осуществляется по готовности приемника или передатчика последовательного порта (*UART*). Разрешение прерывания от этого источника обеспечивается установкой в единицу разряда *ES* регистра *IE*. Сброс флага осуществляется программно.

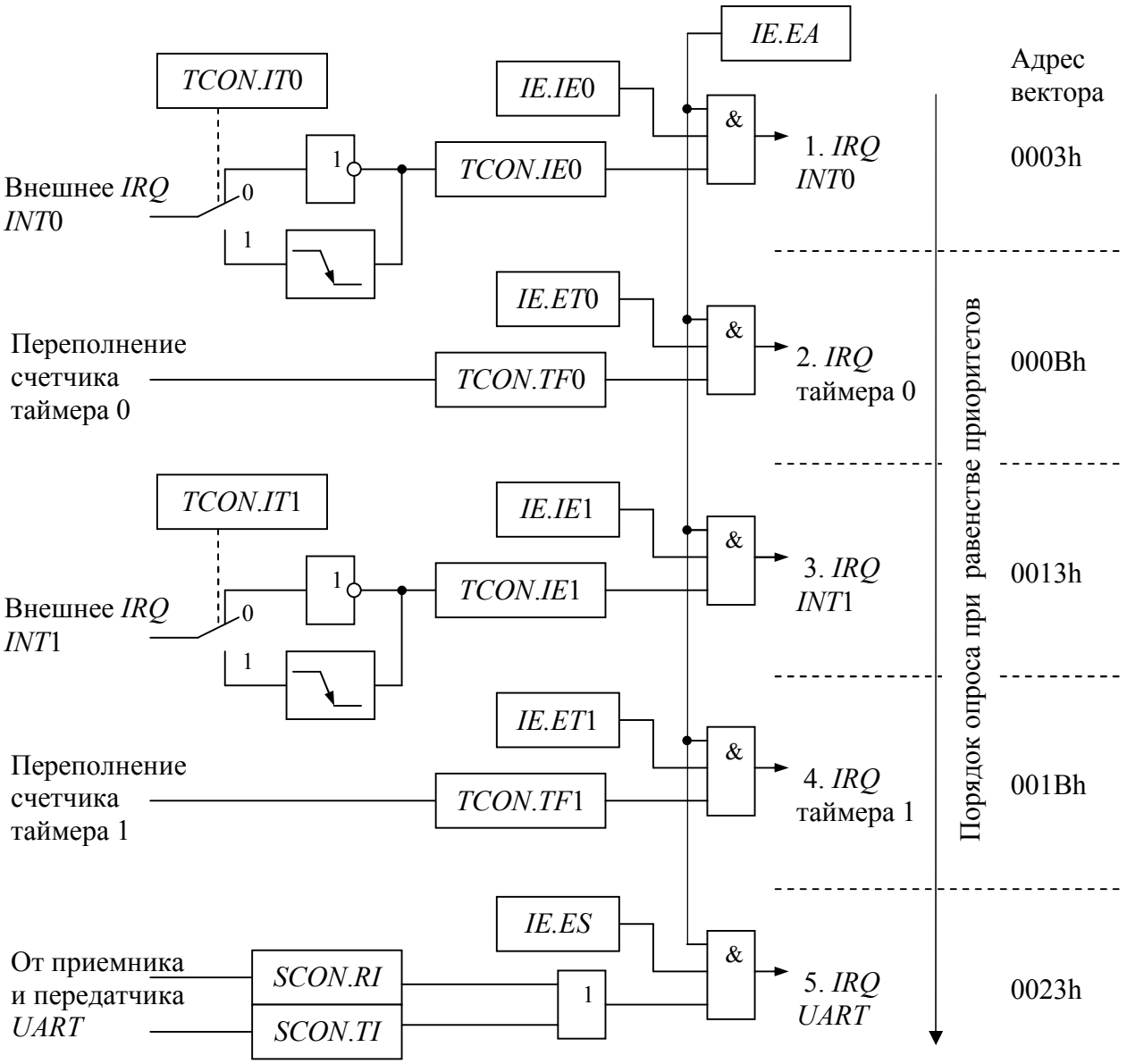

Рис. 1.5 – Схема формирования запросов прерываний в МК 8051

Механизм приоритетов прерываний предназначен для выбора одного из источников прерываний при одновременном приходе нескольких запросов, а также для решения вопроса о прерывании текущей программы обслуживания прерывания вновь поступившем запросом. Все источники прерываний проверяются на наличие запроса во время фазы *S*5*P*2 каждого машинного цикла (рис.1.6, кроме команды *RETI* и команд обращение к регистрам *IE*, *IP*). В течение следующего машинного цикла анализируются биты регистра приоритетов *IP* и выполняется внутренний выбор (поллинг) более приоритетного запроса.

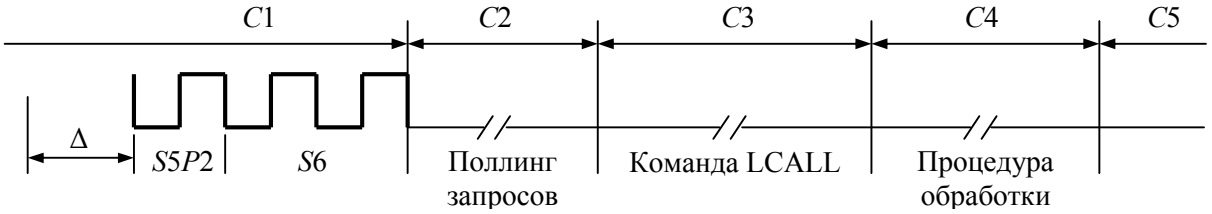

Рис. 1.6 – Циклы вызова процедуры обслуживания прерывания (наиболее быстрый вариант, когда *C*1 является последним машинным циклом команды)

Обработка запроса прерывания начинается после выполнения текущей команды прерываемой программы до конца и сохранения содержимого счетчика команд (адреса возврата) в стеке. Это происходит посредством аппаратно генерируемой команды *LCALL*. В результате в счетчик команд загружается адрес соответствующего обработчика прерываний (рис.1.5). Поскольку восьми байт выделенных под обработчик прерывания обычно мало, то в ячейке памяти по этому адресу размещают команду безусловного перехода к начальному адресу фактической подпрограммы обслуживания прерывания. Такая подпрограмма должна обеспечить сохранность используемых ею регистров и слова состояния (обычно они сохраняются в стеке). При переходе на подпрограмму обработки прерывания автоматически (независимо от регистра *IE*) запрещаются все прерывания с низким (бит регистра *IP* которых сброшен) и равным уровнем приоритета. Таким образом, прерывания с низким приоритетом могут прерываться запросами с высоким уровнем (бит регистра *IP* которых установлен), но обработка высокоприоритетного прерывания не может быть прервана. В конце обработчика прерывания обязательно должна быть записана команда возврата из подпрограммы *RETI*, которая загружает из стека в счетчик команд адрес возврата и разблокирует прерывания.

Минимальное время, необходимое на переход к обработчику прерывания, составляет 38 тактов. Максимальное время, необходимое на переход к обработчику прерывания для случая завершения текущей команды умножения или деления, составляет 86 тактов.

1.3 ПАРАЛЛЕЛЬНЫЙ ВВОД-ВЫВОД

Схемотехническая организация портов 8051 различна. На рис.1.7 показана обобщенная схема вывода параллельного порта 8051. Видно, например, что чтение данных возможно только при записи в этот порт логической единицы. Такая схема не позволяет получить значительных выходных токов, что часто усложняет проектирование устройств на основе этого МК.

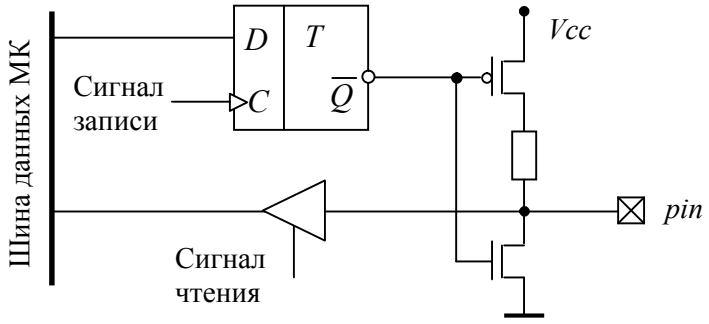

Рис. 1.7 – Обобщенная схема вывода параллельного порта

Помимо основной функции порта (прием/передача данных в параллельном коде) схемотехника портов всех МК (в т.ч. и 8051) предусматривает реализацию специальных функций (входы/выходы таймеров, компараторов, АЦП, ЦАП и служебных сигналов). Таким образом, физическая линия микросхемы (*pin*) разделяется во времени (одновременно используется) несколькими функциональными узлами МК.

Каждый из четырех портов МК 8051 содержит защелку – *D*-триггер, выходной драйвер (представляющий собой последовательное соединение транзисторов, управляемых парафазным сигналом) и входную цепь (типа повторитель с тремя состояниями). По сигналу сброса во все защелки портов записываются единицы – режим ввода данных. Каждая линия любого порта может быть индивидуально настроена как вход или выход. Если к МК подключена внешняя память, то порты *P*0 и *P*2 не могут использоваться, т.к. реализуют функцию мультиплексированной шины адреса/данных.

Выходные драйверы (каскады) портов *P*1-*P*3 выполнены на полевых транзисторах с внутренней нагрузкой, а драйвер порта *P*0 – на транзисторах с открытым стоком.

На рис.1.8 представлена схема порта *P*0.

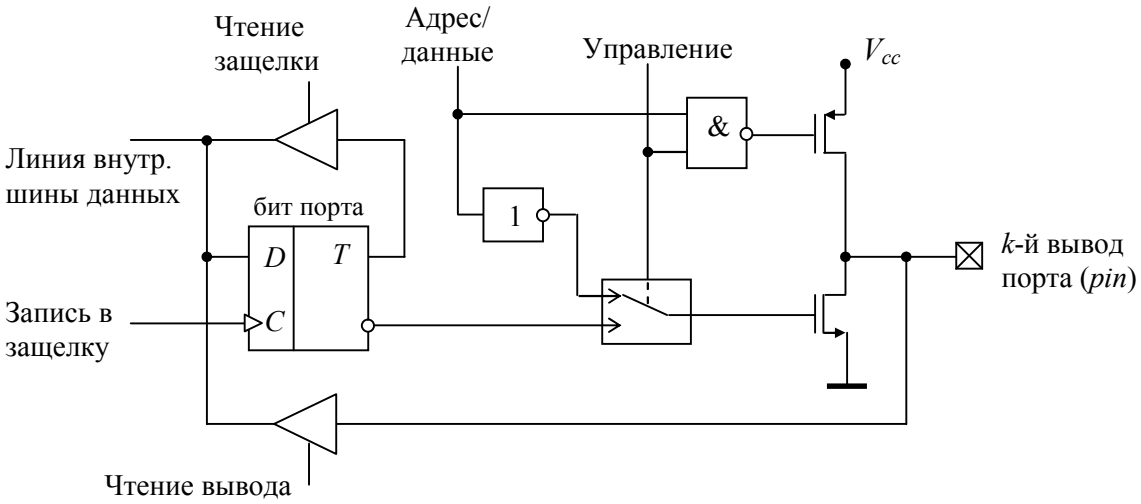

Рис. 1.8 – Схема вывода параллельного порта *P*0

Высокий уровень сигнала "управление" позволяет перевести линии порта в режим, когда к МК подключена внешняя память, при этом сигнал "адрес/данные" выполняет роль линии мультиплексированной шины адреса/данных (младшие восемь бит). При низком уровне сигнала "управление" схема работает в режиме порта (*k*-й бит регистра), при этом транзистор верхнего плеча драйвера закрыт и получается схема открытого стока (при использовании могут

понадобиться внешние подтягивающие резисторы). Для чтения вывода порта необходимо в защелку записать "1", это закроет транзистор нижнего плеча драйвера и обеспечит третье состояния вывода (состояние высокого импеданса).

На рис.1.9 представлена схема порта *P*2, которая отличается от схемы порта *P*0 (рис.1.8) наличием встроенного подтягивающего резистора *R*, поэтому этот порт не может обеспечить состояние высокого импеданса. Высокий уровень сигнала "управление" позволяет перевести линии порта в режим, когда к МК подключена внешняя память, при этом сигнал "адрес/данные" выполняет роль линии мультиплексированной шины адреса/данных (старшие восемь бит). При низком уровне сигнала "управление" схема работает в режиме порта (*k*-й бит регистра). Для чтения вывода порта необходимо в защелку записать "1", это закроет транзистор нижнего плеча драйвера и обеспечит возможность прохождения сигнала с вывода (*pin*) к внутренней шине МК с подтягиванием к уровню напряжения питания.

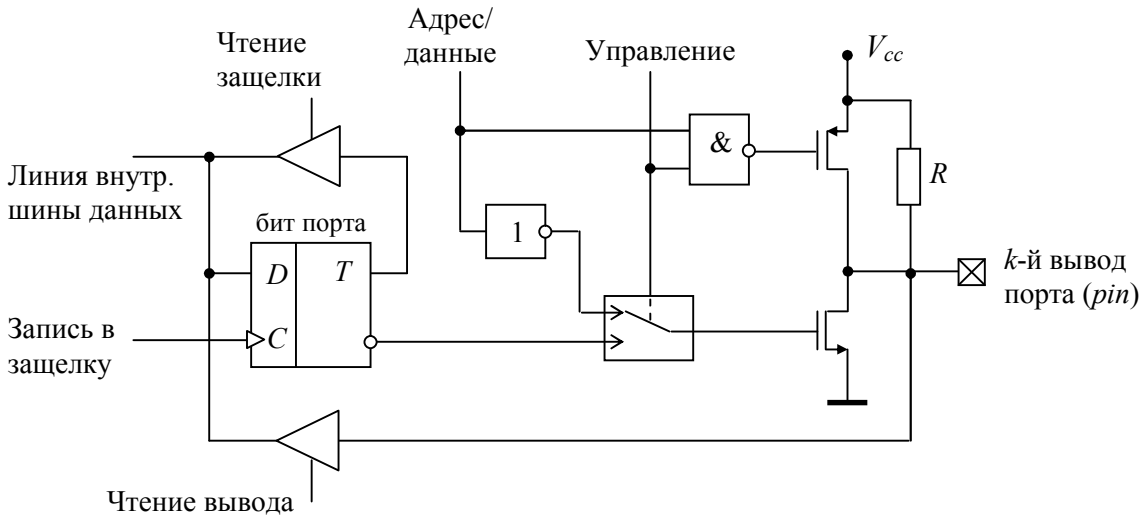

Рис. 1.9 – Схема вывода параллельного порта *P*2

На рис.1.10 представлена схема порта *P*1. Для чтения вывода порта необходимо в защелку записать "1", это закроет транзистор нижнего плеча драйвера и обеспечит возможность прохождения сигнала с вывода (*pin*) к внутренней шине МК с подтягиванием к уровню напряжения питания.

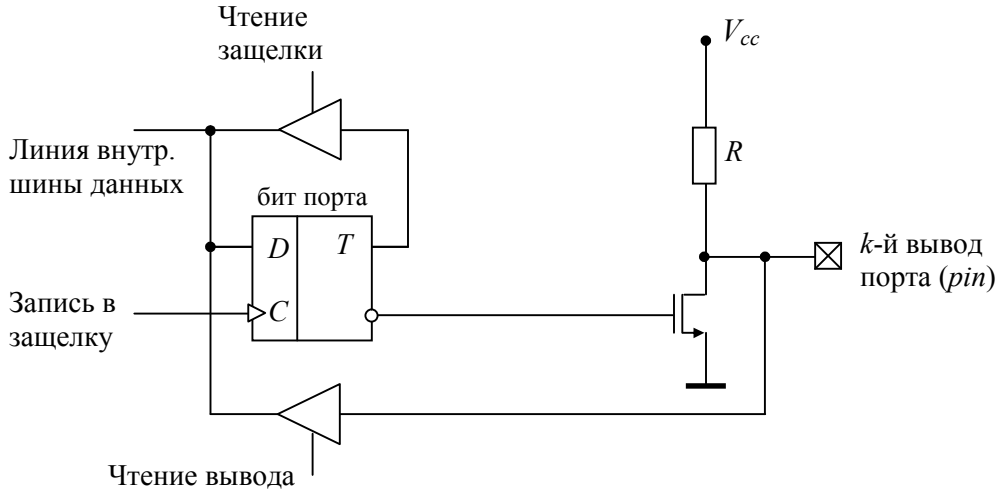

Рис. 1.10 – Схема вывода параллельного порта *P*1

На рис.1.11 представлена схема порта *P*3. Каждая линия порта реализует специальную функцию (альтернативное назначение). Для включения альтернативной функции линии порта необходимо записать в защелку "1", а для его чтения еще необходимо, чтобы сигнал "альтернативный сигнал выхода" был высокого уровня.

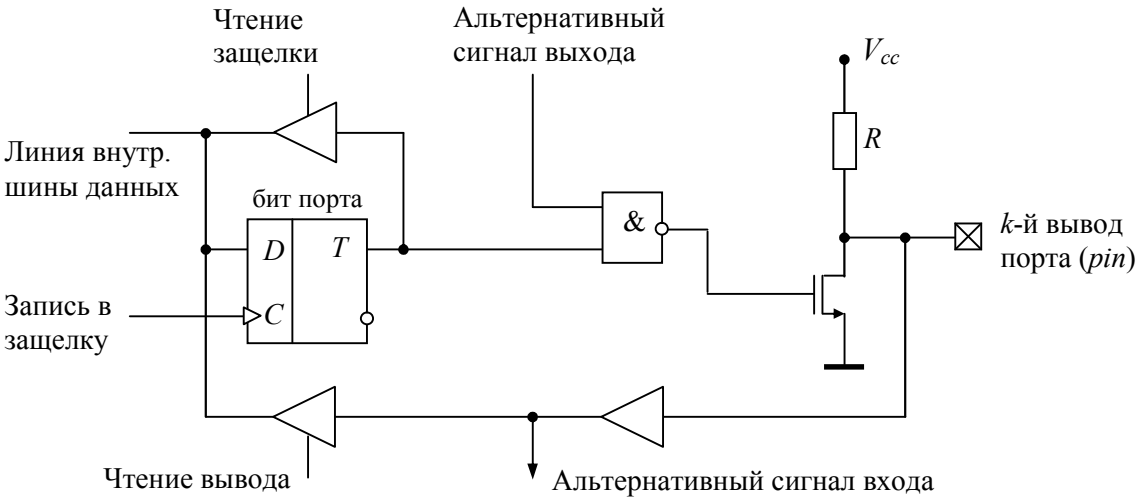

Рис. 1.11 – Схема вывода параллельного порта *P*3

Обращение к портам происходит с помощью команд, оперирующих с байтом, отдельным битом или произвольной группой бит. При этом в одной команде порт может быть и источником и приемником данных, в этом случае устройство управления МК автоматически включает режим "чтение-модификация-запись". Этот режим предполагает чтение защелки (а не линии порта) изменение значения и запись результата обратно в защелку в цикле одной команды (отметим, что содержимое защелки и состояние внешней линии порта могут не совпадать).

1.4 ТАЙМЕРЫ

В базовых моделях МК 8051 имеются два программируемых 16-разрядных таймера/счетчика. На рис.1.12 представлена упрощенная структура таймера/счетчика.

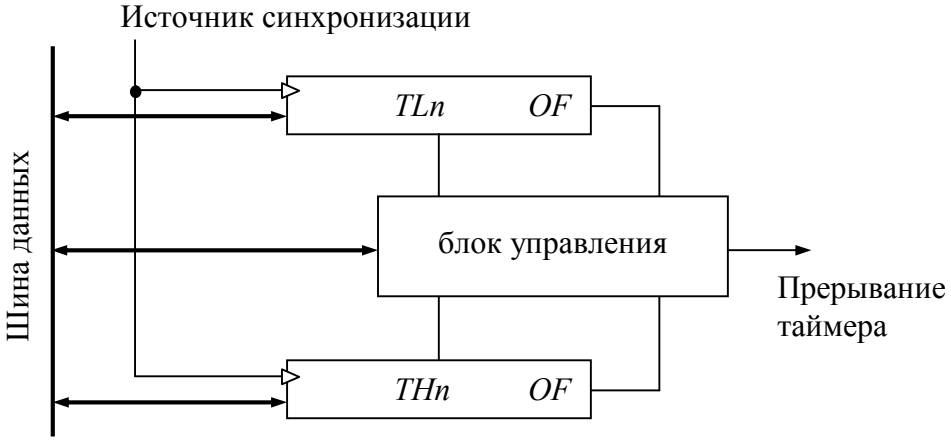

Рис. 1.12 – Упрощенная структура таймера МК 8051

В этой структуре *TLn* и *THn* являются 8-разрядными счетчиками-регистрами и представляют младшие и старшие разряды таймера, где *n* – номер таймера (0, 1 и более – для старших моделей). В качестве источника синхронизации служат синхроимпульсы МК, поступающие с *prescaler* (входного делителя). Кроме того, таймер может синхронизироваться от внешнего источника (входы *Tn*). Опрос внешнего входного сигнала *Tn* осуществляется в фазе *S*5*P*2 каждого машинного цикла (аналогично опросу входов внешних прерываний – рис.10). Инкремент счетчика происходит при фиксации изменения входного сигнала с высокого уровня на низкий (т.е. в предыдущем цикле была зафиксирована "1", а в следующем – "0"). Новое значение счетчика (инкрементированное) будет сформировано в середине цикла следующего за циклом, в котором обнаружен переход с "1" на "0" входа *Tn*. Так как для распознавания периода требуется два машинных цикла (12 тактов каждый), то максимальная частота переключения входного сигнала *Tn* ограничена величиной 1/24 от тактовой частоты генератора МК. Длительность входных сигналов *Tn* должна быть не меньше длительности одного цикла МК. Например, при тактовой частоте МК F_{clk}=12 МГц диапазон изменения интервала времени таймера составляет 1-65536 мкс, а частота переключений на входе *Tn* – не более 500 кГц.

Такая организация таймера позволяет реализовывать часто востребованные функции:

- реализация меток реального времени;
- измерение числа событий за определенный промежуток времени;
- измерение длительности импульса;
- измерение частоты сигнала;
- реализация (программно-аппаратная) ШИМ сигнала.

В МК 8051 существует четыре режима работы таймера:

1) Режим 0 – таймер конфигурируется как 13-разрядный счетчик.
2) Режим 1 – таймер конфигурируется как 16-разрядный счетчик.
3) Режим 2 – применяется для задания скорости последовательного порта. Когда содержимое счетчика *TLn* переполняется, в него загружается содержимое счетчика *THn*, который работает как регистр (не изменяется в ходе счета, но доступен программно). И счет возобновляется, а сигнал переполнения используется для задания скорости обмена последовательного порта (*UART*).
4) Режим 3 – *TLn* и *THn* работают как два независимых 8-разрядных счетчика (этот режим обеспечивается только таймером 0). Счетчик *TL*0 переключается внешним сигналом, а *TH*0 – внутренними тактовыми сигналами (реализуется тахометр).

Для управления таймерами/счетчиками в МК предусмотрены два регистра специальных функций: *TMOD* (089h) и *TCON* (088h).

Назначение бит регистра *TMOD* приводится в нижеследующей таблице.

Номер бита	7	6	5	4	3	2	1	0
Наименование	*Gate*	*C/T*	*M*1	*M*0	*Gate*	*C/T*	*M*1	*M*0
	Таймер/счетчик 1				Таймер/счетчик 0			

Gate – управление блокировкой	=0 – таймер разрешен, если бит *TCON.TRn* установлен =1 – таймер разрешен, если на входе внешнего прерывания *INTn* высокий уровень и бит *TCON.TRn* установлен
C/T – управление тактированием	=0 – таймер работает от внутреннего генератора с делителем на 12 =1 – таймер работает от сигнала на внешнем входе *Tn*
*M*1, *M*0 – выбор режима работы	0 0 – режим 0 (рис.1.13А), 13-бит счетчик 0 1 – режим 1 (рис.1.13Б), 16-бит счетчик 1 0 – режим 2 (рис.1.14), 8-бит счетчик с перезагрузкой 1 1 – режим 3 (рис.1.15), два независимых 8-бит счетчика (таймер 0)

Назначение бит регистра *TCON* приводится в нижеследующей таблице.

Номер бита	7	6	5	4	3	2	1	0
Наименование	*TF*1	*TR*1	*TF*0	*TR*0	*IE*1	*IT*1	*IE*0	*IT*0
	Таймер/счетчик 1		Таймер/счетчик 0		см. раздел 1.2, внешние прерывания			

TFn – флаг переполнения счетчика
TRn – сигнал разрешения работы счетчика (=0 – не считает, запрещен)

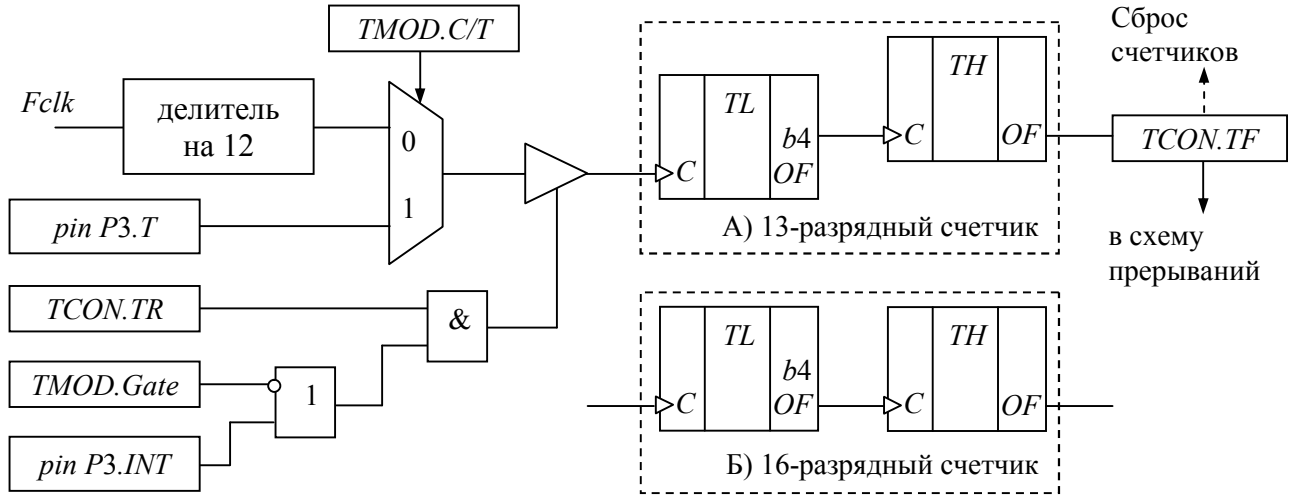

Рис. 1.13 – Схема таймера/счетчика в режимах 0 и 1 (варианты А и Б соответственно)

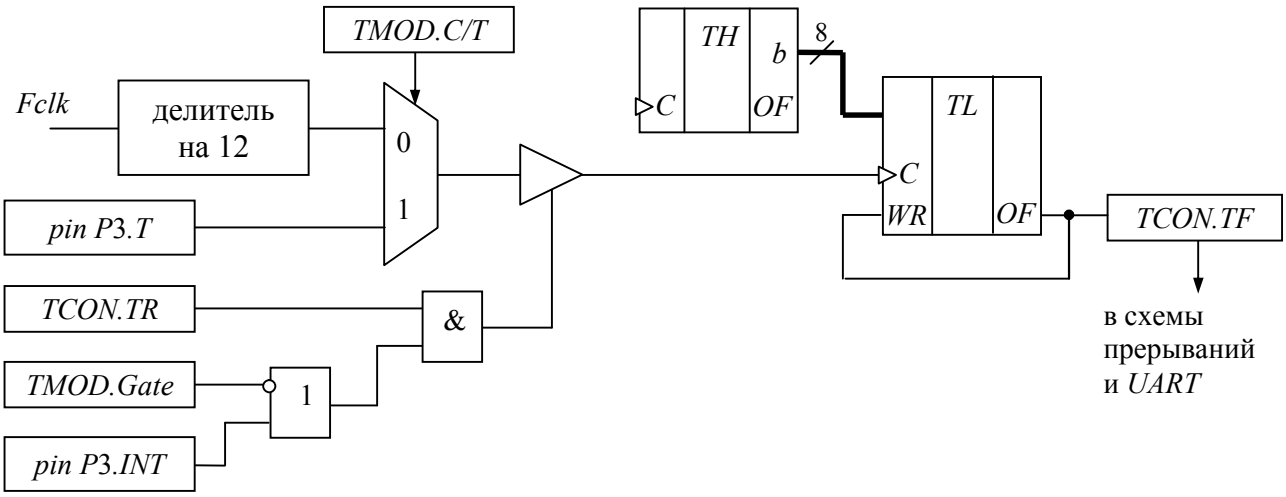

Рис. 1.14 – Схема таймера/счетчика в режиме 2

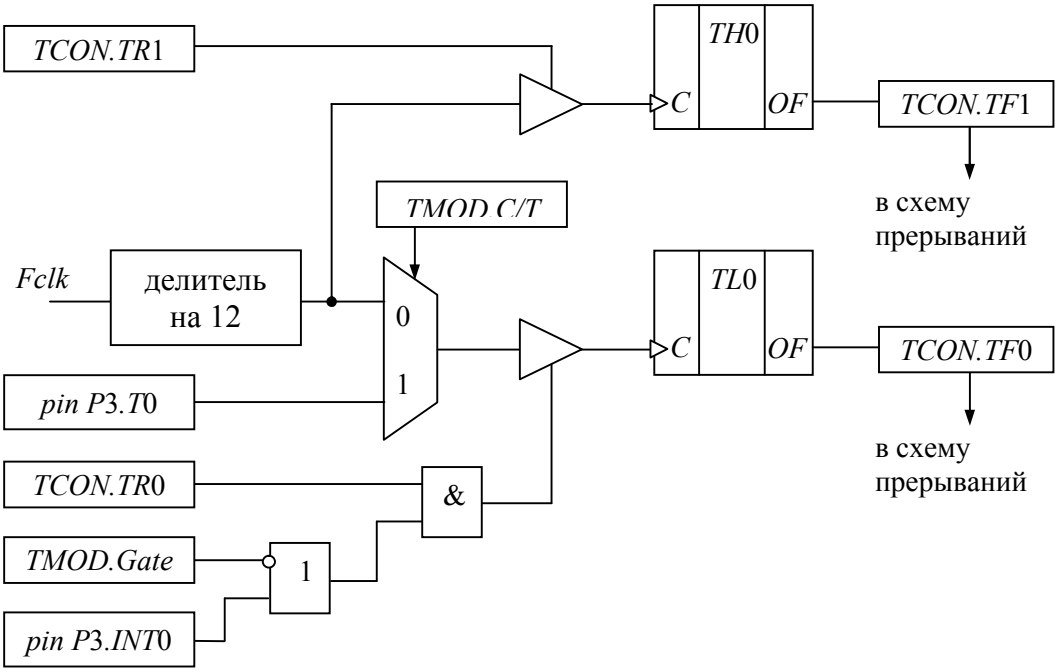

Рис. 1.15 – Схема таймера/счетчика 0 в режиме 3

(*) При включении у таймера/счетчика0 режима 3 таймер/счетчик1 лишается своего бита управления *TCON.TR*1 и флага прерывания *TCON.TF*1. Однако сохраняет свою работоспособность без прерываний. Поэтому в режимах 0, 1, 2 при *TMOD.Gate*1=0 всегда включен и при переполнении в режимах 0 и 1 обнуляется, а в режиме 2 перезагружается, не устанавливая флага прерываний *TCON.TF*1. Управление от входов *P*3.*INT*1, *P*3.*T*1 и применение битов управления *TMOD.C/T*1, *TMOD.Gate*1 для таймера/счетчика1 не зависят от режима таймера/счетчика 0.

1.5 ПОСЛЕДОВАТЕЛЬНЫЙ ВВОД-ВЫВОД

Последовательный порт (универсальный приемопередатчик – *UART*) 8051 способен функционировать как в асинхронном режиме, так и в синхронном. При этом передача данных происходит младшими битами вперед. Упрощенная структурная схема последовательного порта представлена на рис.1.16.

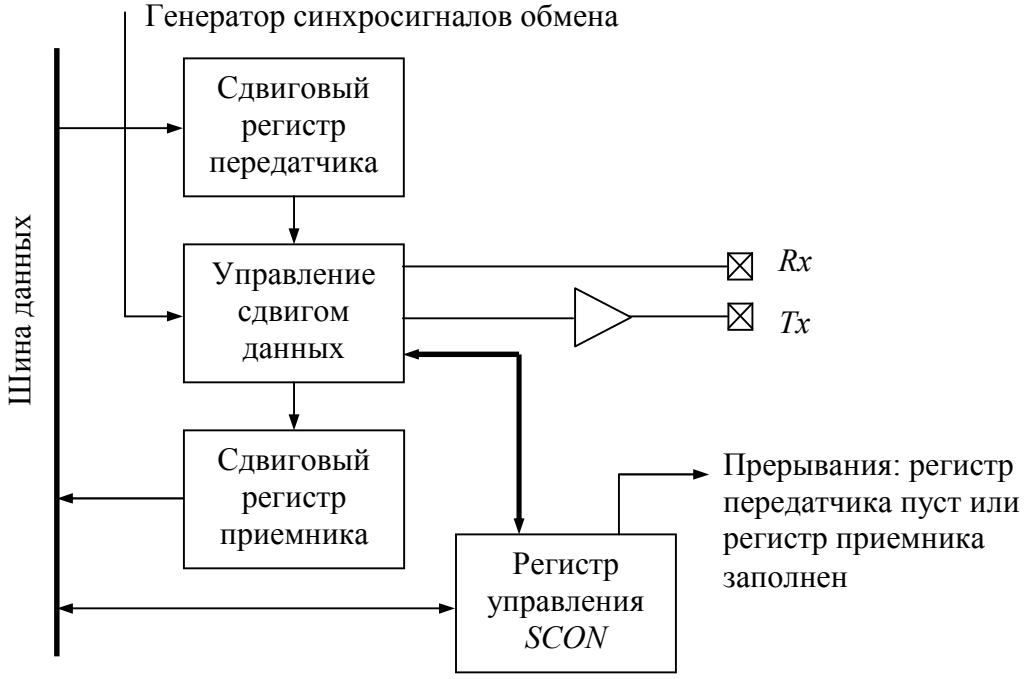

Рис. 1.16 – Структурная схема последовательного порта

Для управления работой интерфейса в МК предусмотрен регистр *SCON* (098h), назначение бит которого приводится в нижеследующей таблице.

Бит		Назначение
0	*RI*	Флаг прерывания приемника (активен, если принят байт данных)
1	*TI*	Флаг прерывания передатчика (активен, если байт передан в линию)
2	*RB*8	Прием 9-го бита для режимов 2 и 3 или стоп-бита для режима 1
3	*TB*8	Передача 9-го бита для режимов 2 и 3
4	*REN*	Разрешение/запрет приема данных
5	SM2	Устанавливается программно для запрета приема сообщений, в которых девятый бит имеет значение 0 (режимы 1, 2 и 3)
6	*SM*1	Режим работы *UART* (*SM*1 *SM*0): 0 0 - режим 0 (синхронный обмен) 0 1 - режим 1 (асинхронный обмен, 8 бит, изменяемая скорость передачи) 1 0 - режим 2 (асинхронный обмен, 9 бит, фиксированная скорость передачи) 1 1 - режим 3 (асинхронный обмен, 9 бит, изменяемая скорость передачи)
7	*SM*0	

Режим 0 (синхронный обмен). Данные посылаются и принимаются по линии *Rx* (полудуплексный режим), а синхросигналы выдаются по линии *Tx*. В этом режиме МК может быть только ведущим устройством. Передача данных инициируется записью слова в регистр данных порта и ведется по переднему фронту тактового сигнала. После приема слова может быть реализовано аппаратное прерывание. Максимальная скорость передачи бит данных определяется частотой тактирования МК деленной на 12.

Режим 1 (асинхронный обмен). Данные выдаются через линию *Tx*, а принимаются через *Rx* (полнодуплексный режим). Пакет передачи представляет собой: старт-бит, 8 бит данных,

стоп-бит. Причем при приеме данных стоп-бит заносится в разряд *RB*8 регистра *SCON*. Скорость работы порта задается с помощью таймера/счетчика1.

Режим 2 (асинхронный обмен). Данные выдаются через линию *Tx*, а принимаются через *Rx* (полнодуплексный режим). Пакет передачи представляет собой: старт-бит, 8 бит данных, программируемый бит, стоп-бит. При передаче и приеме программируемый бит связан с битами *TB*8 и *RB*8 регистра *SCON* соответственно. Его можно интерпретировать как 9-й бит данных или бит четности. Стоповый бит в отличие от режима 1 теряется. Скорость передачи в этом режиме равна либо 1/32, либо 1/64 – определяется управляющим битом *SMOD* регистра *PCON* ("0" – 1/64; "1" – 1/32).

Режим 3 – аналогичен режиму 2, но скорость определяется таймером/счетчиком1.

Во всех режимах передача данных инициируется записью байта в регистр передатчика *SBUF*. Это означает, что для достижения максимальной скорости обмена МК должен отслеживать момент окончания передачи байта для загрузки следующего байта. Этот момент можно контролировать с помощью прерывания.

Во всех режимах флаг прерывания передатчика *SCON.TI* устанавливается аппаратно в конце передачи стоп-бита.

Флаг приемника *SCON.RI* устанавливается аппаратно в конце приема 8-го бита данных в режиме 0 и в середине приема стоп-бита в режимах 1,2,3.

Подпрограмма обработки прерывания *UART* должна сбрасывать соответствующие флаги *RI* и *TI* в регистре *SCON*.

В режимах (1 и 3), где источником тактирования является таймер 1, бит *SMOD* регистра *PCON* также влияет на частоту работы последовательного порта – делит частоту сигнала переполнения таймера/счетчика1 на 32 (*SMOD*=1) или на 64 (*SMOD*=0).

На рис. 1.17а-в представлены для ознакомления временные диаграмма работы узла в всех режимах.

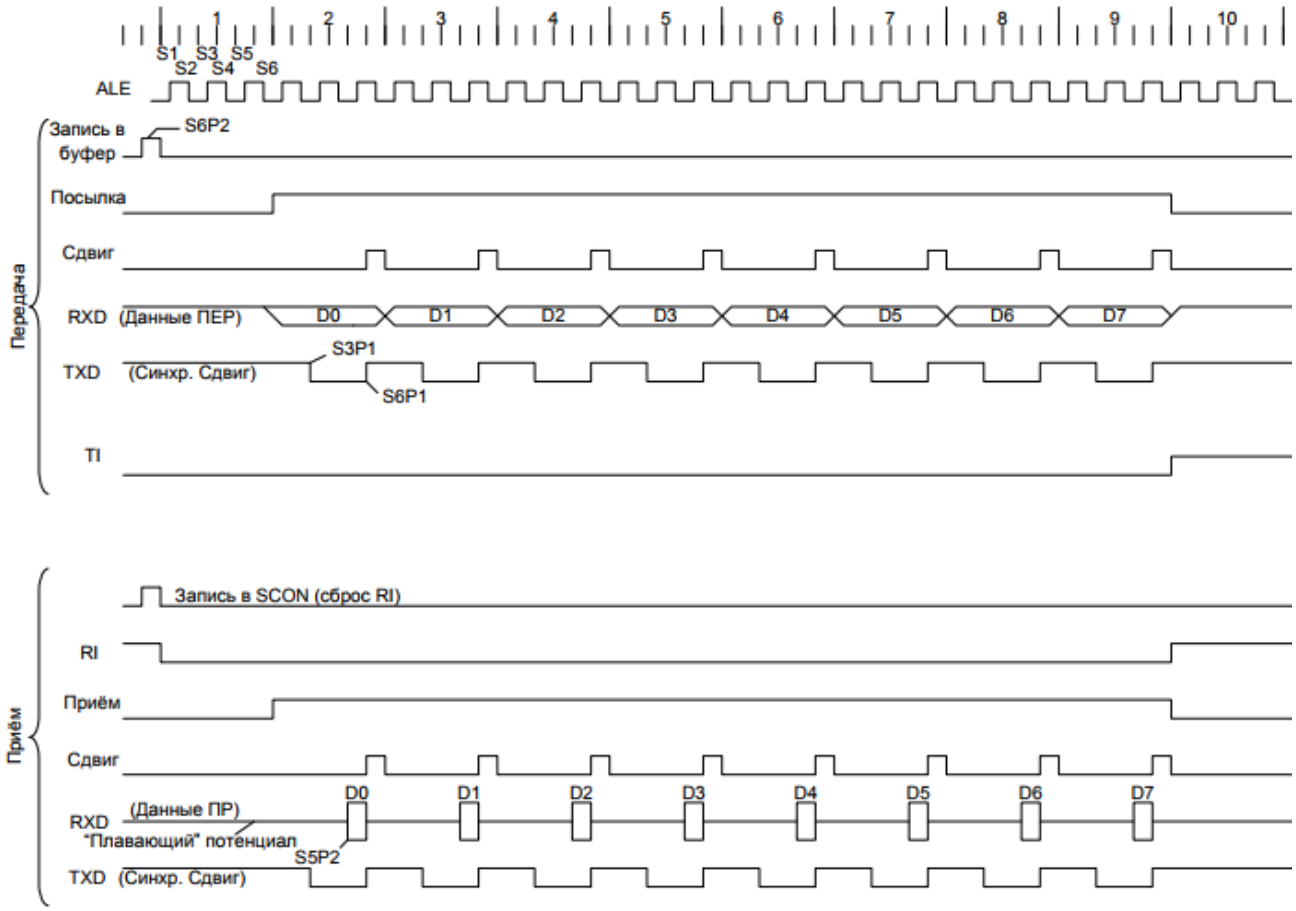

Рис. 1.17а - Временная диаграмма работы *UART* в режиме 0

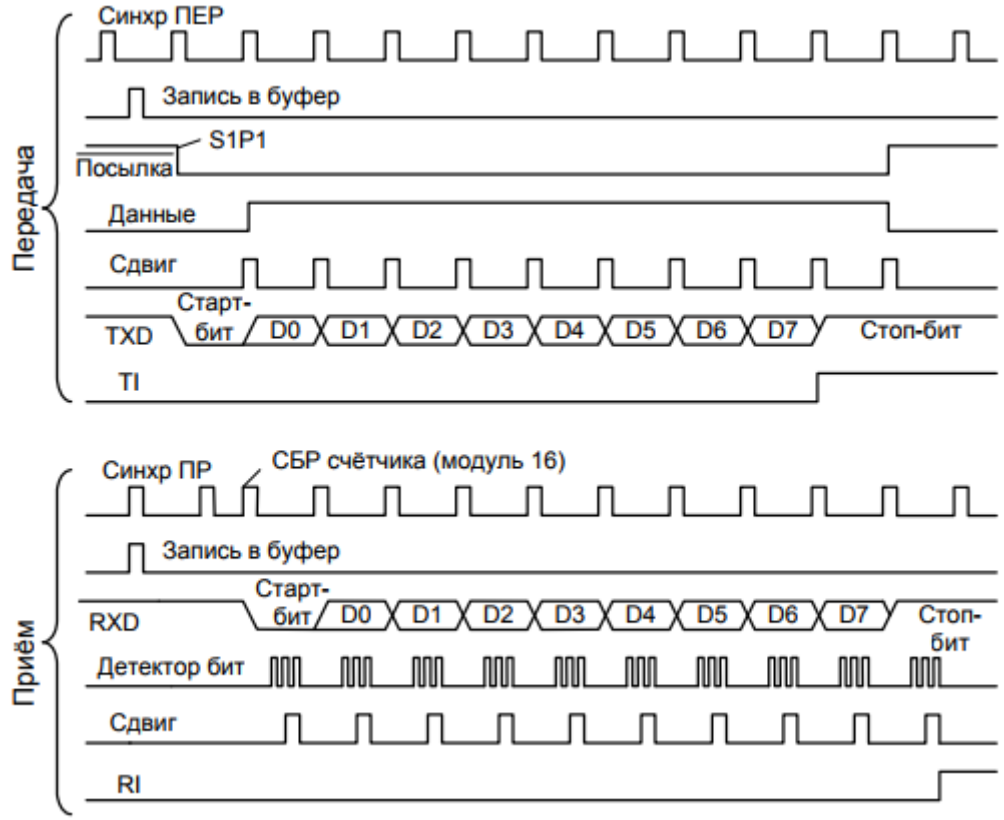

Рис. 1.17б - Временная диаграмма работы *UART* в режиме 1

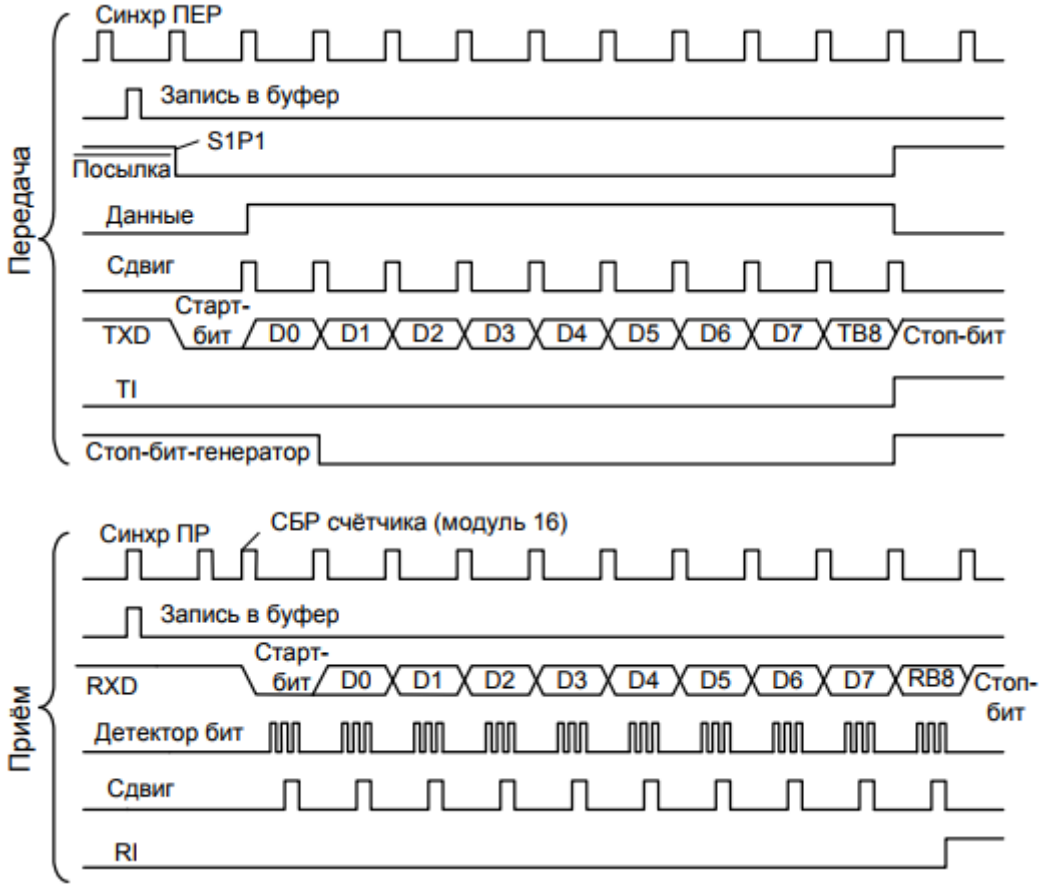

Рис. 1.17в - Временная диаграмма работы *UART* в режимах 2 и 3

1.6 ЭНЕРГОПОТРЕБЛЕНИЕ МК

Одной из ключевых возможностей любого МК является способность снижать свое энергопотребление, отключая часть внутренних узлов и/или понижать частоту тактовых импульсов. Не исключение и МК i8051. Он предоставляет два режима пониженного энергопотребления.

1. Режим холостого хода. В этом режиме блоки прерываний, таймеров и последовательного порта продолжают тактироваться, все регистры сохраняют свое значение, все линии портов ввода-вывода сохраняют свое активное состояние (удерживают логический уровень, который был на момент перехода в режим XX). Однако ядро отключается от тактового генератора. На выводах *ALE*, *PSEN* устанавливается высокий уровень. Режим активируется при установке бита *IDL* в регистре *PCON*. Выход из режима осуществляется по сигналу сброса или при поступлении любого сигнала прерывания. В последнем случае после выполнения процедуры обработки прерывания выполнение программы продолжается с команды, следующей за командой установившей бит *PCON.IDL*.
2. Режим выключенного питания (*Power Down Mode*). В этом режиме не тактируются: ядро, таймеры, *UART*, система прерываний. Однако состояние внутренней памяти (в т.ч. регистры) и линий портов ввода-вывода остаются неизменными. На выводах *ALE*, *PSEN* устанавливается низкий уровень. Режим включается установкой бита *PD* регистра *PCON*. При этом напряжение питание с вывода V_{cc} (*pin* 40) можно снизить или отключить, но необходимо к выводу *Reset* подключить резервное питание, обеспечивающее сохранность памяти и регистров. Выход из режима осуществляется только по сигналу сброса.

(*) При одновременной установке битов *IDL* и *PD* регистра *PCON* последний имеет приоритет – будет включен режим *PDM*.

Регистр *PCON* (87h) и назначение его разрядов приводится в нижеследующей таблице.

Бит		Назначение
0	*IDL*	Бит холостого хода. Установка в «1» переводит МК в режим пониженного энергопотребления – режим холостого хода.
1	*PD*	Бит пониженной мощности. При установке в «1» МК переходит в *power down mode*.
2	*GF*0	Флаги общего назначения – могут использоваться по усмотрению пользователя.
3	*GF*1	
4	-	Резерв. Не рекомендуется записывать в эти биты «1».
5	-	
6	-	
7	*SMOD*	Удвоенная скорость передачи. Установка в «1» удваивает скорость *UART* – см. раздел 1.5.

При сбросе *PCON*=0xxx0000. Биты 3-0 регистра поддерживаются не всеми моделями i8051.

1.7 СТОРОЖЕВОЙ ТАЙМЕР

На рис.1.18 представлена структурная схема сторожевого таймера МК 8051.

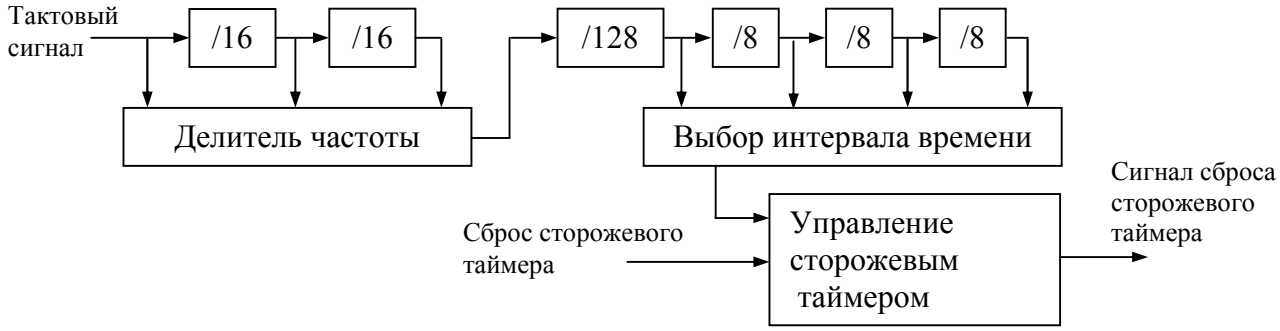

Рис. 1.18 – Структурная схема сторожевого таймера 8051

Управляемый программно делитель и блок выбора интервала времени позволяет запрограммировать сторожевой таймер на отсчет интервалов длительностью от 10 мс до 24 мин при тактовой частоте 12 МГц.

Для сброса сторожевого таймера, т.е. для предотвращения сброса МК необходимо выполнить следующую последовательность команд:

```
mov 0C7, #0AAh;       послать контрольные символы
mov 0C7, #055h
mov WDCON, #002h;     сбросить сторожевой таймер
```

1.8 НАЗНАЧЕНИЕ ВЫВОДОВ МК 8051

Классический i8051 (*MCS*-51) и отечественный аналог КМ1816ВЕ51 выполнены на основе *n*-МОП технологии и выпускались в 40-выводном корпусе – рис.1.19.

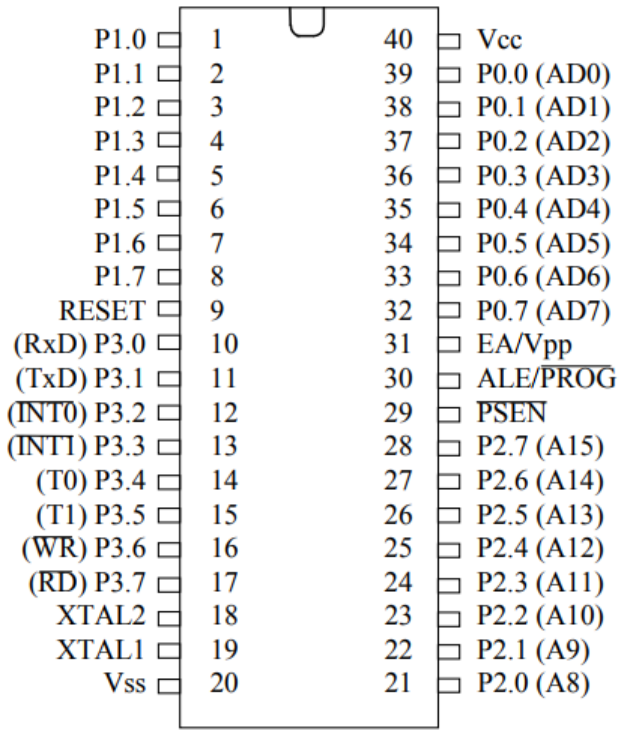

Рис. 1.19 – Цоколевка микросхемы *MSC*-51

Назначение выводов.

Vss - потенциал общего провода "земля"

Vcc - напряжение питания +5 В

*XTAL*1, *XTAL*2 - выводы подключения кварцевого резонатора

Reset (*RST*) - вход общего сброса МК (не влияет на ячейки памяти данных)

PSEN - строб внешней памяти программ, выдается только при обращении к внешнему ПЗУ

ALE - строб адреса внешней памяти

EA - отключение внутренней памяти программ, низкий уровень заставляет МК выполнять программу из внешнего ПЗУ игнорируя внутреннее

*P*0 - 8-разрядный порт ввода-вывода, при работе с внешними ОЗУ и ПЗУ эти линии мультиплексированы с адресом (младшие линии адреса)

*P*1 - 8-разрядный порт ввода-вывода, каждый разряд порта может быть независимо запрограммирован на вход или выход

*P*2 - 8-разрядный порт ввода-вывода, аналогичный P0, при работе с внешними ОЗУ и ПЗУ эти линии мультиплексированы с адресом (старшие линии адреса)

*P*3 - 8-разрядный порт ввода-вывода, мультиплексирован с узлами специальных функций (*UART*, таймеры, *IRQ*, стробы внешней памяти).

На рис.1.20 представлена структурная схема контроллера *MCS*-51.

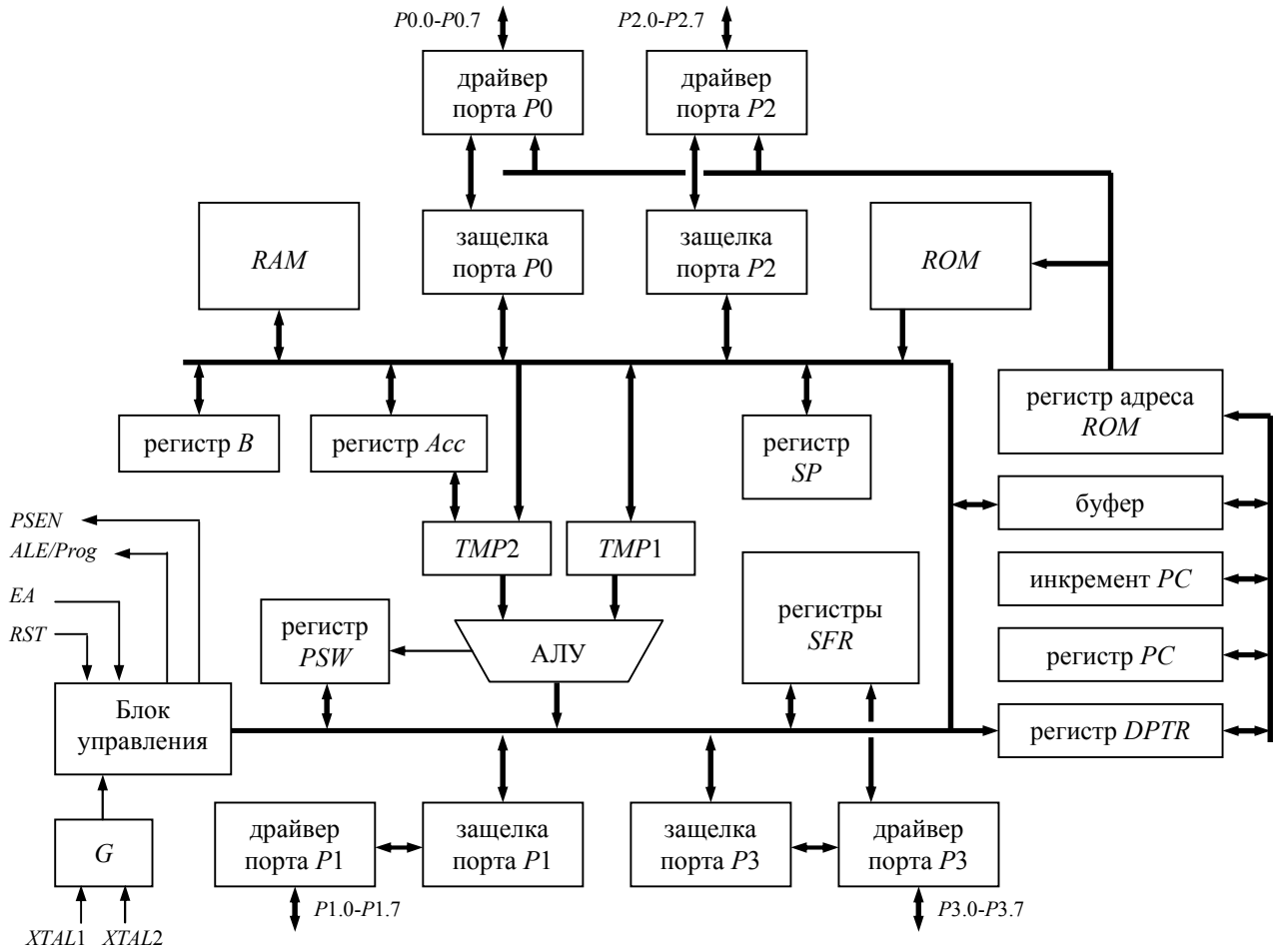

Рис. 1.20 – Структурная схема *MSC*-51

1.9 СИСТЕМА КОМАНД 8051

Система команд МК 8051 содержит 111 базовых команд, которые по функциональному признаку могут быть разделены на пять групп:

1) Команды передачи данных;
2) Арифметические операции;
3) Логические операции;
4) Операции с битами;
5) Команды передачи управления.

Все команды с точки зрения кодирования представляются либо однобайтовыми, либо двухбайтовыми, либо трехбайтовыми – в соответствии с этим признаком определяют 13 типов команд – рис. 1.21, где

КОП – код операции;
#d – 8-разрядная численная константа (хранится в памяти программ);
#d16 – 16-разрядная численная константа (хранится в памяти программ);
ad (ad1, ad2) – 8-разрядный адрес (хранится в памяти программ);
ad16 – 16-разрядный адрес (хранится в памяти программ);
bit – номер бита (хранится в памяти программ);
rel – относительное смещение (хранится в памяти программ).

	1-й байт $d_7 \dots d_0$	2-й байт $d_7 \dots d_0$	3-й байт $d_7 \dots d_0$
1	КОП		
2	КОП	#d	
3	КОП	ad	
4	КОП	bit	
5	КОП	rel	
6	$a_{10}a_9a_8$ КОП	$a_7 \dots a_0$	
7	КОП	ad	#d
8	КОП	ad	rel
9	КОП	ad1	ad2
10	КОП	#d	rel
11	КОП	bit	rel
12	КОП	ad16(high)	ad16(low)
13	КОП	#d16(high)	#d16(low)

Рис. 1.21 – Типы команд МК 8051

Определение и ассемблерная мнемоника команд, их тип в соответствии с рис. 1.21 (Т - тип команды: от 1 до 13), число байтов в командах (Б), а также продолжительность исполнения команд в циклах (Ц) даны в нижеследующей таблице.

Команды передачи данных	Т Б Ц	Операция

Пересылка аккумулятор - регистр (*n*=0÷7)
MOV A, Rn
MOV Rn, A
	1 1 1	
		(A)←(Rn)
		(Rn)←(A)

Пересылка аккумулятор - прямоадресуемый байт
MOV A, ad
MOV ad, A
	3 2 1	
		(A)←(ad)
		(ad)←(A)

Пересылка аккумулятор - байт из ПД (*i*=0,1)
MOV A, @Ri
MOV @Ri, A
	1 1 1	
		(A)←((Ri))
		((Ri))←(A)

Загрузка в аккумулятор константы MOV A, #d	2 2 1	 (A)←#d
Загрузка в регистр константы MOV Rn, #d	2 2 1	 (Rn)←#d

Пересылка регистр - прямоадресуемый байт
MOV Rn, ad
MOV ad, Rn
	3 2 2	
		(Rn)←(ad)
		(ad)←(Rn)

Пересылка байта ПД по прямому адресу
MOV ad, @Ri
MOV @Ri, ad
	3 2 2	
		(ad)←((Ri))
		((Ri))←(ad)

Пересылка прямоадресумая "память - память" MOV add, ads	9 3 2	 (add)←(ads)
Пересылка по прямому адресу константы MOV ad, #d	7 3 2	 (ad)←#d
Пересылка в ПД константы MOV @Ri, #d	2 2 1	 ((Ri))←#d
Загрузка указателя данных MOV DPTR, #d16	13 3 2	 (DPTR)←#d16

Пересылка в аккумулятор байта из ПП
MOVC A, @A+DPTR
MOVC A, @A+PC
	1 1 2	
		(A)←((A) + (DPTR))
		(PC)←(PC)+1, (A)←((A)+(PC))

Пересылка аккумулятор - байт из внешней ПД
MOVX A, @Ri
MOVX @Ri, A
MOVX A, @DPTR
MOVX @DPTR, A
	1 1 2	
		(A)←((Ri))
		((Ri))←(A)
		(A)←((DPTR))
		((DPTR))←(A)

Стек: загрузка/выгрузка PUSH ad POP ad	3 2 2	 (SP)←(SP)+1, ((SP))←(ad) (ad)←(SP), (SP)←(SP)–1
Обмен аккумулятора с регистром XCH A, Rn	1 1 1	 (A)↔(Rn)
Обмен аккумулятора с прямоадресуемым байтом XCH A, ad	3 2 1	 (A)↔(ad)
Обмен аккумулятора с байтом из ПД XCH A, @Ri	1 1 1	 (A)↔((Ri))
Обмен младших тетрад аккумулятора и байта ПД XCHD A, @Ri	1 1 1	 $(A_{0...3})\leftrightarrow((Ri)_{0...3})$

Арифметические команды	Т Б Ц	Операция

Сложение аккумулятора с константой
ADD A, #d
ADDC A, #d
	2 2 1	
		(A)←(A)+#d
		(A)←(A)+#d+(C)

Сложение аккумулятора с регистром (*n*=0÷7)
ADD A, Rn
ADDC A, Rn
	1 1 1	
		(A)←(A)+(Rn)
		(A)←(A)+(Rn)+(C)

Сложение аккумул. с прямоадресуемым байтом
ADD A, ad
ADDC A, ad
	3 2 1	
		(A)←(A)+(ad)
		(A)←(A)+(ad)+(C)

Сложение аккумулятора с байтом из ПД (*i*=0,1)
ADD A, @Ri
ADDC A, @Ri
	1 1 1	
		(A)←(A)+((Ri))
		(A)←(A)+((Ri))+(C)

Десятичная коррекция аккумулятора DA A	1 1 1	Если ($A_{0...3}$)>9 или ((AC)=1), то ($A_{0...3}$)←($A_{0...3}$) + 6, затем если ($A_{4...7}$) >9 или ((C)=1), то ($A_{4...7}$)←($A_{4...7}$) + 6
Вычитание из аккумулятора регистра и заёма SUBB A, Rn	1 1 1	 (A)←(A)–(C)–(Rn)
Вычитание из аккумулятора прямоадресуемого байта и заема SUBB A, ad	3 2 1	 (A)←(A)–(C)–((ad))
Вычитание из аккумулятора байта ПД и заема SUBB A, @Ri	1 1 1	 (A)←(A)–(C)–((Ri))
Вычитание из аккумулятора константы и заема SUBB A, #d	2 2 1	 (A)←(A)–(C)–#d
Инкремент INC A INC Rn INC ad INC @Ri INC DPTR	 1 1 1 1 1 1 3 2 1 1 1 1 1 1 2	 (A)←(A)+1 (Rn)←(Rn)+1 (ad)←(ad)+1 ((Ri))←((Ri))+1 (DPTR)←(DPTR)+1
Декремент DEC A DEC Rn DEC ad DEC @Ri	 1 1 1 1 1 1 3 2 1 1 1 1	 (A)←(A)–1 (Rn)←(Rn)–1 (ad)←(ad)–1 ((Ri))←((Ri))–1
Умножение/деление аккумулятора на регистр *B* MUL AB DIV AB	1 1 4	 (B)(A)←(A)*(B) (B).(A)←(A)/(B), без знака

Логические команды	Т Б Ц	Операция
Логическое умножение		
ANL A, #d	2 2 1	(A)←(A) AND #d
ANL ad, #d	7 3 2	(ad)←(ad) AND #d
ANL A, Rn	1 1 1	(A)←(A) AND (Rn)
ANL A, ad	3 2 1	(A)←(A) AND (ad)
ANL ad, A	3 2 1	(ad)←(ad) AND (A)
ANL A, @Ri	1 1 1	(A)←(A) AND ((Ri))
Логическое сложение		
ORL A, #d	2 2 1	(A)←(A) OR #d
ORL ad, #d	7 3 2	(ad)←(ad) OR #d
ORL A, Rn	1 1 1	(A)←(A) OR (Rn)
ORL A, ad	3 2 1	(A)←(A) OR (ad)
ORL ad, A	3 2 1	(ad)←(ad) OR (A)
ORL A, @Ri	1 1 1	(A)←(A) OR ((Ri))
Логическое исключающее сложение		
XRL A, #d	2 2 1	(A)←(A) XOR #d
XRL ad, #d	7 3 2	(ad)←(ad) XOR #d
XRL A, Rn	1 1 1	(A)←(A) XOR (Rn)
XRL A, ad	3 2 1	(A)←(A) XOR (ad)
XRL ad, A	3 2 1	(ad)←(ad) XOR (A)
XRL A, @Ri	1 1 1	(A)←(A) XOR ((Ri))
Сброс аккумулятора CLR A	1 1 1	(A)←0
Инверсия аккумулятора CPL A	1 1 1	(A)←NOT(A)
Сдвиг аккумулятора влево циклический	1 1 1	
RL A		$(A_{n+1})\leftarrow(A_n)$, n=0÷6, $(A_0)\leftarrow(A_7)$
RLC A		$(A_{n+1})\leftarrow(A_n)$, n=0÷6, $(A_0)\leftarrow$(C), (C)$\leftarrow(A_7)$
Сдвиг аккумулятора вправо циклический	1 1 1	
RR A		$(A_n)\leftarrow(A_{n+1})$, n=0÷6, $(A_7)\leftarrow(A_0)$
RRC A		$(A_n)\leftarrow(A_{n+1})$, n=0÷6, $(A_7)\leftarrow$(C), (C)$\leftarrow(A_0)$
Обмен местами тетрад в аккумуляторе	1 1 1	
SWAP A		$(A_{0\ldots3})\leftrightarrow(A_{4\ldots7})$

Команды операций с битами	Т Б Ц	Операция
Операции с битом переноса	1 1 1	
CLR C		(C) ← 0
SETB C		(C) ← 1
CPL C		(C) ← NOT(C)
Сброс/установка бита с номером bit	4 2 1	
CLR bit		(b) ← 0
SETB bit		(b) ← 1
CPL bit		(b) ← NOT(b)
Логическое умножение переноса и бита	4 2 2	
ANL C, bit		(C) ← (C) AND (b)
ANL C, /bit		(C) ← (C) AND (NOT(b))
Логическое сложение переноса и бита	4 2 2	
ORL C, bit		(C) ← (C) OR (b)
ORL C, /bit		(C) ← (C) OR (NOT(b))
Пересылка бита		
MOV C, bit	4 2 1	(C) ← (b)
MOV bit, C	4 2 2	(b) ← (C)

Команды передачи управления	Т Б Ц	Операция
Длинный переход LJMP ad16	12 3 2	(PC)←ad16
Абсолютный переход в границах 2 кБ AJMP ad11	6 2 2	(PC)←(PC)+2, (PC_{0-10})←ad11
Короткий относит. переход в границах 256 байт SJMP rel	5 2 2	(PC)←(PC)+2, (PC)←(PC)+rel
Косвенный относительный переход JMP @A+DPTR	1 1 2	(PC)←(A)+(DPTR)
Переход, если аккумулятор =0, ≠0 JZ rel JNZ rel	5 2 2	(PC)←(PC)+2, если (A)=0,то (PC)←(PC)+rel (PC)←(PC)+2, если (A)≠0,то (PC)←(PC)+rel
Переход, если перенос =1, ≠1 JC rel JNC rel	5 2 2	(PC)←(PC)+2, если (C)=1, то (PC)←(PC)+rel (PC)←(PC)+2, если (C)=0, то (PC)←(PC)+rel
Переход, если бит =1, =0 JB bit, rel JNB bit, rel	11 3 2	(PC)←(PC)+3, если (b)=l, то (PC)←(PC)+rel (PC)←(PC)+3, если (b)=0, то (PC)←(PC)+rel
Переход, если бит установлен, с последующим сбросом бита JBC bit, rel	11 3 2	(PC)←(PC)+3, если (bit)=1, то (bit)←0 и (PC)←(PC)+rel
Декремент операнда и переход, если ≠0 DJNZ Rn, rel DJNZ ad, rel	8 2 2 8 3 2	(PC)←(PC) +2 (3), (op)←(op)–1, если (op) ≠ 0, то (PC)←(PC)+rel
Сравнение операнда1 с операндом2 и переход, если не равно CJNE A, ad, rel CJNE A, #d, rel CJNE @Ri, #d, rel	8 3 2 10 3 2 10 3 2	(PC)←(PC)+3, если (op1)≠(op2), то (PC)←(PC)+rel, если (op1)<(op2), то (C)←1, иначе (C)←0
Длинный вызов подпрограммы LCALL adl6	12 3 2	(PC)←(PC) + 3, (SP)←(SP)+1, ((SP))←($PC_{0...7}$), (SP)←(SP)+1, ((SP))←($PC_{8...15}$), (PC)←ad16
Абсолютный вызов подпрограммы в пределах 2 кБ ACALL ad11	6 2 2	(PC)←(PC)+2, (SP)←(SP)+1, ((SP))←($PC_{0...7}$), (SP)←(SP)+1, ((SP))←($PC_{8...15}$), (PC_{0-10})←ad11
Возврат из подпрограммы RET	1 1 2	($PC_{8...15}$)←((SP)), (SP)←(SP)–1, ($PC_{0...7}$) ←((SP)), (SP)←(SP)–1
Возврат из прерывания RETI	1 1 2	($PC_{8...15}$)←((SP)), (SP)←(SP)–1, ($PC_{0...7}$)←((SP)), (SP)←(SP)–1
Пустая операция NOP	1 1 1	(PC)←(PC)+1

Примечание. Ассемблер допускает использование обобщенного имени команд *JMP* и *CALL*, которые в процессе трансляции заменяются оптимальными по формату командами перехода (*AJMP, SJMP, LJMP*) или вызова (*ACALL, LCALL*).

Команды, влияющие на флаги результата представлены в нижеследующей таблице.

Мнемоника	Флаги
ADD A, <байт источника> ADDC A, <байт источника> SUBB A, <байт источника>	AC, C, OV
CLR C CPL C SETB C MOV C, <бит источника> ANL C, <бит источника> ANL C, </бит источника> ORL C, <бит источника> ORL C, </бит источника> RLC A RRC A	C
CJNE <байт назначения>, <байт источника>, <смещение>	C
DA A	AC, C
MUL AB DIV AB	C=0, OV

2. Общие сведения о МК *Motorola* 68*HC*05

Семейство МК 68*HC*xx имеют принстонскую архитектуру (Фон-Неймана), а ядром МК является *CISC*-процессор. Существует более 180 моделей 65*HC*05, специализированных для различных областей применения.

На рис.2.1 представлена упрощенная структурная схема МК.

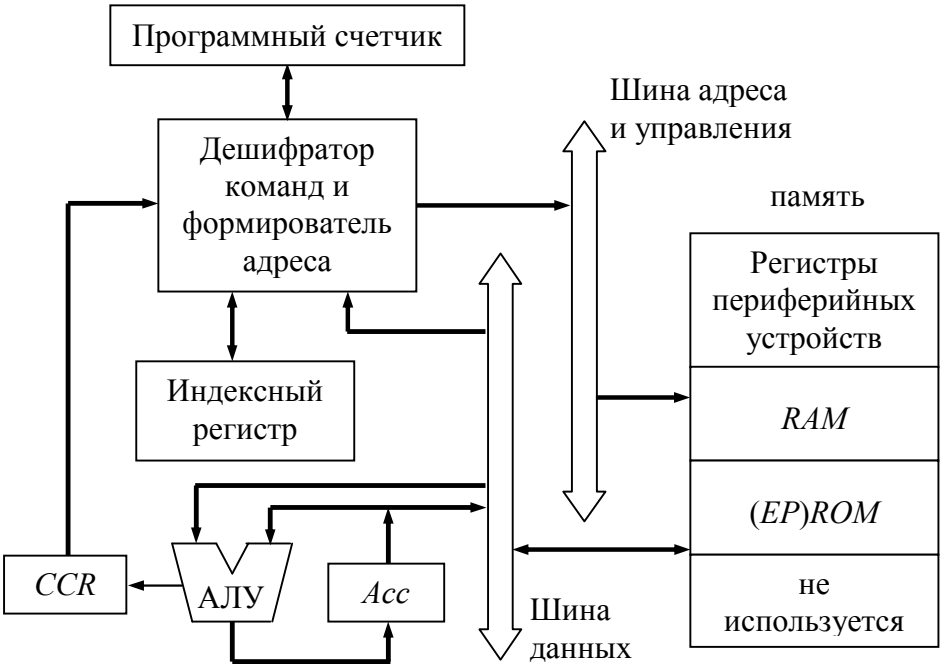

Рис. 2.1 – Структурная схема МК семейства 68*HC*05

CCR – регистр условий. На рис.2.2 представлена его структура и назначение битов.

Бит	7	6	5	4	3	2	1	0
Флаг	1	1	1	*A*	*I*	*N*	*Z*	*C*

Перенос между тетрадами – устанавливается при переносе (заеме) в (из) старшую тетраду при сложении (вычитании)

Маскирование прерываний – установка данного бита разрешает обслуживание прерываний

Перенос/заем – устанавливается при выходе результата за границы 8-разрядной сетки

Нулевой результат – устанавливается при нулевом результате операции

Отрицательный результат – устанавливается при отрицательном результате

Рис. 2.2 – Регистр условий (*CCR*), состав и назначение битов

После включения питания МК прежде чем начать работу вводит задержку длительностью 4064 такта, необходимую для стабилизации работы генератора и окончания переходных процессов, вызванных включением питания. После этого, а также после снятия активного сигнала *Reset* процессор загружает вектор начального состояния – адрес первой выполняемой команды, расположенной в конце адресного пространства *EPROM*.

Большинство МК семейства 86*HC*05 содержит в своем составе масочное ПЗУ, в котором имеется несколько программ. Эти программы предназначены для программирования МК через последовательный или параллельный интерфейс. Активация режима программирования происходит путем подачи на входы МК определенной комбинации сигналов при включении МК.

Особенностями МК данного семейства является:

1. Указатель стека в этом МК является 6-разрядным индексным регистром, значение которого всегда используется со смещением $0C0 (два старших бита всегда установлены). Начальным состоянием указателя стека является $0FF, т.е. стек при помещении в него слов растет вниз. При вызове подпрограммы содержимое программного счетчика сохраняется в стеке (первым сохраняется младший байт).
2. Реализация прерываний похожа на процесс начального запуска. При поступлении запроса на прерывание при разрешенных прерываниях МК выбирает из конца *EPROM* адрес обработчика прерывания. При этом в стеке сохраняется контекст регистров в следующем порядке: старший байт программного счетчика, младший байт программного счетчика, индексный регистр, аккумулятор, регистр условий. Эта процедура занимает 10 машинных тактов с учетом загрузки вектора прерывания.
3. Доступ к периферийным регистрам, которые занимают первые 32 байта в адресном пространстве, осуществляется непосредственно путем записи/чтения по соответствующим адресам.
4. Программный код может быть загружен с помощью специальной программы хранящейся в ПЗУ в ОЗУ, упрощая тем самым процесс отладки программы.

2.1 СПОСОБЫ АДРЕСАЦИИ ДАННЫХ

МК семейства 68*HC*xx при хранении данных в памяти существенно отличаются от других МК, которые при сохранении 16-разрядного числа в памяти используют правило: младший байт по меньшему адресу. В МК *Motorola* все наоборот.

В МК 68*HC*05 реализуются следующие виды адресации:

- Регистровая (регистр, с которым работает команда, содержится в ее теле): INCA – инкремент аккумулятора.
- Непосредственная (операнд содержится в команде): ADD #2 – добавить 2 к содержимому аккумулятора.
- Прямая (в команде содержится адрес операнда): LDA $37 – загрузить в аккумулятор содержимое ячейки памяти с адресом 37h (адресует память в первых 256 байтах).
- Расширенная (16-разрядный адрес ячейки памяти содержится в команде): goto $1234 – загрузить в программный счетчик значение 1234h.
- Индексная: 8-разрядный адрес операнда находится в индексном регистре
- Индексная со смещением (8-разрядное смещение содержится в команде): адрес операнда вычисляется суммированием смещения и значением индексного регистра.
- Индексная с 16-разрядным смещением: значение индексного регистра суммируется с 16-разрядным смещением, содержащимся в команде.

2.2 ПАРАЛЛЕЛЬНЫЙ ВВОД-ВЫВОД ДАННЫХ

Порты ввода-вывода этих МК мало отличаются от портов других МК. На рис.2.3 представлена структура порта ввода-вывода МК данного семейства.

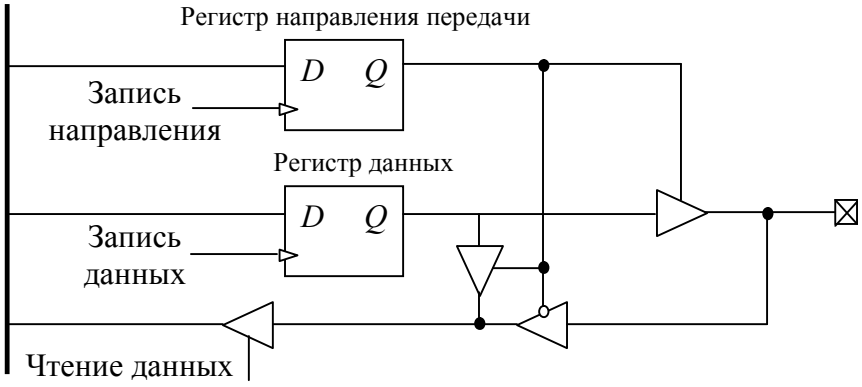

Рис. 2.3 – Структурная схема порта ввода-вывода семейства МК 68*HC*05

2.3 ПРОГРАММИРУЕМЫЙ ТАЙМЕР

Таймеры МК этого семейства существенно отличаются от таймеров других МК. Вместо того чтобы формировать временные интервалы, заданные пользователем, таймеры 68*HC*05 работают непрерывно, позволяя программе следить за их состоянием. Кроме того, структура таймера изменяется в пределах семейства. Рассмотрим два таймера МК простого: 68*HC*05*J*1*A* и более сложного: 68*HC*05*C*8.

Таймер МК 68*HC*05*J*1*A* работает по внутреннему тактовому сигналу или от внешнего синхросигнала деленного на два. При переполнении таймера может быть вызвано прерывание. При работе с таймером надо учитывать особенности:

1) для реализации задержки необходимо производить циклический опрос таймера;
2) сигнал переполнения счетчика таймера заведен на линейку делителей;
3) сигнал с последней ступени линейки делителей служит синхроимпульсами для сторожевого таймера (*COP*).

На рис.2.4 показана структура таймера МК 68*HC*05*J*1*A*.

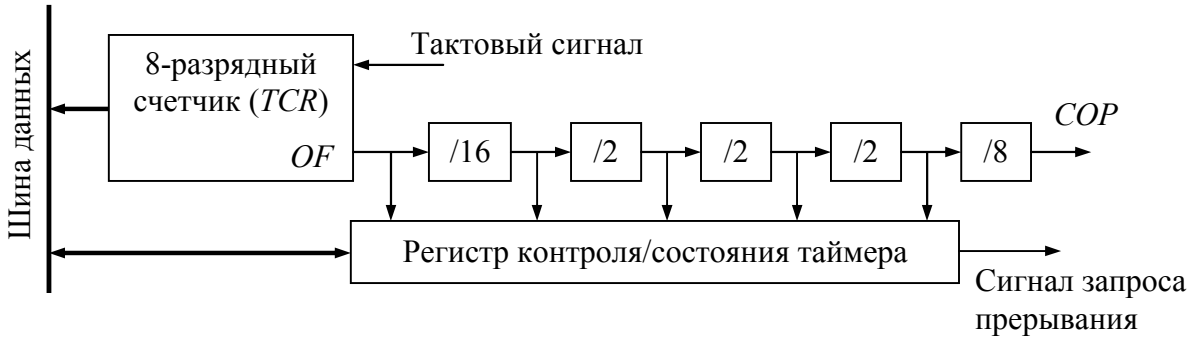

Рис. 2.4 – Структурная схема таймера МК 68*HC*05*J*1*A*

На рис.2.5 показана структура таймера МК 68*HC*05*C*8. Такая конфигурация более гибкая, но и более сложная. Здесь таймер может реагировать на различный входной сигнал (по уровням или перепадам), есть возможность аппаратного отсчета временного интервала, доступны режимы: сравнения и захвата.

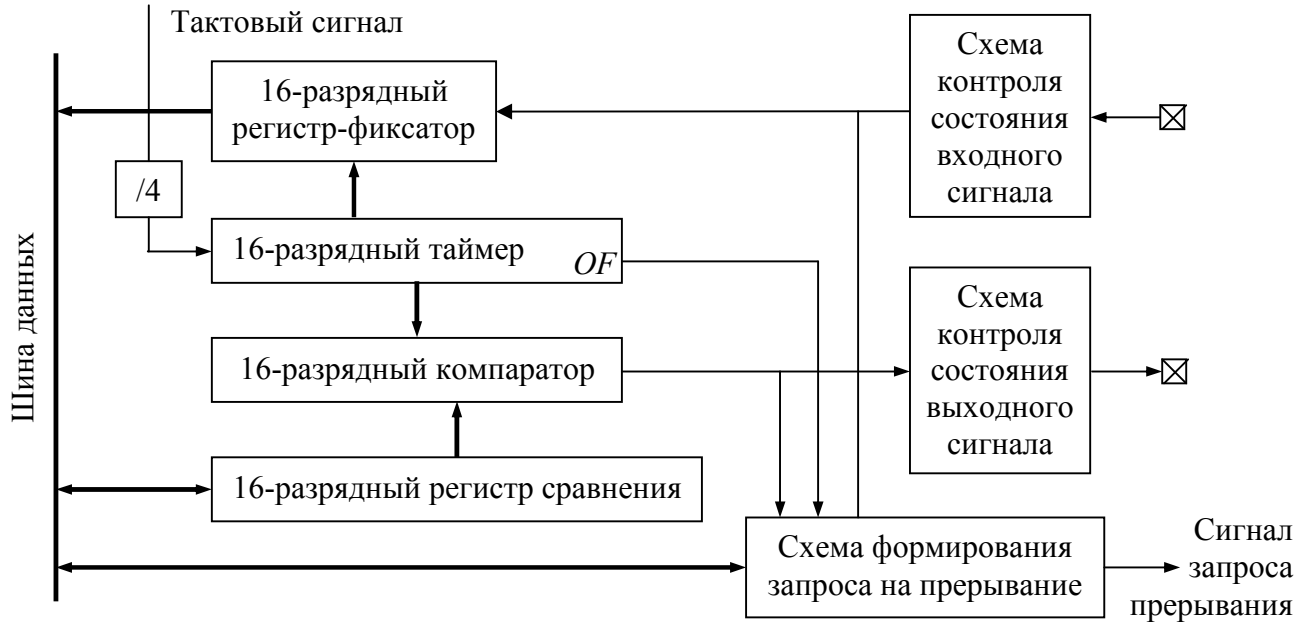

Рис. 2.5 – Структура таймера МК 68*HC*05*C*8

2.4 ПОСЛЕДОВАТЕЛЬНЫЙ ВВОД-ВЫВОД

Как и в других МК последовательный порт этого семейства МК может работать в синхронном и асинхронном режимах.

Последовательный порт асинхронного приемопередатчика является полным дуплексным портом. Этот режим (*SCI*) обеспечивает обмен данными в формате *RS*-232. На рис.2.6 представлены структурные схемы передатчика, приемника и генератора синхросигналов модуля *SCI* в 68*HC*05.

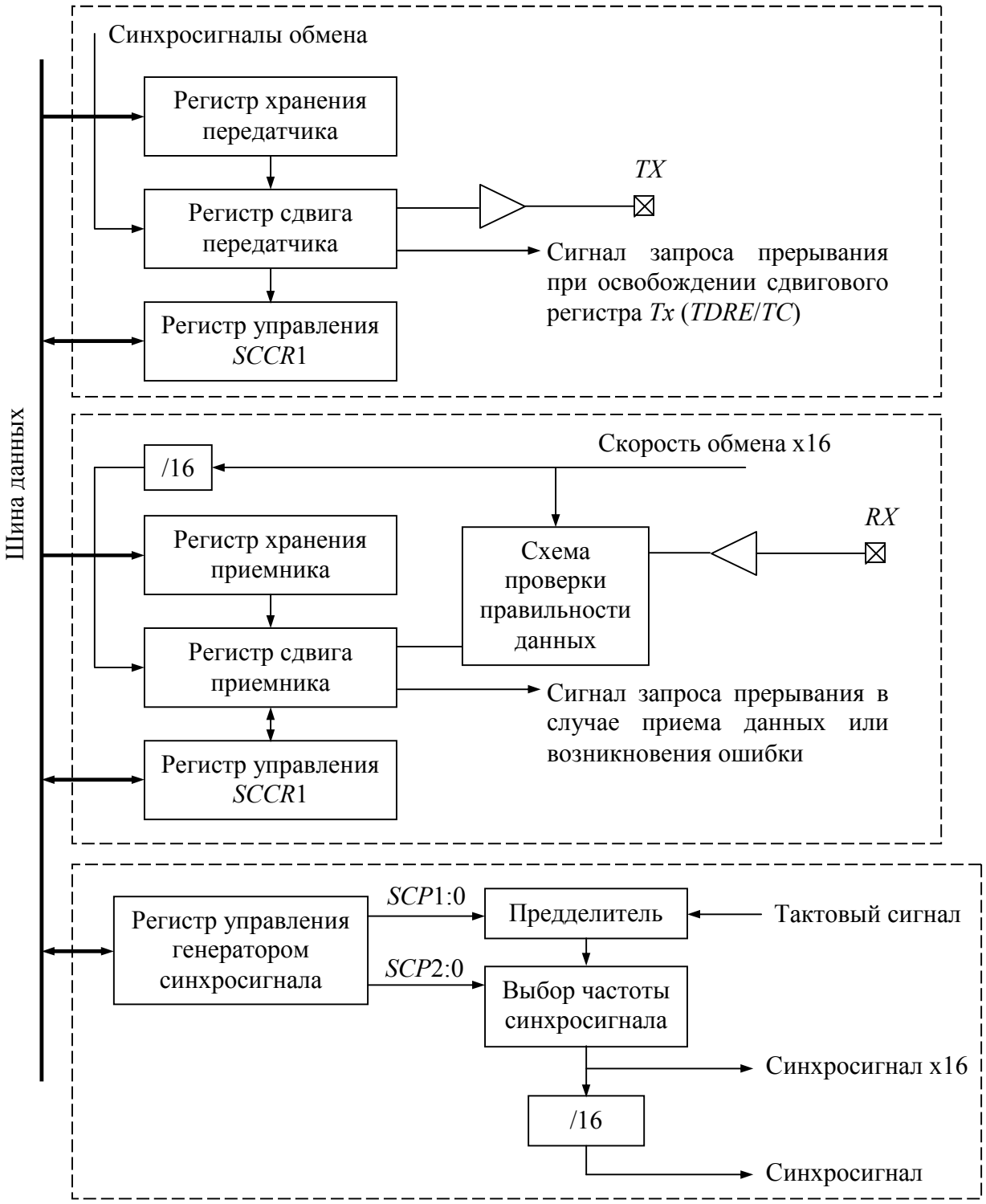

Рис. 2.6 – Структурные схемы передатчика, приемника и генератора синхросигналов модуля *SCI* в 68*HC*05 (асинхронный режим)

Синхронный модуль последовательного интерфейса (*SPI*) разрабатывался для взаимодействия с преобразователями последовательного кода в параллельный. Поэтому синхронный режим последовательного порта отличается от аналогичного режима других МК. Структурная схема последовательного порта в синхронном режиме представлена на рис.2.7. Передача данных инициируется записью байта в сдвиговый регистр и происходит по переднему фронту синхроимпульса. При этом первым передается старший бит.

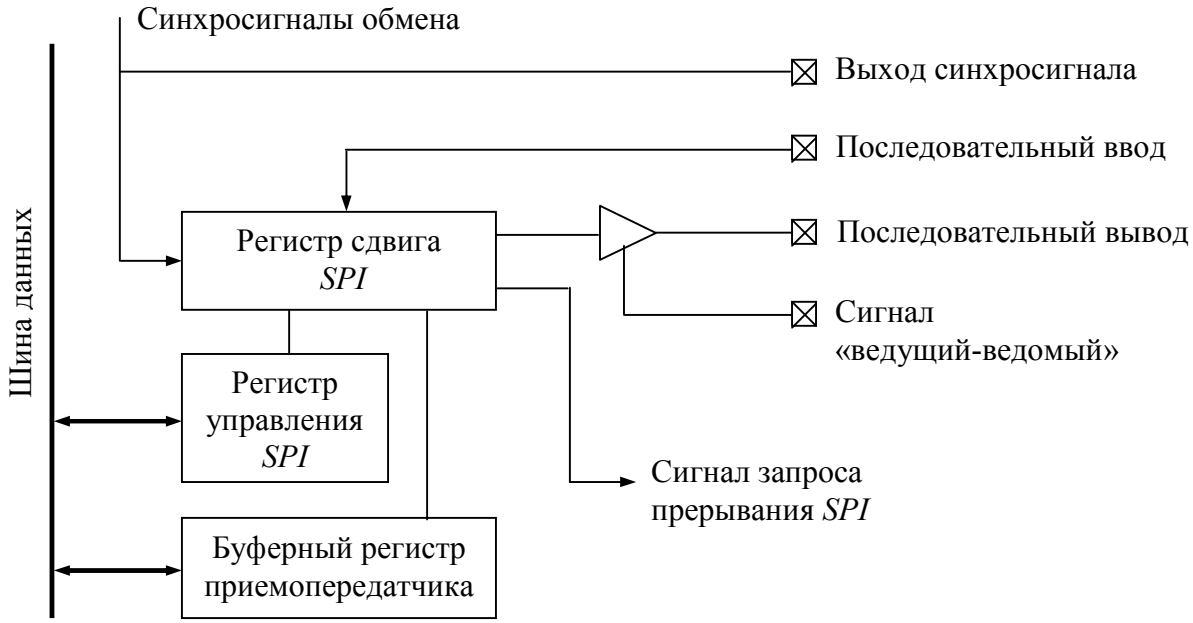

Рис. 2.7 – Структурная схема модуля *SPI* в МК 68*HC*05

3. *AVR* микроконтроллеры.

Компания *Atmel* на сегодняшний день, как и другие компании, производят широкую гамму микроконтроллеров, подходящих для решения различных задач. При этом, например серия МК *AT*89*Sxxxx* повторяет по возможностям и системе команд серию МК *Intel* 87*C*51. Также *Atmel* производит ряд собственных МК, основные характеристики некоторых из них приведены в табл.3.1.

Таблица 3.1 – Некоторые МК семейств 8 бит

МК	*Flash КБ*	*SRAM* (*EEPROM*) байт	F_{max} (МГц)	Макс *I/O pin*	Прерывания (внеш)	*SPI*	АЦП каналов	*UART*	ШИМ каналов	*Vcc* (В)	Корпус
AT90S1200	1	0 (64)	12	15	3 (1)					2.7-6.0	DIP20, SO20, SSOP20
AT90S2313	2	128 (128)	10	15	10 (2)			1	1	2.7-6.0	DIP20, SO20
ATmega128	128	4096 (4096)	16	53	34 (8)	1	8	2	8	4.5 - 5.5	QFP64
ATmega162	16	1024 (512)	16	35	28 (3)	1		2	4	4.5 - 5.5	DIP40, QFP44
ATmega168	16	1024 (512)	20	23	26 (26)	1	8	*USART*	3	1.8 - 5.5	DIP28, QFP32, MLF32
ATmega32	32	2048 (1024)	16	32	19 (3)	1	8	1	4	4.0 - 5.5	DIP40, QFP44, MLF44
ATmega324P	32	2048 (1024)	20	32	31 (3)	1	8	1	3	4.5 - 5.5	DIP40, QFP44 MLF44
ATmega48	4	512 (256)	24	23	26 (26)	1	8	*USART*	3	1.8 - 5.5	DIP28, QFP32, MLF32
ATmega64	64	4096 (2048)	16	53	34 (8)	1	8	2	8	4.5 - 5.5	QFP64
ATmega644P	64	4096 (2048)	20	32	31 (3)	1	8	1	3	4.5 - 5.5	DIP40, QFP44 MLF44
ATmega8	8	1024 (512)	16	23	18 (2)	1	8	1	3	4.5 - 5.5	DIP28, QFP32, MLF32
ATtiny11	1		6	6	4 (1)					4 - 5.5	DIP8, SO8
ATtiny12	1	0 (64)	8	6	5 (1)					4 - 5.5	DIP8, SO8
ATtiny2313	2	128 (128)	20	18	10 (2)	*USI*		*USART*	4	1.8 - 5.5	DIP20, SO20
ATtiny26	2	128 (128)	16	16	11 (1)	*USI*	11		2	4.5 - 5.5	DIP20, SO20, MLF32

- *AT*90*S*1200, *AT*90*S*2313, *ATtiny*11 – не имеют детектора питания
- *AT*90*S*1200, *AT*90*S*2313, все *ATtiny* – не имеют аппаратного умножителя
- Разрядность АЦП – 10 бит.
- *USART* (*Universal Synchronous Asynchronous Receiver-Transmitter*) отличается от *UART* (*Universal Asynchronous Receiver-Transmitter*) более высокой максимальной скоростью (до 4 Мбит/сек) и большими возможностями.
- Интерфейс *USI* присутствует во многих МК и представляет собой заготовку из сдвигового регистра, 4х-битного счетчика и набора регистров инициализации с прерываниями. Может работать как в 3х-проводном режиме (как *SPI*) так и в 2х-проводном режиме (*TWI* или *I2C*), также у него есть нестандартные применения для реализации *UART*.

Необходимо отметить, что все МК одного семейства совместимы по исходным кодам, обеспечены программным обеспечением, а также аппаратными средствами, позволяющими отлаживать приложения с учетом реального масштаба времени. МК семейства *AVR* делятся на три группы (подсемейства): *Classic*, *Tiny* и *Mega*. МК семейства *Classic* (*AT*90*S*xxxx) в

настоящее время уже не выпускаются, однако все «классические» *AVR* имеют полные аналоги в семействах *Tiny* и *Mega*.

Архитектура всех МК является гарвардской с элементами принстонской архитектуры, что обеспечивает эффективность МК при решении широкого круга задач. Так архитектура включает в себя 32 регистра общего назначения (РОН), выполняют большинство команд за один цикл – свойственно *RISC* системам; память программ и данных разделены – свойственно гарвардской архитектуре; имеются предопределенные регистры – свойственно принстонской архитектуре; имеются сложные команды, требующие нескольких циклов – свойственно *CISC* системам и т.п. В общем, если попытаться сформулировать отличительные черты МК этих серий, то получится примерно следующее:

1. Производительность порядка 1 *MIPS*/МГц, вычислительное ядро *AVR* на ряде задач по производительности превосходит 16-разрядный процессор *i*80286.
2. Усовершенствованная *RISC* архитектура, концепция которой предполагает наличие набора команд, состоящего из минимума компактных и быстро выполняющихся инструкций. Это упрощает устройство ядра и ускоряет его работу: типовая инструкция выполняется за один такт (кроме команд ветвления программы, обращения к памяти и некоторых других, оперирующих с данными большей длины). В *AVR* имеется двухступенчатый конвейер, когда в одном такте совмещаются операции выполнения одной команды и загрузки другой.
3. Раздельные шины памяти команд и данных. Реализована гарвардская архитектура, т.е. данные и команды могут выбираться одновременно.
4. 32 регистра общего назначения (РОН). *Atmel* была первой компанией, ушедшей далеко от классической модели вычислительного ядра, в которой выполнение команд предусматривает постоянный обмен данными между АЛУ и аккумулятором. Введение РОН в таком количестве в ряде случаев позволяет полностью отказаться от расположения локальных и глобальных переменных в ОЗУ и от использования стека. Правда, это привело к некоторому усложнению системы команд пересылок данных.
5. *Flash*-память программ (10 000 циклов стирание/запись) с возможностью внутрисистемного перепрограммирования и загрузки через последовательный канал прямо в готовой схеме.
6. Наличие отдельной области энергонезависимой памяти (*EEPROM*, 100 000 циклов стирание/запись) для хранения данных, с возможностью записи программным путем, или внешней загрузки через *SPI*-интерфейс.
7. Встроенные устройства для обработки аналоговых сигналов: аналоговый компаратор и многоканальный АЦП.
8. Сторожевой таймер, позволяющий осуществлять автоматическую перезагрузку контроллера через определенные промежутки времени.
9. Последовательные интерфейсы *SPI*, *TWI* (*I2C*) и *UART* (*USART*), позволяющие осуществлять обмен данными с большинством стандартных датчиков и других внешних устройств.
10. Таймеры/счетчики с предустановкой и возможностью выбора источника счетных импульсов: как правило, один или несколько 8-разрядных и как минимум один 16-разрядный, в т.ч. способные работать в режиме широтно-импульсной модуляции (ШИМ, англ.: *PWM*).
11. Возможность работы при тактовой частоте от 0 Гц до 20 МГц.
12. Диапазон напряжений питаний от 2.7 до 5.5 В (в некоторых случаях от 1.8 до 6.0В).
13. Низкое потребление и многочисленные режимы энергосбережения, различающиеся числом узлов, остающимися включенными. Выход из «спящих» режимов осуществляется либо по сторожевому таймеру, либо по внешним прерываниям.
14. Встроенный монитор питания – детектор падения напряжения.

Из широкой номенклатуры данного семейства МК рассмотрим *AT*90*S*2313:

- 120 команд, большинство которых выполняются за один цикл;
- 2 Кбайт *Flash* ПЗУ программ с возможностью внутрисистемного перепрограммирования (1000 циклов стирания/записи) и загрузки через *SPI*;
- 128 байт электрически стираемого и программируемого ПЗУ (10000 циклов стирания/ записи) с возможностью внутрисистемной загрузки через *SPI*;
- 15 программируемых линий ввода/вывода;
- полностью статический прибор (работает от 0 до 10 МГц);
- напряжение питания от 2.7 В до 6 В;
- 8-разрядный и 16-разрядный с режимами сравнения и захвата таймеры/счетчики с общим прескаляром;
- функция ШИМ с 8, 9, 10 битным разрешением;
- полный дуплексный *UART*;
- два внешних и восемь внутренних источника сигналов прерывания;
- программируемый сторожевой таймер с собственным генератором;
- встроенный аналоговый компаратор;
- режимы энергосбережения: пассивный (*idle*) и стоповый (*power down*);
- возможность работы без внешних компонентов (встроенный *RC* генератор);
- 20-контактный корпус - рис. 3.1.

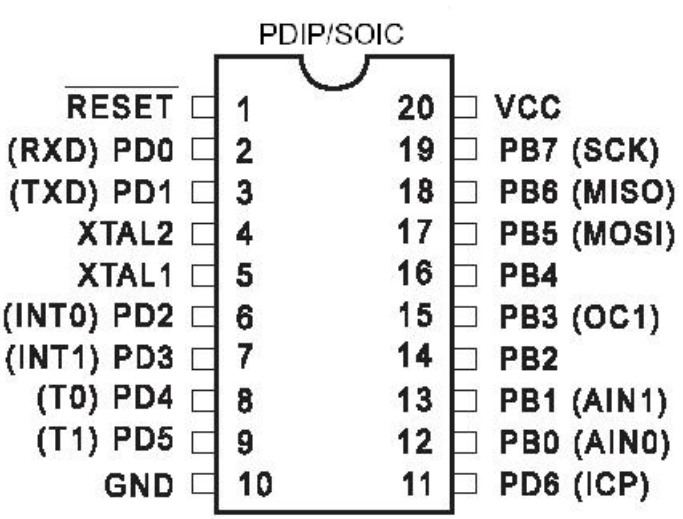

Рис. 3.1 – Цоколевка *AT*90*S*2313

3.1 АРХИТЕКТУРА

На рис.3.2 представлена архитектура МК *AT*90*S*2313.

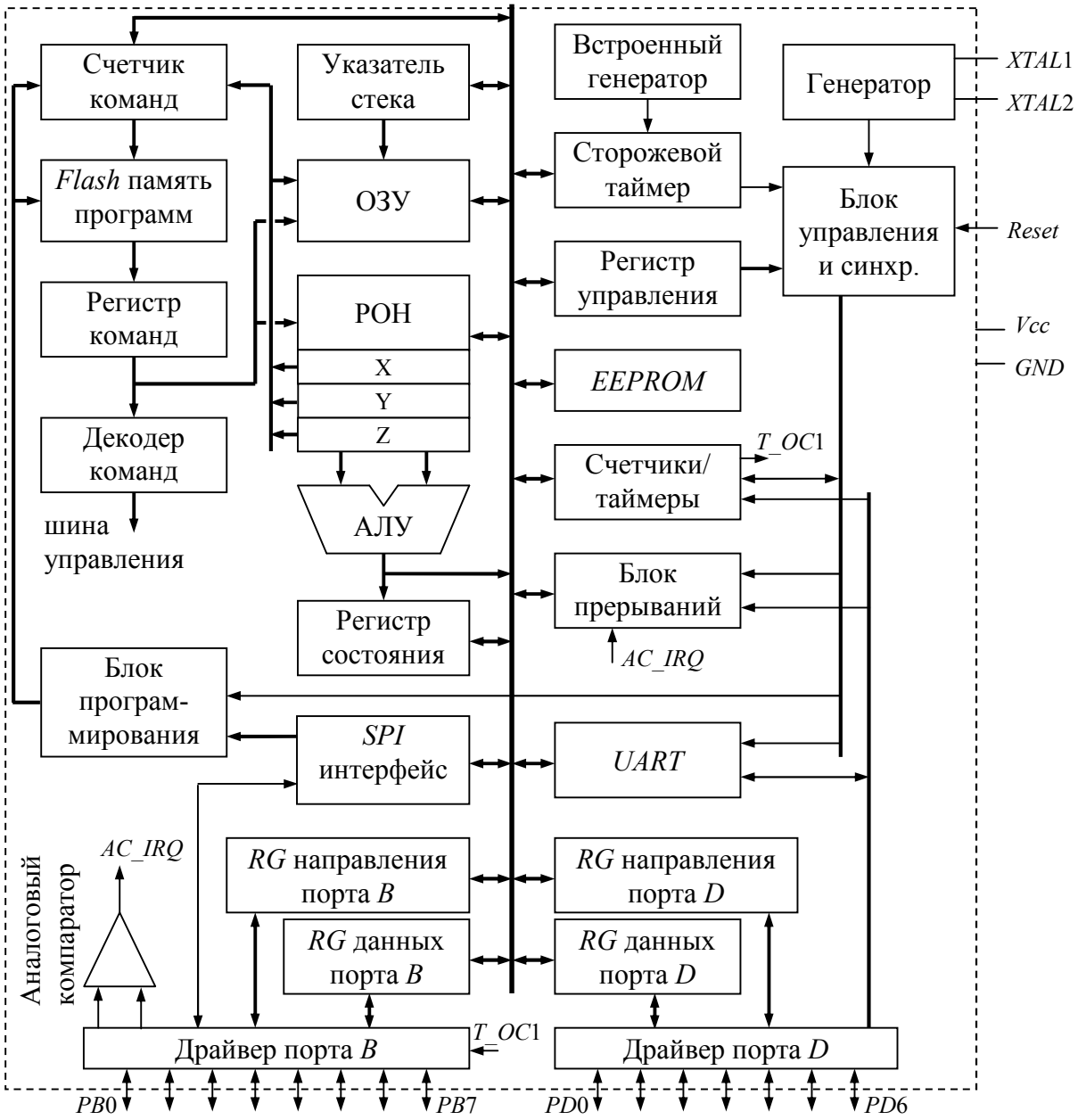

Рис. 3.2 – Блок схема МК *AT*90*S*2313 и расположение выводов

Можно видеть, что использование сразу всех блоков невозможно (например: компаратор и *SPI* разделяют выводы порта *B*, число линий порта *D* – 7 (не 8) и они разделяются с *UART* и таймерами, внешними прерываниями и т.п.). Такая ситуация свойственна всем МК.

Прежде чем начать рассмотрение блоков структурной схемы и соответственно возможностей и попутно системы команд рассмотрим распределение памяти МК и систему адресации. Существует два способа обращения (адресации) к ресурсам МК: регистрам (РОН), портам ввода-вывода и памяти. Первый способ – это прямое обращение к требуемой области. Второй – обращение к областям как к единому пространству адресов. На рис.3.3 представлено распределение памяти МК. Соответственно для обращения к каждой области памяти существуют соответствующие команды:

- к регистрам ввода-вывода: IN/OUT – для обмена данными между РОН и портами;
- к памяти: LD – загрузить из памяти (*load*) и ST – записать в память (*store*) – используются для пересылки данных между ОЗУ и РОН, между ПЗУ и РОН (только чтение);
- к РОН: MOV – для пересылки данных между РОНами.

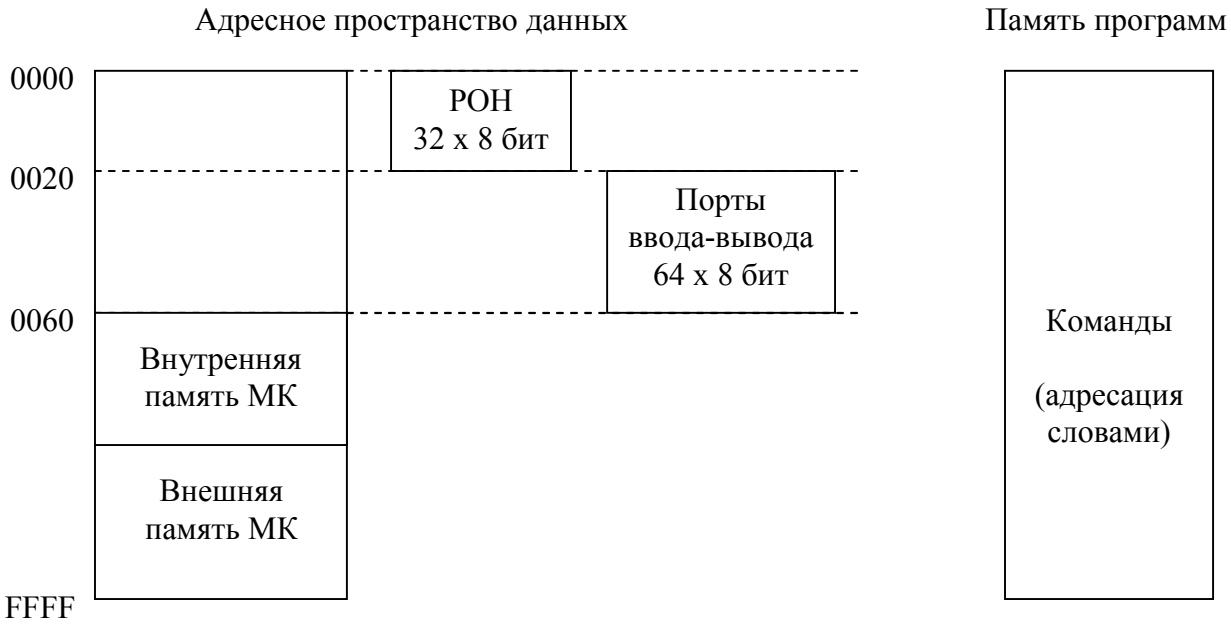

Рис. 3.3 – Распределение памяти семейства МК *AVR AT*90*S*xxxx

3.1.1 СПОСОБЫ АДРЕСАЦИИ ДАННЫХ

В МК *AVR* по признаку адресации можно все команды можно разбить на две группы: непосредственная адресация и индексная адресация. К первой группе относятся команды, работающие с РОН и непосредственными операндами, указанными в самой команде – рис.3.4 – для команд с одним, двумя операндами и с непосредственным операндом. Ко второй группе команд относятся команды с косвенной адресацией операнда, при этом адрес формируется согласно структуре – рис.3.5 (возможна автоматическая инкрементация/декрементация индексных регистров до выполнения команды или после нее).

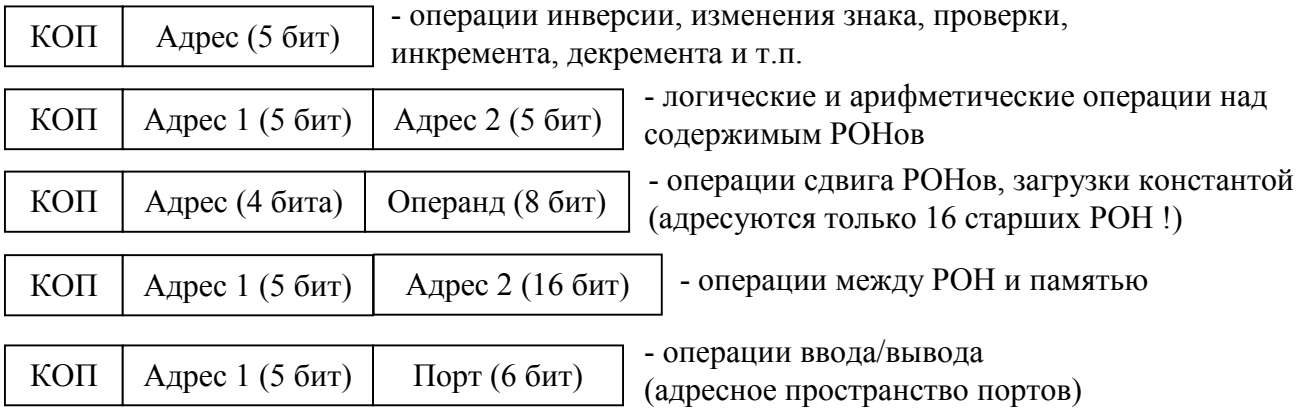

Рис. 3.4 – Структура команд с непосредственной адресацией операндов

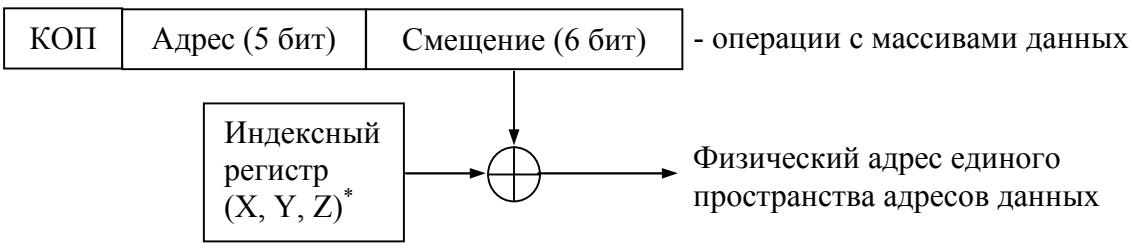

Рис. 3.5 – Формирование адреса операнда команд с косвенной адресацией

3.1.2 РОНы и АЛУ

АЛУ выполняет 91 из 120 команд МК. Все 32 регистра подключены к АЛУ. Последние шесть 8-разрядных регистров попарно объединены и образуют три 16-разрядных индексных регистра (*X*, *Y*, *Z*). При этом индексный регистр *Z* может применяться для чтения данных из памяти программ (младший бит адреса указывает читаемый байт в 16-разрядном слове из ПЗУ).

АЛУ выполняет три категории команд: арифметические, логические и битовые, результат операций отражается на флагах регистра состояния. На рис.3.6 представлен состав и назначение бит регистра состояния *SREG*.

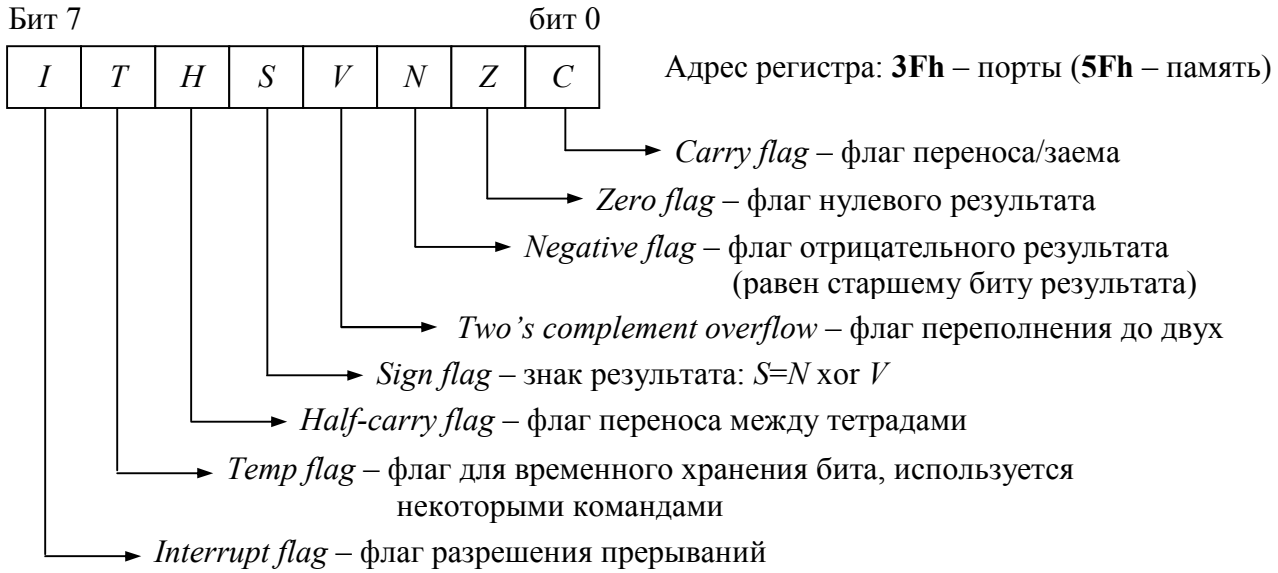

Рис. 3.6 – Регистр состояния МК (*SREG*)

3.1.3 ПРЕРЫВАНИЯ

Данный МК поддерживает 10 прерываний. Каждому прерыванию присвоен вектор прерывания, которые расположены в начале памяти программ. Каждый вектор прерывания содержит команду относительного перехода на заданный адрес обработчика прерывания (*IRQ handler*) – 2 байта (1 слово). Назначение векторов прерываний следующее:

Адрес	Вектор	Назначение
0000	1	– вектор сброса (*Reset*)
0001	2	– внешний запрос (*INT*0)
0002	3	– внешний запрос (*INT*1)
0003	4	– таймер 1, событие захвата (*Timer Capt* 1)
0004	5	– таймер 1, событие совпадения (*Timer Comp* 1)
0005	6	– таймер 1, переполнение (*Timer Ovf* 1)
0006	7	– таймер 0, переполнение (*Timer Ovf* 0)
0007	8	– *UART*, прием завершен (*UART RX*)
0008	9	– *UART*, регистр данных пуст (*UART UDRE*)
0009	10	– *UART*, передача завершена (*UART TX*)
000A	11	– Аналоговый компаратор (*ANA_COMP*)

Следовательно, любая программа, использующая прерывания, должна начинаться с адреса выше 000Bh (адресация в ПЗУ словами).

Особенности обработки внешних прерываний:

- активным сигналом может быть *rise, falling, low level* – см. *MCU Control RG* (биты *ISC*);
- входные сигналы *IRQ* фиксируются триггерами (флаги) – *GIFR* (*General Interrupt Flag RG*, биты *INTF*1, *INTF*0), лог. "1" в битах говорит о наличии прерывания

	7	6	5	4	3	2	1	0
GIFR $3A ($5A)	*INTF*1	*INTF*0	-	-	-	-	-	-
	R/W	*R/W*	*R*	*R*	*R*	*R*	*R*	*R*
Начал. знач.	0	0	0	0	0	0	0	0

- есть регистр маски внешних прерываний *GIMSK* (*General Interrupt Mask RG*), лог. "1" в битах этого регистра разрешает прерывания

	7	6	5	4	3	2	1	0
GIMSK $3B ($5B)	*INT*1	*INT*0	-	-	-	-	-	-
	R/W	*R/W*	*R*	*R*	*R*	*R*	*R*	*R*
Начал. знач.	0	0	0	0	0	0	0	0

- если прерывание разрешено битами *GIMSK* и флагом *I* (*SREG*), то запрос по входам *INT* может быть выполнен, при этом соответствующие биты *GIFR* автоматически сбрасываются при выполнении обработчика прерываний (для программной очистки этих битов необходима запись «1»);
- фиксация активного сигнала по входам *INT* происходит всегда, даже если соответствующие выводы сконфигурированы как выход (порт), это позволяет эмулировать программное прерывание.

По аналогии обрабатывается прерывания от таймера, отличия состоят в других наименованиях регистров:

- входные сигналы фиксируются триггерами – *TIFR* (*Timer/Counter Interrupt Flag*)

	7	6	5	4	3	2	1	0
TIFR $38 ($58)	*TOV*1	*OCF*1*A*	-	-	*ICF*1	-	*TOV*0	-
	R/W	*R/W*	*R*	*R*	*R/W*	*R*	*R/W*	*R*
Начал. знач.	0	0	0	0	0	0	0	0

TOV1 – (*Timer/Counter Overflow Flag*) устанавливается в «1» когда таймер 1 переполняется, бит сбрасывается аппаратно когда выполняется соответствующее прерывание (для программной очистки бита надо записать во флаг «1»);

OCF1A – (*Output Compare Flag* 1*A*) устанавливается в «1» когда значение таймера 1 совпадает со значением регистра *OCR*1 (*Output Compare RG*), бит сбрасывается аппаратно когда выполняется соответствующее прерывание (для программной очистки бита надо записать во флаг «1»);

ICF1 – (*Input Capture Flag*) устанавливается в «1» когда происходит событие захвата, т.е. показывает, что текущее значение таймера 1 скопировано в регистр *ICR*1 (*Input Capture RG*), бит сбрасывается аппаратно когда выполняется соответствующее прерывание (для программной очистки бита надо записать во флаг «1»);

TOV0 – (*Timer/Counter Overflow Flag*) устанавливается в «1» когда таймер 0 переполняется, бит сбрасывается аппаратно когда выполняется соответствующее прерывание (для программной очистки бита надо записать во флаг «1»);

- регистр маски прерываний таймеров *TIMSK* (*Timer/Counter Interrupt Mask RG*)

	7	6	5	4	3	2	1	0
TIMSK $39 ($59)	*TOIE*1	*OCIE*1*A*	-	-	*TICIE*1	-	*TOIE*0	-
	R/W	*R/W*	*R*	*R*	*R/W*	*R*	*R/W*	*R*
Начал. знач.	0	0	0	0	0	0	0	0

TOIE1 – (*Timer/Counter*1 *Overflow Interrupt Enable*) установка бита в «1» разрешает прерывание таймера 1 по переполнению;

OCIE1A – (*Timer/Counter*1 *Output Compare Match Interrupt Enable*) установка бита в «1» разрешает прерывание таймера 1 по сравнению;

TICIE1 – (*Timer/Counter*1 *Input Capture Enable*) установка бита в «1» разрешает прерывание таймера 1 по захвату;

TOIE0 – (*Timer/Counter*0 *Overflow Interrupt Enable*) установка бита в «1» разрешает прерывание таймера 0 по переполнению;

- если прерывание разрешено битами *TIMSK* и флагом *I* (*SREG*), то запрос от таймера может быть обслужен.

Общий порядок обработки прерываний следующий:

1) фиксация сигнала прерывания входными схемами;
2) проверка масок прерываний;
3) выбор разрешенного прерывания с наивысшим приоритетом (по номеру: 0 – наивысший);
4) проверка флага *I* (*SREG*), если равен «1», то начало обработки прерывания;
5) сброс соответствующего бита во флагах фиксирующих сигнал прерывания (*TIFR*, *GIFR*);
6) сохранение в стеке значения программного счетчика (*PC*), уменьшение значения указателя стека на 2;
7) передача управления процедуре обработки прерывания и сброс бита *I* (*SREG*).

При возврате из прерывания:

1) из стека загружается значение *PC,* значение указателя вершины стека увеличивается на 2;
2) бит *I* (*SREG*) устанавливается в «1»;
3) передается управление прерванной программе.

Время отклика МК на прерывание определяется 4 тактами. Однако если прерывание пришло во время выполнения сложной инструкции, то запрос ожидает ее окончания. Возврат из прерывания также занимает 4 такта.

Управляющий регистр *MCUCR* (*MCU Control RG*)

	7	6	5	4	3	2	1	0
MCUCR \$35 (\$55)	-	-	*SE*	*SM*	*ISC*11	*ISC*10	*ISC*01	*ISC*00
	R	*R*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

SE – (*Sleep Enable*) установка в «1» разрешает спящий режим;

SM – (*Sleep Mode*) управляет типом спящего режима:

"0" – *idle* режим – остановка *CPU*, работа таймеров разрешена, *Watchdog* и система прерываний функционируют. Выход из режима осуществляется поступлением прерывания. Выполнение программы при этом начинается немедленно.

"1" – *power down* режим – все выключается, работают только *Watchdog*, внешние прерывания и схема сброса. Только эти виды прерываний могут «разбудить» МК.

ISC11, ISC10 – (*Interrupt Sense Control* 1) выбор типа активного сигнала по входу *INT*1:

00 – низкий уровень;
01 – зарезервировано;
10 – задний фронт сигнала (*falling edge*);
11 – передний фронт сигнала (*rise edge*).

ISC01, ISC00 – (*Interrupt Sense Control* 0) выбор типа активного сигнала по входу *INT*0:

00 – низкий уровень;
01 – зарезервировано;
10 – задний фронт сигнала (*falling edge*);
11 – передний фронт сигнала (*rise edge*).

Для входа в спящий режим (*Sleep mode*) необходимо определить его тип (бит *SM*) и разрешить спящий режим (бит *SE*), а затем инструкция *SLEEP* переведет МК в соответствующий режим с пониженным энергопотреблением. Если во время «сна» происходит прерывание МК «пробуждается», выполняет процедуру обработки прерывания и начинает выполнять программу с команды, следующей за командой *SLEEP*. Если во время «сна» произошел сброс МК, то МК «пробуждается» и начинает работу с начала программы по вектору *Reset*.

3.1.4 СБРОС МК

МК имеет три источника сигнала сброса:

1) при включении питания (после достижения напряжения уровня V_{POT} включается МК);
2) при поступлении сигнала сброса с внешнего вывода МК (длительность сигнала должна превышать 50 нс);
3) от *Watchdog*.

Во всех случаях выполнение программы начинается спустя некоторое время после сброса, что необходимо для окончания различных переходных процессов. Длительность этой задержки по умолчанию составляет 16 мс (16k циклов *Watchdog*). Однако можно программно уменьшить ее до 0.28 мс (256 циклов *Watchdog*), этим управляет *Fuse* бит *FSTRT* (при этом необходимо применение внешнего кварцевого резонатора). На рис.3.7 представлена структурная схема цепей сброса.

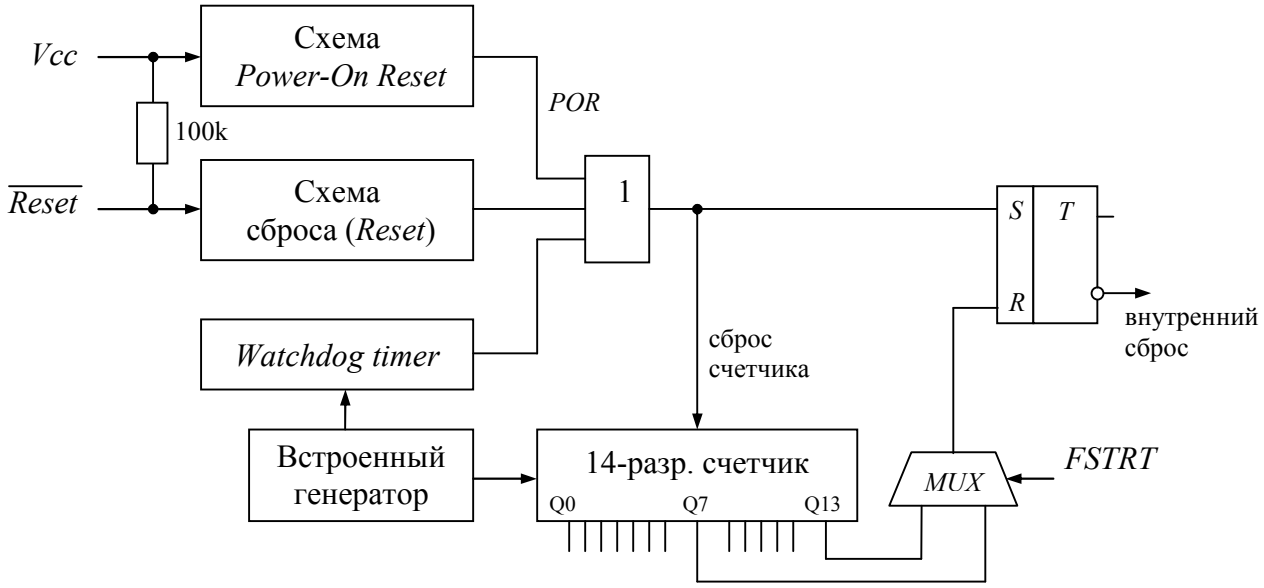

Рис. 3.7 – Структура цепей сброса

3.2 АССЕМБЛЕР

Все команды можно разделить на 4 большие группы:

- команды пересылки данных;
- команды вычислительного процесса;
- команды передачи управления;
- команды управления ходом работы МК.

Команды пересылки данных.

Эту группу команд можно разделить на три вида: обмен типа RG-RG, RG-mem, RG-порт.

MOV Rd,Rs – Rd=Rs
(mov R1, R20)

LDI Rd,const – Rd=const, в качестве Rd могут быть только 16 старших РОН
(ldi R22, $A5)

IN Rd,port – читает указанный порт (00-$3F) и помещает значение в Rd
(in R0, $10)

OUT port, Rs – записывает содержимое Rs в указанный порт (00-$3F)
(out $11, R4)

LDS Rd,addr – читает байт из памяти (addr - слово) и помещает в Rd
(lds R16, $0000 - аналогично mov R16, R0
lds R2, $0020 - аналогично in R2, $00
lds R2, $0080 - обращение к памяти)

STS Addr,Rs – сохраняет байт из Rs в памяти (addr - слово)
(sts $0055, R2)

LD Rd,index – загружает Rd байтом из памяти с адресом из индексного регистра X (R26:R27), Y (R28:R29) или Z (R30:R31) или R24:R25
(ld R1, X)

Команда LD Rd,index± – загрузка байта из памяти и инкрем./декрем. индексного RG
(ld R12,Y+ или ld R17,X-)

Команда LD Rd, ±index – инкрем./декрем. индексного RG и загрузка байта из памяти
(ld R12,+Y или ld R17,-X)

Команда LDD Rd,index+Ofs – загрузка байта в Rd из памяти по адресу определяемому суммой содержимого индексного регистра (Y или Z) и указанного смещения (0<=Ofs<=63)
(ldd R3,Y+6)

Команды ST index,Rd; ST index±,Rd; ST ±index,Rd; STD index+Ofs,Rd – аналогичны командам загрузки LD, но направление передачи – наоборот

Команда PUSH Rs – сохранить содержимое Rs в стеке ([sp]=Rs; sp=sp-1)

Команда POP Rd – извлечь из стека байт в Rd (sp=sp+1: Rd=[sp])

Команда LPM – чтение памяти программ: R0=[Z]

Резюме: команды типа LDS; STS; LPM выполняются за 3 цикла
команды типа LD; ST; LDD; STD; PUSH; POP выполняются за 2 цикла
остальные команды пересылки выполняются за 1 цикл

Команды арифметических, логических и битовых операций.

ADD Rd,Rs - Rd=Rd+Rs

ADC Rd,Rs - сложение значений регистров с учетом флага переноса

ADIW Rdd,const - сложение 16-разрядного регистра (R24,R26,R28,R30) с константой (0-63)
(adiw R27:R26,3 - X=X+3 как 16-разрядный регистр)

INC Rd - инкрементация значения регистра (не влияет на флаг переноса)

SUB Rd,Rs - Rd=Rd-Rs

SBC Rd,Rs - вычитание с учетом заема (флага переноса)

SUBI Rd,const - Rd=Rd-const (подобной команды сложения нет -> const<0 это решение)

SBCI Rd,const - вычитание из значения регистра константы с учетом заема

SBIW Rdd,const - вычитание из 16-разрядного регистра (R24,R26,R28,R30) константы (0-63)
DEC Rd - декрементация значения регистра (не влияет на флаг переноса)
NEG Rd - изменяет знак числа хранимого в Rd (Rd=0-Rd)
COM Rd - Rd=$FF-Rd

Команды сравнения CP Rd,Rs; CPC Rd,Rs и CPI Rd,const выполняют сравнение между двумя регистрами, регистрами с учетом переноса, а также регистром и константой соответственно. CPI работает только с 16-ю старших РОН, константа – 8 разрядов.

TST Rd – проверка битов: Z=((Rd or 0)==0); N=(Rd>>7); V=0; S=N
MUL Rd,Rs – R1:R0=Rd*Rs; C=((Rd*Rs)>>15)

Команды логического умножения: AND Rd,Rs; ANDI Rd,const; CBR Rd,const выполняют соответственно Rd=Rd&Rs; Rd=Rd&const; Rd=Rd&(not const). ANDI и CBR работают только с 16-ю старших РОН, константа 8 бит.

Команды логического сложения OR Rd,Rs; ORI Rd,const; SBR Rd,const выполняют соответственно Rd=Rd|Rs; Rd=Rd|const; Rd=Rd|(const). ORI и SBR работают только с 16-ю старших РОН, константа 8 бит.

Команда «исключающего или» EOR Rd,Rs - Rd=Rd xor Rs

Команды CLR Rd; SER Rd – устанавливают все биты регистра соответственно в нули и единицы.

SWAP Rd - меняет местами старшую и младшую тетрады регистра Rd
BST Rd, bit - передает указанный по номеру бит из регистра Rd в бит *T* регистра *SREG*
BLD Rd, bit - передает указанный по номеру бит из бита *T* регистра *SREG* в регистр Rd
BCLR bit - сбрасывает указанный по номеру бит в регистре *SREG* (см табл.3.2)
BSET bit - устанавливает указанный по номеру бит в регистре *SREG* (см табл.3.2)

Таблица 3.2 – Соответствие номера бита *SREG* битовым командам

bit	Бит регистра *SREG*	Синоним	
		сброс бита	уст-ка бита
000	Флаг переноса	clc	sec
001	Флаг нуля	clz	sez
010	Флаг отрицательного результата	cln	sen
011	Флаг переполнения	clv	sev
100	Флаг знака	cls	ses
101	Флаг промежуточного переноса	clh	seh
110	Временный бит	clt	set
111	Флаг разрешения прерываний	cli	sei

CBI ioreg, bit – сбрасывает указанный по номеру бит порта ioreg (только младшие 32 порта ввода-вывода)
SBI ioreg, bit – устанавливает указанный по номеру бит порта ioreg (только младшие 32 порта ввода-вывода)

Команды LSL Rd и LSR Rd – логический сдвиг регистра на один разряд влево и вправо соответственно, заполнение нулями, выдвигаемый бит передается во флаг переноса.

Команды ROL Rd и ROR Rd – циклический сдвиг регистра на один разряд влево и вправо соответственно с учетом флага переноса как 9-го разряда.

ASR Rd – арифметический сдвиг регистра вправо на один разряд – то же что и LSR, но заполнение знаковым разрядом.

Резюме: Все команды выполняются за 1 цикл, исключая: MUL, CBI, SBI – за 2 цикла.

Команды ветвления

rjmp Label – безусловный относительный переход на метку Label (±2 кб) //PC=PC+1+offs (2 байт)
rcall Label – вызов подпрограммы Label (±2 кб) //push PCL; push PCH; PC=PC+1+offs (2 байта)
jmp Label – безусловный переход на метку Label (4 Мб) //PC=offs (4 байта)
call Label – вызов подпрограммы Label (4 Мб) //push PCL; push PCH; PC=offs (4 байта)

ijmp – безусловный переход по значению индексного регистра Z (±2 кб) //PC=Z (2 байта)
icall – вызов подпрограммы по значению регистра Z //push PCL; push PCH; PC=Z (2 байта)
ret – возврат из подпрограммы //pop PCH; pop PCL (2 байта)
reti – возврат из подпрограммы прерывания //pop PCH; pop PCL; SREG.I=1 (2 байта)

Условные переходы (±63 байта)

brbc bit, Label – переход на метку Label если *SREG*.bit=0 (2 байта) //см табл. 3.3
brbs bit, Label – переход на метку Label если *SREG*.bit=1 (2 байта) //см табл. 3.3

Таблица 3.3 – Соответствие номера бита *SREG* команде ветвления

Бит состояния регистра SREG	Номер бита	Синоним команды	
		бит=0	бит=1
Флаг переноса	000	brcc, brsh	brcs, brlo
Флаг нуля	001	brne	breq
Флаг отрицательного результата	010	brpl	brmi
Флаг переполнения	011	brvc	brvs
Флаг знака	100	brge	brlt
Флаг промежуточного переноса	101	brhc	brhs
Временный бит	110	brtc	brts
Флаг разрешения прерываний	111	brid	brie

Команды пропуска

sbic RegIO, bit – пропустить следующую команду если RegIO.bit=0
sbis RegIO, bit – пропустить следующую команду если RegIO.bit=1
(*) sbic, sbis могут работать только с младшими 32 портами
srbc RG, bit – пропустить следующую команду если RG.bit=0
srbs RG, bit – пропустить следующую команду если RG.bit=1
cpse Rd,Rs – пропустить следующую команду если (Rd-Rs)=0

Резюме: Команды rjmp, rcall, jmp, icall выполняются за 3 цикла
call, ret, reti – 4 цикла; ijmp – 2 цикла; cpse – 1 цикл (не изменяет флагов !)
Остальные команды выполняются за 1-2 цикла (условие не выполняется – 1 цикл, условие выполняется и след. команда не jmp, call – 2 цикла, иначе – 3 цикла).

Прочие команды

NOP – ничего не делать в течении 1 цикла
WDR – сброс сторожевого таймера (1 цикл)
SLEEP – ожидать прихода прерывания (1 цикл)

Примеры написания программ

1. Реализация целочисленной КИХ фильтрации
   Пусть дан оцифрованный сигнал (4 бита на точку со знаком) в памяти по адресу $60 и длиной 50 байт. Коэффициенты КИХ фильтра длиной 7 хранятся в ПЗУ и представляют собой числа со знаком квантованные в 4 бита. Выполнить фильтрацию сигнала с округлением результата до 8 бит со знаком и разместить его на месте входного сигнала.
   Общие положения:
   - регистр Z указывает на коэффициенты фильтра;
   - регистр X указывает на исходные данные и используется для сохранения результата;
   - для представления результата свертки необходимо (3 бита исх. данные)+(3 бита коэф.)+(1 бит знака)+(3 бита на сумму свертки)=10 бит – округление результата на 2 бита;
   - R16 зарезервируем для организации внешнего цикла;
   - пару R18:R17 отведем для вычисления свертки;
   - R19 зарезервируем для организации внутреннего цикла.

……
LDI R26, $60; загружаем указатель на исх. данные – X

```
LDI R27, 0;  или EOR R27,R27
LDI R16, $60+50-7; адрес следующей ячейки за последним элементом массива исх. данных
L_FIR_ext:
    LDI R30, coef_FIR&$ff; загружаем указатель на коэффициенты фильтра – Z
    LDI R31, coef_FIR>>8
    LDI R17, 0; мл. часть свертки
    LDI R18, 0; ст. часть свертки
    LDI R19, 7; длина фильтра
    PUSH R26; сохранили значение указателя X (старший байт – всегда ноль, т.к. ОЗУ 128 байт)
    L_FIR_int:
        LPM; R0=коэф. фильтра
        INC R30; Z=Z+1 – к следующему коэффициенту
        LD R1,X+; загрузка отсчета сигнала и инкремент указателя
        MUL R0,R1; R1:R0=R0*R1
        ADD R17,R0
        ADC R18,R1
        DEC R19
        BRBC 1, L_FIR_int; переход если SREG.Z=0 (аналог: brne L_FIR_int)
    POP R26; восстановили значение указателя X
    ASR R18; сдвиг вправо ст. части свертки, выдвинутый бит – во флаге переноса
    ROR R17; сдвиг вправо мл. части свертки с заполнением из флага переноса
    ASR R18
    ROR R17; округленный результата свертки в R17
    ST R17,X+; сохр. результат на место исх. данных и инкремент указателя
    CPSE R28, R16; пропустить след. команду если X>=$60+50-7
                        (аналог CPI R28,const и brne label – дальность 63 байта против 2 кб)
    RJMP L_FIR_ext
……
coef_FIR:
    db 0,1,2,3,4,5,6; коэффициенты фильтра
……
```

2. Поиск максимального элемента массива данных.

Пусть имеется массив данных. Найти максимальный элемент массива.

Общие положения:

- массив расположен в памяти МК по адресу $60 и имеет длину 30 байт;
- регистр X указывает на текущий элемент массива;
- регистр R16 хранит максимальное значение массива.

```
……
LDI R26, $60; загрузили указатель на данные
LDI R27, 0
LD R16, X+; загружаем первый элемент, считая его максимальным, и инкрементируем
указатель
LDI R18, $60+30; адрес следующего за последним элементом массива
L_loop:
    LD R17, X+; загружаем очередной элемент массива и инкрементируем указатель
    CP R17, R16
    BRBS 4, L_next; переход если знак результата – отрицательный
        MOV R16,R17
    L_next:
    CPSE R26, R18
    RJMP L_loop
; R16 содержит максимальный элемент массива….
```

3.3 Аппаратные возможности МК

3.3.1 ТАКТОВЫЙ ГЕНЕРАТОР

Тактовый генератор представляет собой устройство, которое обеспечивает работу МК как самостоятельного устройства без навесных элементов, либо от внешнего генератора (сигнал подается на вход *XTAL*1), либо от внутреннего с подключением внешнего кварцевого резонатора к выводам *XTAL*1 и *XTAL*2.

3.3.2 СТОРОЖЕВОЙ ТАЙМЕР

Сторожевой таймер – устройство, предназначенное для перезапуска МК в случае его зависания. Структурная схема устройства представлена на рис.3.8. Таким образом, имеется возможность выключения *Watchdog*, задания восьми различных длительностей, программного перезапуска счетчика.

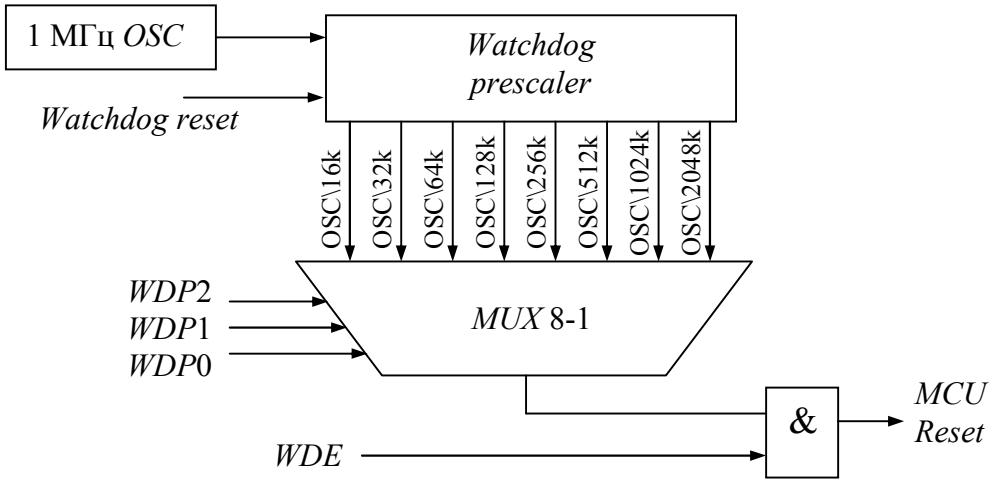

Рис. 3.8 – Структура *Watchdog*

Все эти функции реализуются с помощью управляющего регистра *WDTCR*. Перед включении *Watchdog* таймера необходимо его сбросить!!!

	7	6	5	4	3	2	1	0
WDTCR $21 ($41)	-	-	-	*WDTOE*	*WDE*	*WDP*2	*WDP*1	*WDP*0
	R	*R*	*R*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

WDE – (*Watchdog Enable*) установка в «1» разрешает работу *Watchdog*, установка «0» запрещает. Однако для выключения *Watchdog* необходимо установить в «1» бит разрешения *WDTOE*, значение которого автоматически обнуляется через 4 такта.

WDP2-WDP0 – (*Watchdog prescaler*) биты управляющие коэффициентом деления для задания различных интервалов сброса МК – см. табл.3.4.

Таблица 3.4 – Значения интервалов до сброса МК

*WDP*2	*WDP*1	*WDP*0	Делитель, циклов	Период сброса при V_{cc}=3 В	Период сброса при V_{cc}=5 В
0	0	0	16k	47 мс	15 мс
0	0	1	32k	94 мс	30 мс
0	1	0	64k	0,19 с	60 мс
0	1	1	128k	0,38 с	0,12 с
1	0	0	256k	0,75 с	0,24 с
1	0	1	512k	1,5 с	0,49 с
1	1	0	1024k	3,0 с	0,97 с
1	1	1	2048k	6,0 с	1,9 с

3.3.3 ТАЙМЕРЫ

Как уже отмечалось, тактирование таймеров может происходить разными сигналами. На рис.3.9 представлена структурная схема, из которой видны пути сигналов тактирующих счетчики таймеров.

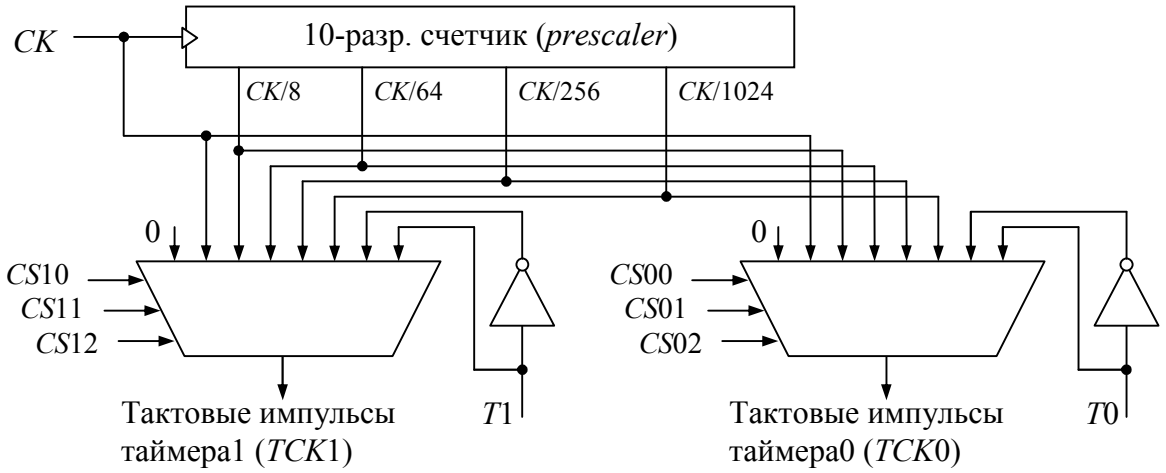

Рис. 3.9 – Структурная схема цепей тактирования таймеров МК

В данной модели МК реализованы два таймера на основе 8-разрядного счетчика и 16-разрядного счетчика. Оба счетчика могут работать только на одинаковой частоте (делитель один). Однако могут тактироваться разными внешними синхросигналами.

8-разрядный таймер, структурная схема которого показана на рис.3.10, с точки зрения программного управления имеет:

- регистр управления (*TCCR*0);
- флаг переполнения (*TIFR.TOV*0);
- флаг управления прерыванием при переполнении счетчика (*TIMSK.TOIE*0).

При тактировании счетчика внешним сигналом активным является передний фронт.

	7	6	5	4	3	2	1	0
TCCR0 $33 ($53)	-	-	-	-	-	*CS*02	*CS*01	*CS*00
	R	*R*	*R*	*R*	*R*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

CS02-CS00 – выбор источника тактирования счетчика – табл.3.5 (также см. рис.3.9)

Таблица 3.5 – Коэффициент деления частоты таймера 0

*CS*02	*CS*01	*CS*00	Описание
0	0	0	стоп (запрет счета)
0	0	1	*CK*
0	1	0	*CK*/8
0	1	1	*CK*/64
1	0	0	*CK*/256
1	0	1	*CK*/1024
1	1	0	внешний вывод *T*0, задний (*falling*) фронт
1	1	1	внешний вывод *T*0, передний (*rising*) фронт

(*) счетчик будет считать от внешнего сигнала, даже если соответствующий *pin* сконфигурирован как выход.

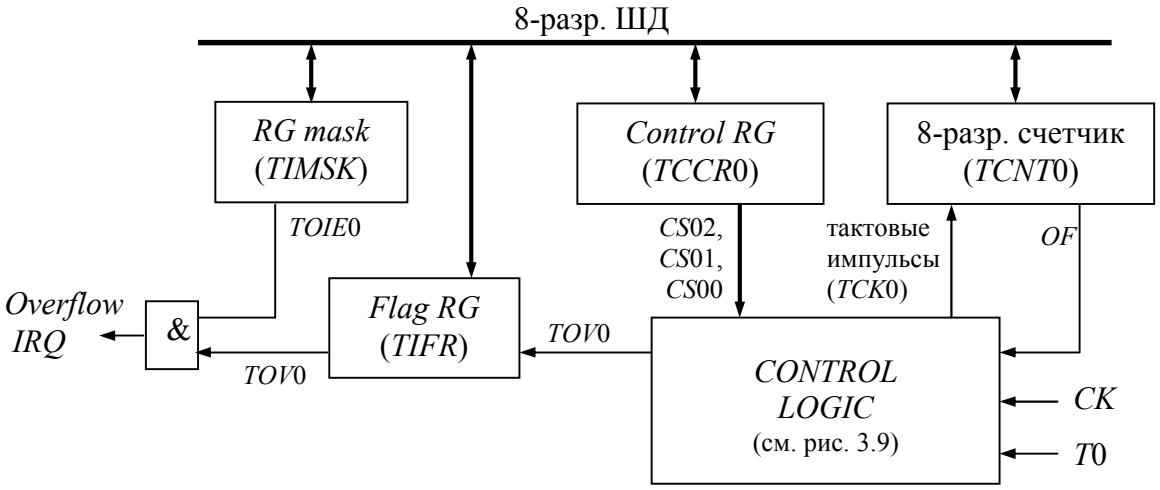

Рис. 3.10 – Структурная схема таймера 0

16-разрядный таймер, структурная схема которого показана на рис.3.11, с точки зрения программного управления имеет:

- регистры управления (*TCCR*1*A*, *TCCR*1*B*);
- флаг переполнения и его маска прерывания (*TIFR.TOV*1*, TIMSK.TOIE*1);
- регистр входного захвата (*ICR*1);
- флаг захвата и его маска прерывания (*TIFR.ICF*1*, TIMSK.TICIE*1);
- регистр сравнения (*OCR*1*A*);
- флаг сравнения и его маска прерывания (*TIFR.OCF*1*A, TIMSK.OCIE*1*A*).

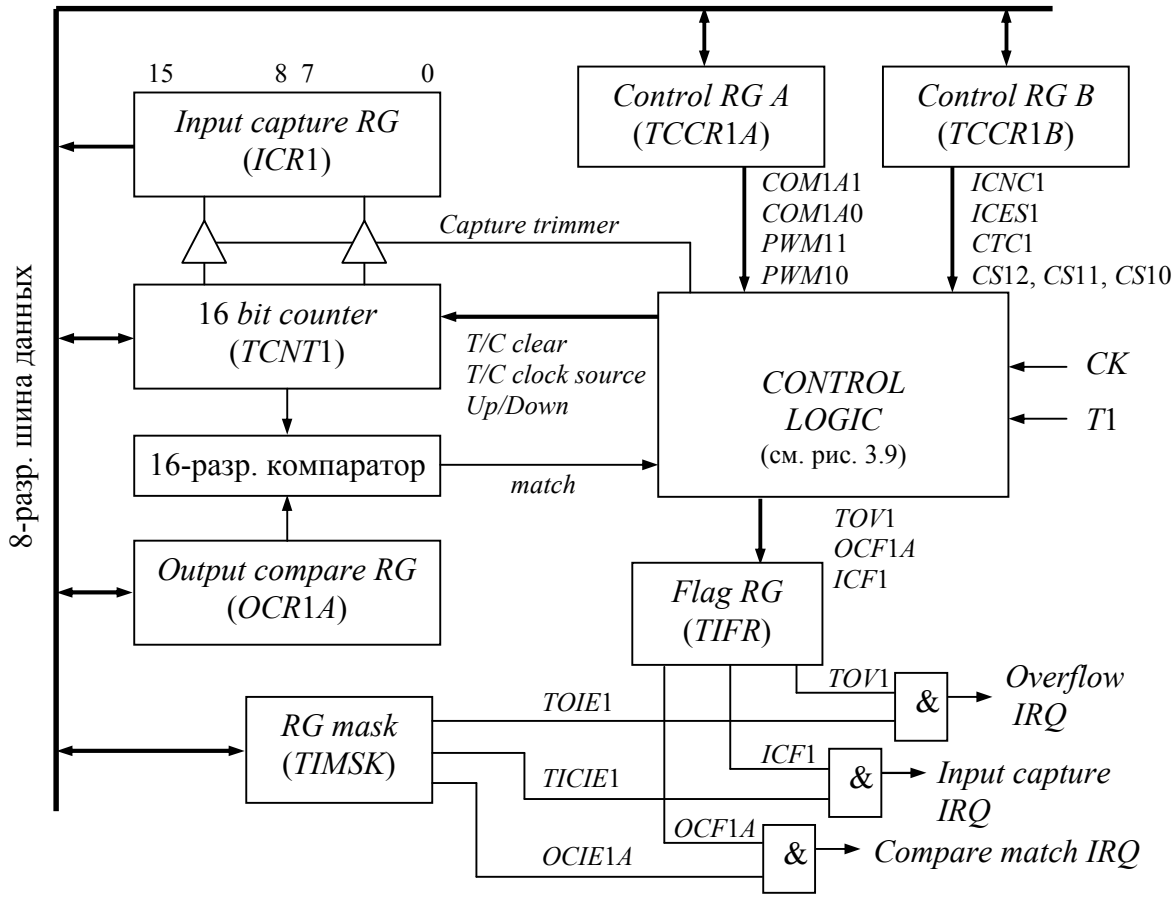

Рис. 3.11 – Структурная схема таймера 1

Возможности таймера 1 значительно шире по сравнению с таймером 0. В частности, с помощью встроенных регистра сравнения и компаратора возможно, чтобы счетчик работал как делитель на любой коэффициент (т.е. досчитывал от начального значения до значения регистра сравнения, сбрасывался, вызывая прерывание по необходимости, и начинал сначала). Таким образом, имеется возможность реализации генератора обеспечивающего широтно-импульсную модуляцию (8, 9, 10 бит).

Также имеется возможность аппаратного измерения длительностей внешних процессов, т.е. запустив счетчик на счет, схемы захвата могут зафиксировать значение счетчика в момент прихода внешнего сигнала. Причем для выделения активного сигнала (фронт или уровень) в МК встроена специальная схема, показанная на рис.3.12.

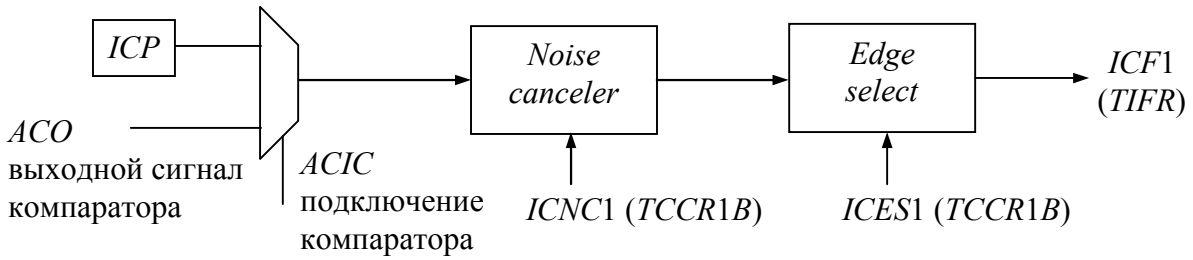

Рис. 3.12 – Структурная схема цепей захвата

Рассмотрим назначение бит управляющего регистра таймера 1.

	7	6	5	4	3	2	1	0
TCCR1A $2F ($4F)	*COM*1*A*1	*COM*1*A*0	-	-	-	-	*PWM*11	*PWM*10
	R/W	*R/W*	*R*	*R*	*R*	*R*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

COM1A1, **COM1A0** – (*Compare Output Mode* 1) определяют состояние внешнего вывода *OC*1 (*pin* 15) в момент совпадения значения счетчика и регистра сравнения:

0 0 - таймер отключен от вывода *OC*1;
0 1 - состояние линии *OC*1 изменяется на противоположное;
1 0 - на выходе *OC*1 устанавливается низкий уровень (лог. 0);
1 1 - на выходе *OC*1 устанавливается высокий уровень (лог. 1);
(*) Отметим, что в режиме ШИМ эти разряды имеют другой смысл.

PWM11, **PWM10** – (*Pulse Width Modulator*) выбирают режим ШИМ:

0 0 - ШИМ выключен;
0 1 - ШИМ 8 бит;
1 0 - ШИМ 9 бит;
1 1 - ШИМ 10 бит.

	7	6	5	4	3	2	1	0
TCCR1B $2E ($4E)	*ICNC*1	*ICES*1	-	-	*CTC*1	*CS*12	*CS*11	*CS*10
	R/W	*R/W*	*R*	*R*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

ICNC1 – (*Input Capture*1 *Noise Canceler*) если установлен в «0», то функция подавления шумов отключена (первый же активный уровень сигнала приведет к срабатыванию триггера); если «1» - функция включена (для фиксации триггером входного сигнала необходимо чтобы активный его уровень был зафиксирован на протяжении 4 тактов);

ICES1 – (*Input Capture*1 *Edge Select*) определяет активный фронт сигнала захвата; 0 – задний (падающий) фронт; 1 – передний (нарастающий);

CTC1 – (*Clear Timer/Counter*1 *on Compare match*) если установлен в «1», то при равенстве значений счетчика и регистра сравнения *OCR*1*A* значение счетчика обнуляется (сравнение выполняется за 1 цикл независимо от коэффициента деления) кроме ШИМ режима;
CS12, CS11, CS10 – (*Clock Select*) определяют коэффициент деления частоты, а также выбирают источник тактирования в соответствии с табл.3.5 (заменяя индексы *CS*0*x* на *CS*1*x* и *T*0 на *T*1).

Назначение разрядов текущего значения счетчика таймера очевидно, однако поскольку последний 16-разрядный, а шина 8-разрядная, то есть определенные условия, позволяющие одновременно прочитать/записать значение счетчика. Для этих целей в МК встроен дополнительный регистр, обеспечивающий временное хранение одного байта при доступе к 16-разрядному значению регистра таймера:

- для записи слова (ядро МК в таймер) необходимо записывать сначала старшую половину слова (т.е. в регистр *TCNT*1*H*), значение которого будет записано во временный регистр, а затем записывать младшую половину слова (т.е. в регистр *TCNT*1*L*), значение которого, объединяясь со значением временного регистра, одновременно записывается в счетчик;
- для чтения слова (таймер в ядро МК) необходимо сначала читать младший регистр *TCNT*1*L* при этом значение старшей половины будет автоматически одновременно с младшей размещено во временном регистре, а следующее чтение из старшей половины приведет к передаче значения из временного регистра.

	7	6	5	4	3	2	1	0
TCNT1H $2D ($4D)	b15	b14	b13	12	b11	b10	b9	b8
TCNT1L $2C ($4C)	b7	b6	b5	b4	b3	b2	b1	b0
Доступ	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

Аналогичным образом происходит обращение к регистру сравнения (*OCR*1) и регистру захвата (*ICR*1 только чтение). При этом используется один и тот же временный регистр!

	7	6	5	4	3	2	1	0
OCR1AH $2B ($4B)	b15	b14	b13	12	b11	b10	b9	b8
OCR1AL $2A ($4A)	b7	b6	b5	b4	b3	b2	b1	b0
Доступ	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
ICR1H $25 ($45)	b15	b14	b13	12	b11	b10	b9	b8
ICR1L $24 ($44)	b7	b6	b5	b4	b3	b2	b1	b0
Доступ	*R*	*R*	*R*	*R*	*R*	*R*	*R*	*R*
Начал. знач.	0	0	0	0	0	0	0	0

Далее, рассмотрим таймер 1 в режиме ШИМ. Когда включен режим ШИМ регистр *OCR*1 конфигурируется как 8-, 9- или 10-разрядный регистр, счетчик свободно считает от нуля до максимального значения и обратно к нулю при этом частота сигнала определяется в соответствии с табл. 3.6.

Таблица 3.6 – Выбор частоты ШИМ сигнала

PWM разрешение	*TOP* значение таймера	Частота
8 бит	255 (0FFh)	$f_{TC1}/510$
9 бит	511 (01FFh)	$f_{TC1}/1022$
10 бит	1023 (03FFh)	$f_{TC1}/2046$

При совпадении значения счетчика со значением регистра *OCR*1 выход ШИМ устройства *OC*1 (*PB*3) изменяет свое состояние в соответствии с табл.3.7 (битами регистра *TCCR*1).

Таблица 3.7 – Функциональность сигнала *OC*1

*COM*1*A*1	*COM*1*A*0	Действия
0	0	Выход *OC*1 не подключен
0	1	
1	0	*OC*1 сбрасывается в «0» при совпадении и счете вверх, устанавливается в «1» при совпадении и счете вниз
1	1	Наоборот (инвертированный ШИМ)

(*) Начальное состояние выхода *OC*1 неопределено.

(*) Запись *OCR*1*A* происходит через временный регистр – необходимо для синхронизации, полноту формирования импульса.

Пример работы с таймером – тахометр.

Пусть события (изменение сигнала) поступает на вход *INT*0 (вход *pin*6), таймер 0 осуществляет отсчет интервалов времени. Глобальная переменная R16 считает число событий, глобальная переменная *R*17 считает число полных циклов таймера (до 16). Отношение значения *R*16 к интервалу времени *T*=(*n**256**T*0) равно числу событий в единицу времени. Причем, выбирая делитель частоты таймера и параметр *n* так, чтобы величина *T* была близка к 1 сек., *R*16 содержит измеряемую величину (*n*=16, делитель 1024).

```
rjmp main;  вектор сброса
rjmp couse; вектор обработчика прерывания INT0
nop
nop
nop
rjmp full; вектор обработчика прерывания переполнения таймера 0
nop
nop
nop
nop
nop
main:
ldi R16, 0; иниц. глобальных переменных
ldi R17, 16
; настраиваем стек в конец памяти
ldi R20, $60+63
out $3d, R20; SPL=$60+63
; инициализируем таймер 0, разрешаем прерывания
bset 7; SREG.7=1 – разрешить IRQ (аналог sei)
; разрешаем прерывания таймера 0 по переполнению: TIMSK.1=1
in R20, $39; читаем TIMSK
ori R20, 2; установили бит 1
out $39, R20
; разрешаем прерывания по INT0: GIMSK.6=1
```

```
in R20, $3b; читаем GIMSK
ori R20, 64; установили бит 6
out $3b, R20
; выбираем тип активного сигнала на линии INT0 – низкий уровень
in R20, $35; читаем MCUCR
andi R20, 255-3; сбросили два мл. бита
out $3b, R20
; Устанавливаем делитель таймера 0, запуская счет
ldi R20, 5; установили биты CS02, CS01, CS00 – коэф. деления 1024
out $33, R20; в TCCR0
loop:
    brbc 1, loop; переход если SREG.Z=0 (R17>0)
; R16 содержит число событий в единицу времени
……
; например, вывод на индикатор (экран)
……
; реинициализация глобальных переменных
ldi R17,16
ldi R16,0
rjmp main
; обработчики прерываний
cause:
    inc R16; зарегистрировали событие
    reti
full:
    dec R17
    reti
……
```

3.3.4 ПОРТЫ ВВОДА/ВЫВОДА И ИХ АЛЬТЕРНАТИВНЫЕ ФУНКЦИИ

Порт *B* является 8-разрядным с независимым программированием каждого вывода порта. Физически порт обеспечивается тремя адресами:

Data RG – **PORTB**

	7	6	5	4	3	2	1	0
PORTB $18 ($38)	b7	b6	b5	b4	b3	b2	b1	b0
	R/W	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

Direction RG – **DDRB**

	7	6	5	4	3	2	1	0
DDRB $17 ($37)	b7	b6	b5	b4	b3	b2	b1	b0
	R/W	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

Input pins RG – **PINB**

	7	6	5	4	3	2	1	0
PINB $16 ($36)	b7	b6	b5	b4	b3	b2	b1	b0
	R	*R*	*R*	*R*	*R*	*R*	*R*	*R*
Начал. знач.	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a

Ниже в таблице представлено определение каждого *n* бита порта.

DDRBn	*PortBn*	*I/O*	*Pull-up*	Описание
0	0	*input*	-	Высокоомный вход
0	1	*input*	+	Подтянутый к *Vcc* вход
1	0	*output*		Выход – лог. 0
1	1	*output*		Выход – лог. 1

Альтернативные функции порта

Бит порта	Альтернативная функция
*PB*0	*AIN*0 - неинверсный вход аналогового компаратора
*PB*1	*AIN*1 - инверсный вход аналогового компаратора
*PB*3	*OC*1 - выход таймера/счетчика1 по совпадению (*compare match output*)
*PB*5	*MOSI* - выход данных *SPI* интерфейса
*PB*6	*MISO* - вход данных *SPI* интерфейса
*PB*7	*SCK* - линия тактирования *SPI* интерфейса

Также интерфейс *SPI* применяется при загрузке *Flash* памяти (программирование МК)

Для лучшего понимания функций порта рассмотрим его функциональную схему. На рис.3.13-3.18 показаны схемы драйверов выводов порта *B*, где
RD, *WD* – стробы чтения и записи регистра направления (*DDRB*);
RL, *WP* – стробы чтения и записи регистра порта (*PORTB*);
RP – строб чтения вывода порта;
Reset – внутренний сигнал сброса (установки нуля).

Нагрузочная способность портов 20 мА.

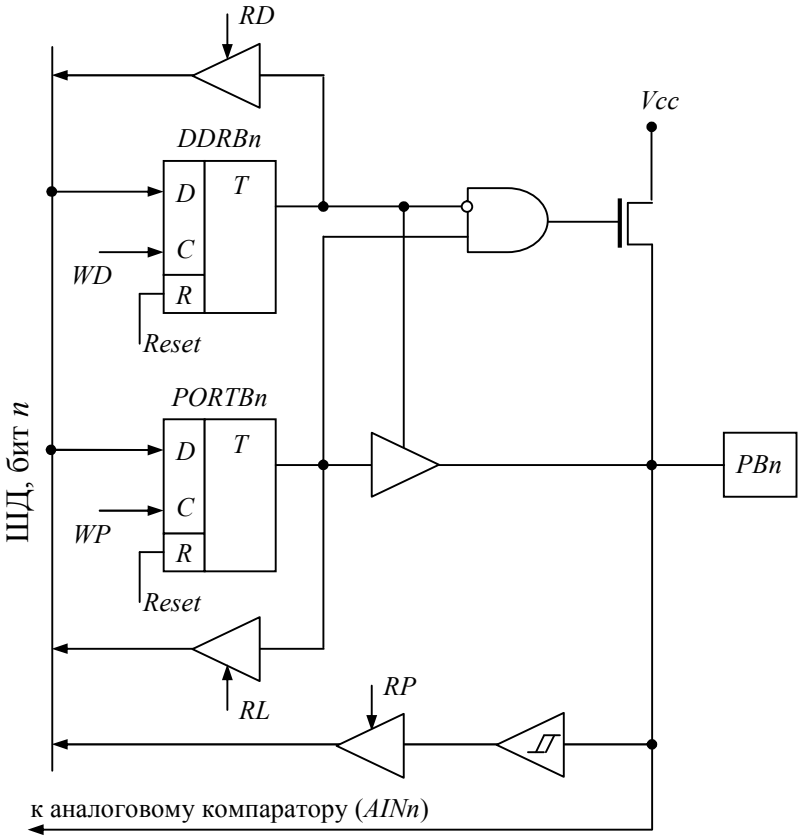

Рис. 3.13 – Схема линий порта *PBn* (*n*=0, 1)

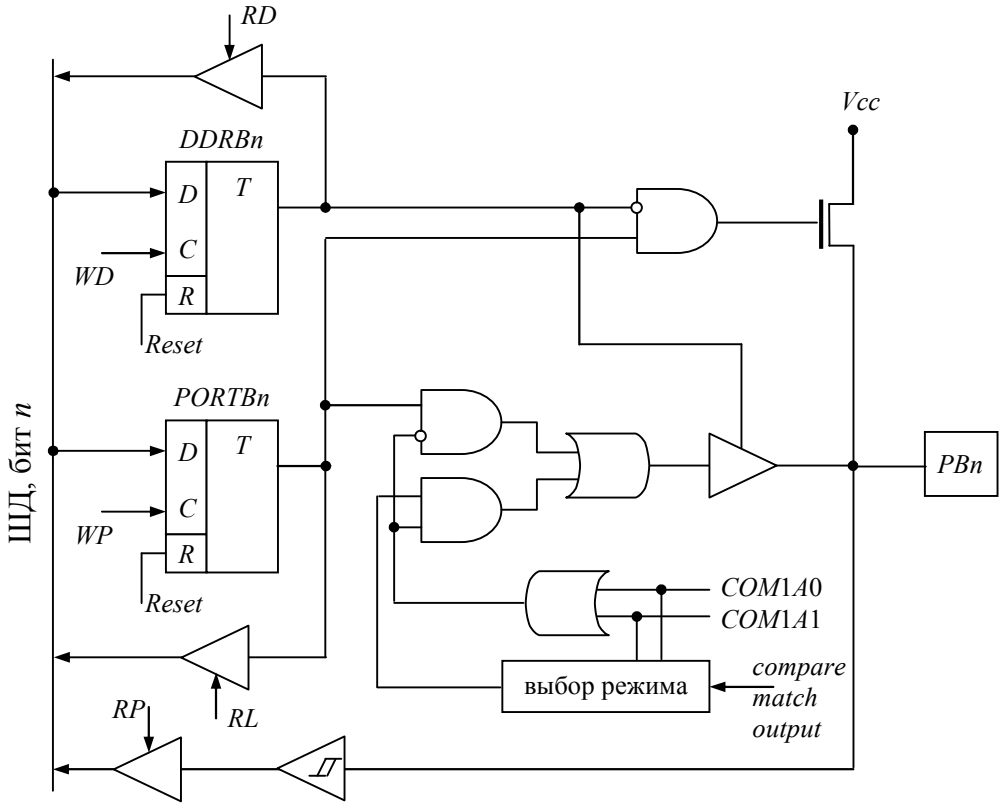

Рис. 3.14 – Схема линий порта *PBn* (*n*=3)

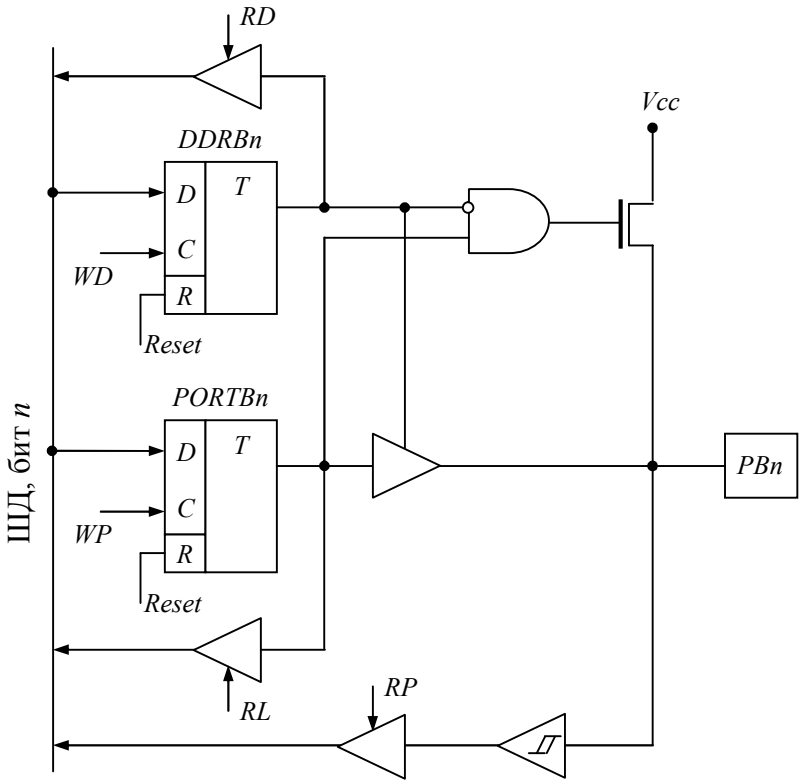

Рис. 3.15 – Схема линий порта *PBn* (*n*=2, 4)

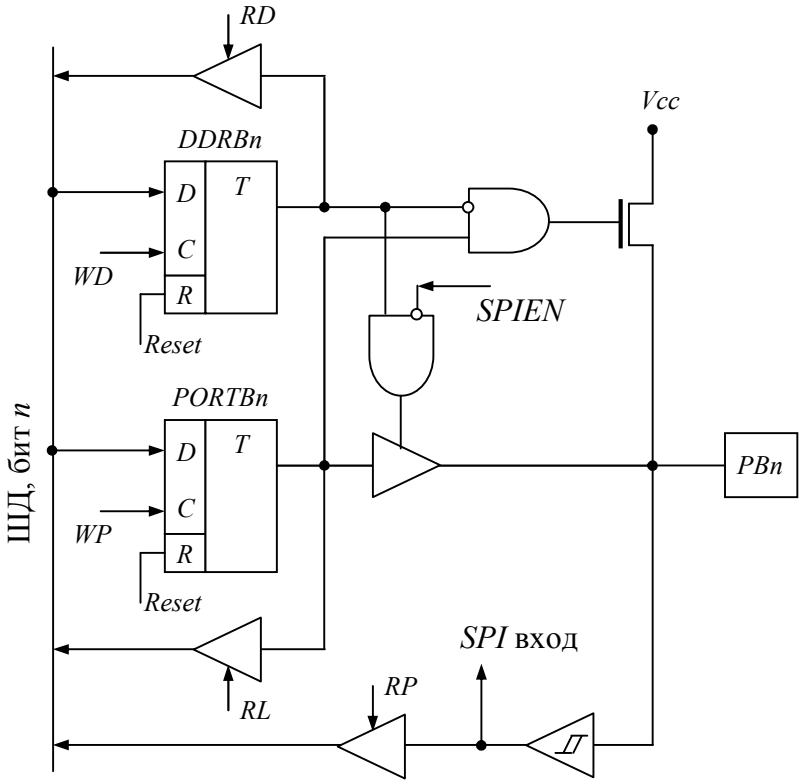

Рис. 3.16 – Схема линий порта *PBn* (*n*=5)

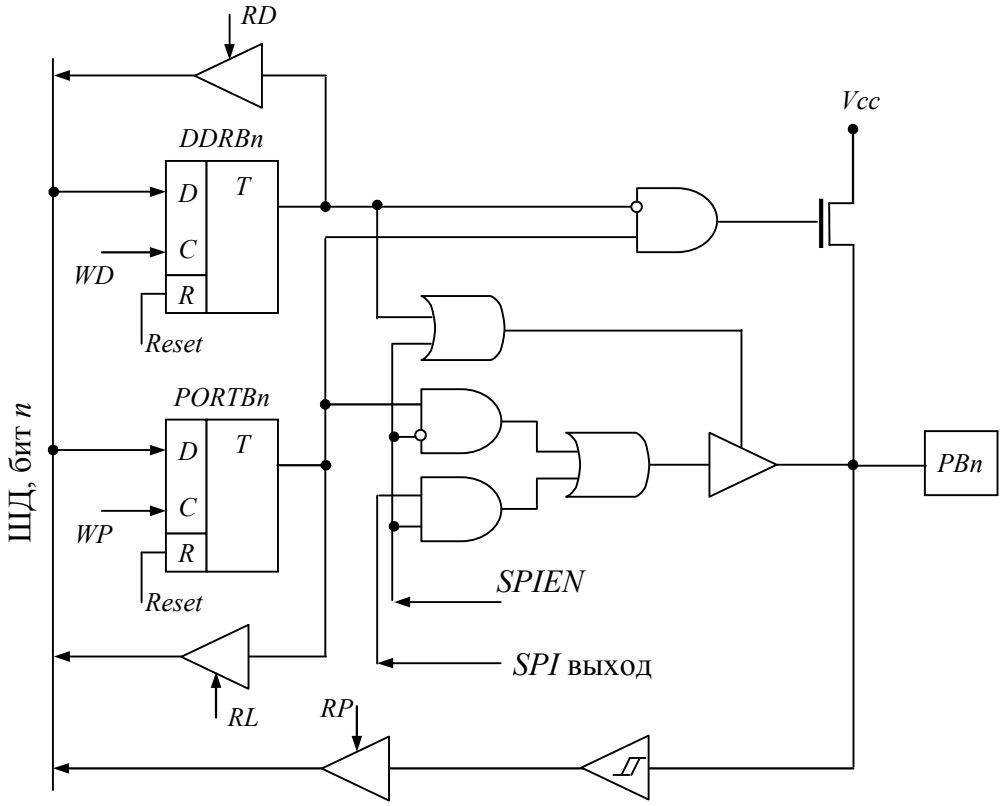

Рис. 3.17 – Схема линий порта *PBn* (*n*=6)

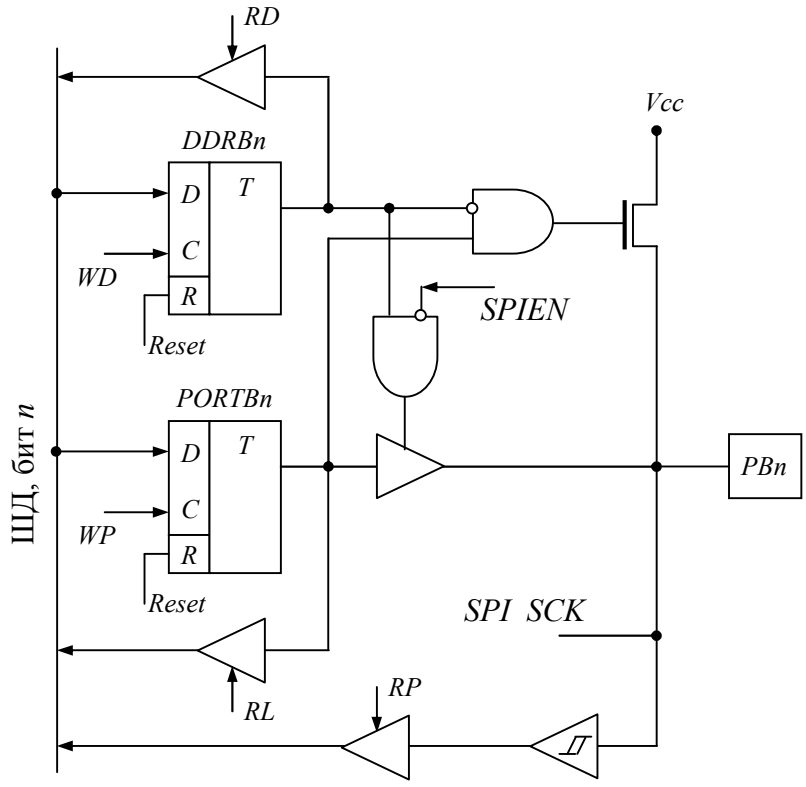

Рис. 3.18 – Схема линий порта *PBn* (*n*=7)

Порт *D* является 7-разрядным с независимым программированием каждого вывода порта. Физически порт обеспечивается тремя адресами:

Data RG – **PORTD**

	7	6	5	4	3	2	1	0
PORTD $12 ($32)	-	b6	b5	b4	b3	b2	b1	b0
	R	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

Direction RG – **DDRD**

	7	6	5	4	3	2	1	0
DDRD $11 ($31)	-	b6	b5	b4	b3	b2	b1	b0
	R	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

Input pins RG – **PIND**

	7	6	5	4	3	2	1	0
PIND $10 ($30)	-	b6	b5	b4	b3	b2	b1	b0
	R	*R*	*R*	*R*	*R*	*R*	*R*	*R*
Начал. знач.	0	n/a	n/a	n/a	n/a	n/a	n/a	n/a

Ниже в таблице представлено определение каждого *n* бита порта.

DDRDn	*PortDn*	*I/O*	*Pull-up*	Описание
0	0	*input*	-	Высокоомный вход
0	1	*input*	+	Подтянутый к *Vcc* вход
1	0	*output*		Выход – лог. 0
1	1	*output*		Выход – лог. 1

Альтернативные функции порта

Бит порта	Альтернативная функция
*PD*0	*RxD* - вход приемника *UART*
*PD*1	*TxD* - выход передатчика *UART*
*PD*2	*INT*0 - вход внешнего прерывания
*PD*3	*INT*1 - вход внешнего прерывания
*PD*4	*T*0 - вход таймера/счетчика0
*PD*5	*T*1 - вход таймера/счетчика1
*PD*6	*ICP* - вход схемы захвата таймера/счетчика1

Для лучшего понимания функций порта рассмотрим его функциональную схему. На рис.3.19-3.23 показаны схемы драйверов выводов порта *D*, где
RD, *WD* – стробы чтения и записи регистра направления (*DDRD*);
RL, *WP* – стробы чтения и записи регистра порта (*PORTD*);
RP – строб чтения вывода порта;
Reset – внутренний сигнал сброса (установки нуля).

Нагрузочная способность портов 20 мА.

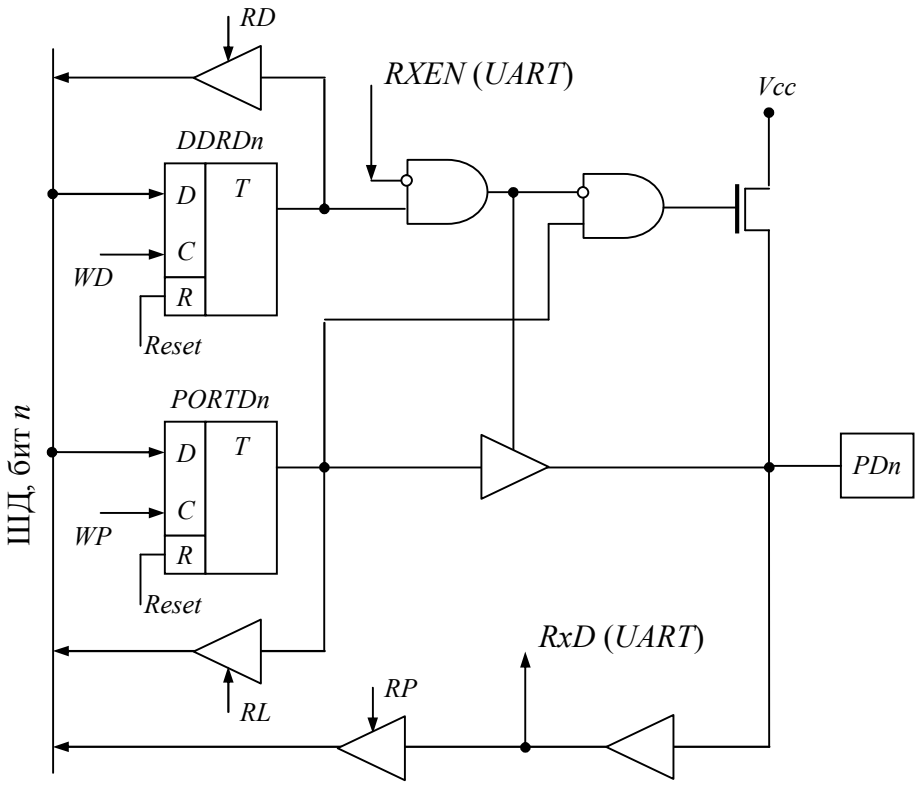

Рис. 3.19 – Схема линий порта *PDn* (*n*=0)

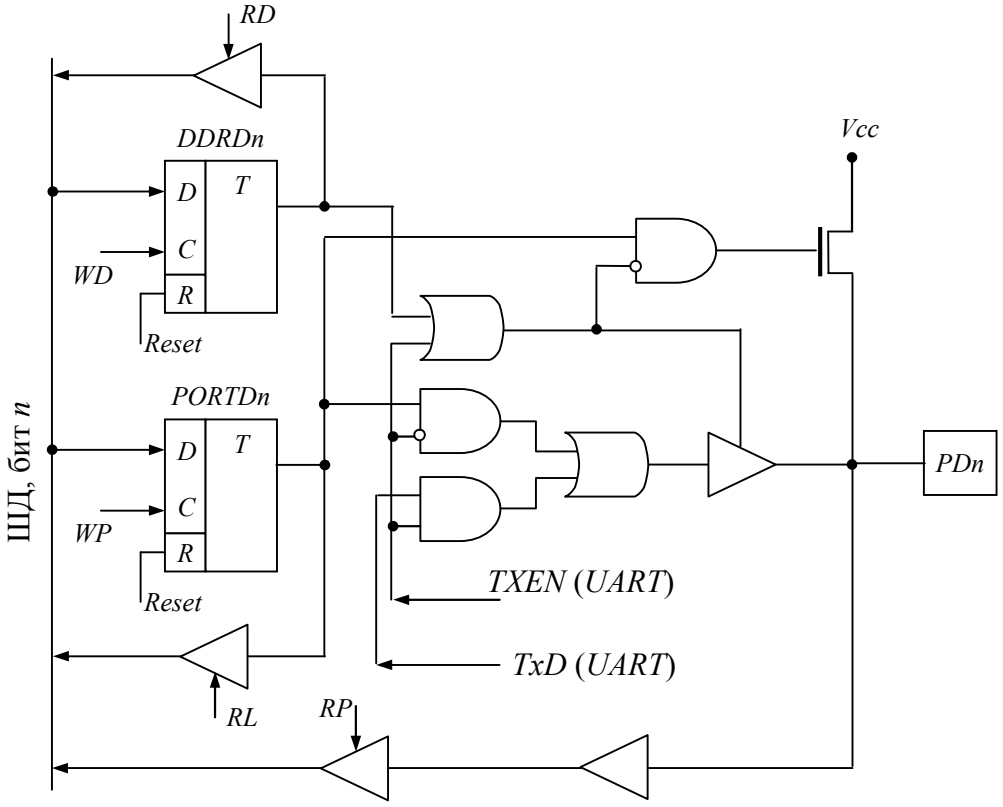

Рис. 3.20 – Схема линий порта *PDn* (*n*=1)

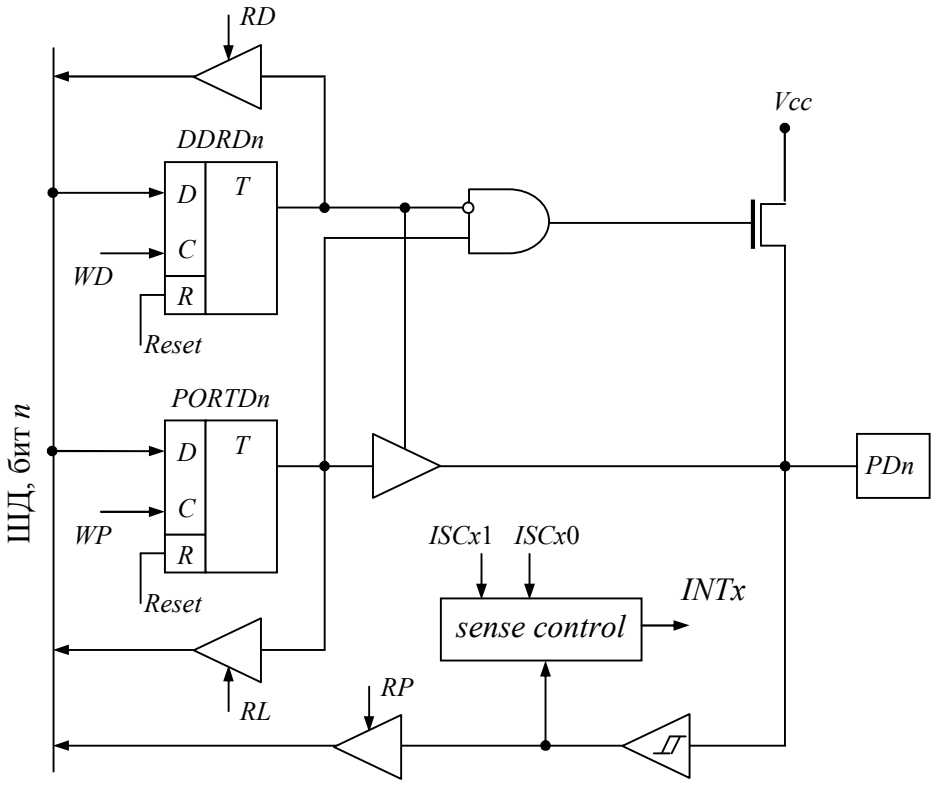

Рис. 3.21 – Схема линий порта *PDn* (*n*=2, 3)

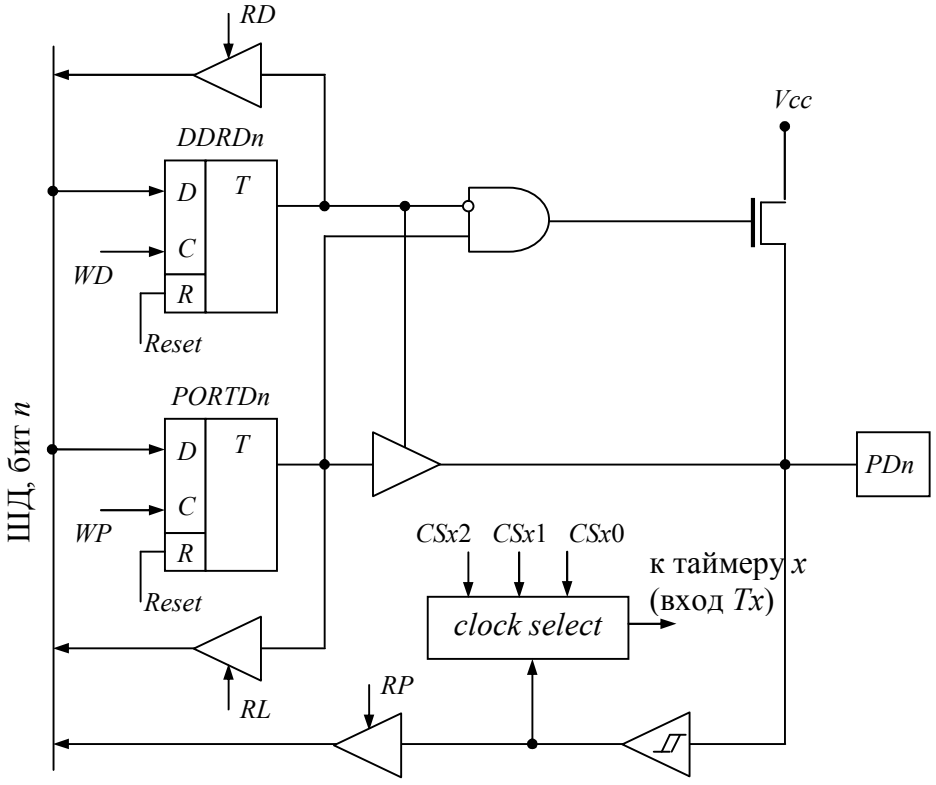

Рис. 3.22 – Схема линий порта *PDn* (*n*=4, 5)

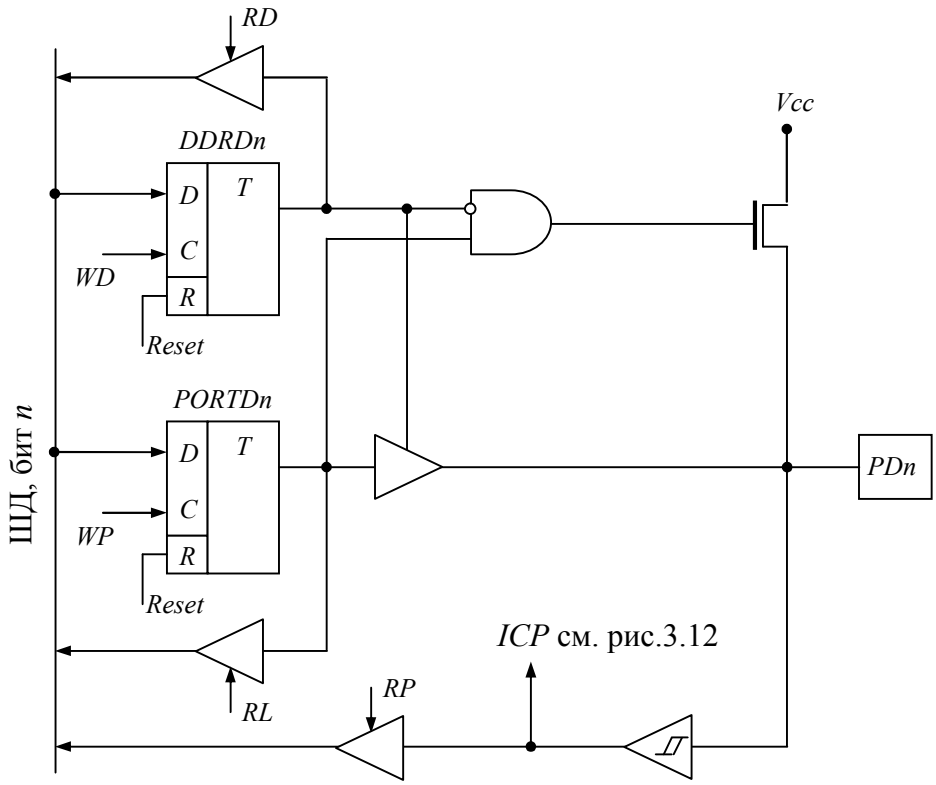

Рис. 3.23 – Схема линий порта *PDn* (*n*=6)

3.3.5 АНАЛОГОВЫЙ КОМПАРАТОР

Структурная схема компаратора представлена на рис.3.24.

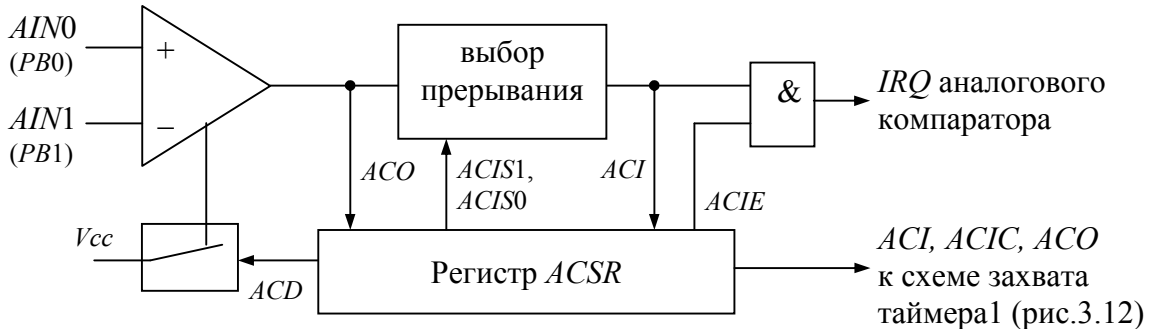

Рис. 3.24 – Структурная схема компаратора

Управление компаратором осуществляется с помощью одного регистра *ACSR* (*Analog Comparator Control and Status RG*).

	7	6	5	4	3	2	1	0
ACSR $08 ($28)	*ACD*	-	*ACO*	*ACI*	*ACIE*	*ACIC*	*ACIS*1	*ACIS*0
	R/W	*R*	*R*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

ACD (*Analog Comparator Disable*) – если установлен в «1», то питание компаратора выключено, это снижает потребление МК.
ACO (*Analog Comparator Output*) – прямой выход ОУ.
ACI (*Analog Comparator Interrupt Flag*) – устанавливается при возникновении прерывания от компаратора, автоматически сбрасывается при выполнении процедуры обслуживания запроса, для программного сброса бита необходимо записать лог. «1».
ACIE (*Analog Comparator Interrupt Enable*) – разрешение прерываний от компаратора.
ACIC (*Analog Comparator Input Enable*) – если установлен в «1», то выход компаратора коммутируется на вход схемы захвата таймера 1 (иначе выход компаратора и таймер 1 не соединены).
ACIS1, ACIS0 (*Analog Comparator Interrupt Mode Select*) – определяют тип срабатывания компаратора в соответствии с табл. 3.8.

Таблица 3.8 – Выбор типа прерываний аналогового компаратора

*ACIS*1	*ACIS*0	Вид прерывания
0	0	Прерывание по переключению выхода (изменению *ACO*)
0	1	Зарезервировано
1	0	Прерывание по спадающему фронту *ACO* (из "1" в "0")
1	1	Прерывание по нарастающему фронту *ACO* (из "0" в "1")

Пример – мигание светодиода по таймеру (подключен к выводу *PB*7 и земле через *R*).

```
rjmp main
nop
rjmp switch
nop
nop
nop
nop
nop
nop
nop
nop
switch:
    sbis $18, 7;  если PB7=1 пропустить след. команду – светодиод включен, надо выкл.
      rjmp L01; светодиод выключен, надо включить
    cbi $18, 7; выключить светодиод
    reti
    L01:
    sbi $18, 7; включить светодиод
    reti
main:
ldi R16, $60+63
out $3d, R16; SPL=$60+63
ldi R16, $80
out $17, R16; DDRB.7=1 – выход, остальные – входы
clr R16
out $18, R16; PortB=0, начальное состояние – выключенный светодиод
ldi R16, 5
out $33, R16; делитель таймера 0  =1024
ldi R16, 2
out $39, R16; размаскировали IRQ таймера
sei; SREG.7=1
loop:
rjmp loop
```

3.3.6 ПОСЛЕДОВАТЕЛЬНЫЙ ВВОД/ВЫВОД (*UART*)

*AT*90*S*2313 обеспечивает полный дуплекс при последовательном обмене данными. Основные возможности *UART* (*Universal Asynchronous Receiver and Transmitter*):

- данные 8 или 9 бит;
- встроенная фильтрация шумов;
- обнаружение перегрузки;
- обнаружение ошибки кадра;
- обнаружение ложного старт бита;
- независимые источники прерываний: передача закончена, регистр передачи пуст и прием окончен.

Структурная схема передатчика приведена на рис.3.25. Функционирование передатчика начинается после записи в регистр передачи *UDR* (*UART I/O Data RG*) собственно данных подлежащих передаче. Далее эти данные передаются в сдвиговый регистр:

- немедленно, если символ был записан после выдвижение в линию стоп бита предыдущего символа;
- после окончания передачи предыдущего символа (после передачи его стоп бита), если символ был записан во время передачи предыдущего.

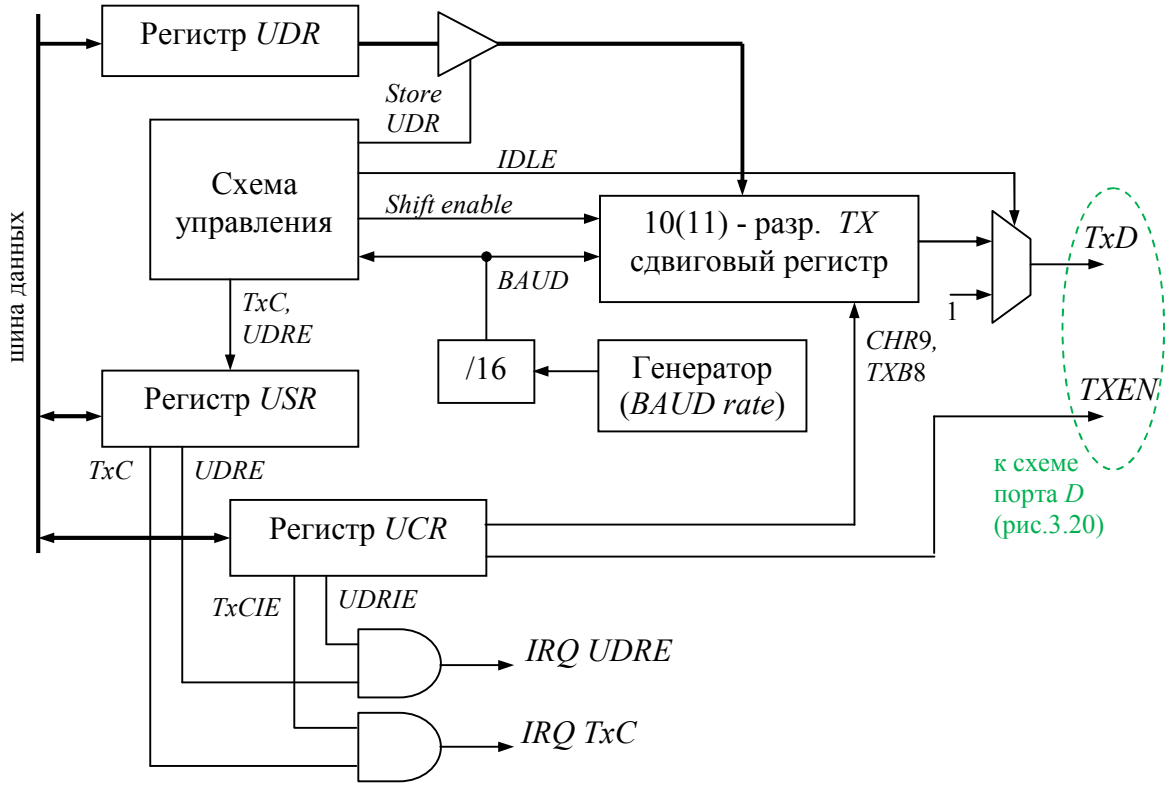

Рис. 3.25 – Структурная схема передатчика *UART*

Если сдвиговый регистр пуст, то после передачи данных из *UDR* в регистр сдвига бит *UDRE* (*UART Data RG Empty*) регистра статуса *USR* (*UART Status RG*) устанавливается в «1», что означает готовность передатчика к приему следующего символа для передачи в линию.

После размещения данных в регистре сдвига нулевой бит сбрасывается (старт бит), 9 или 10 бит устанавливается (стоп бит). Если выбрана длина символа в 9 бит (бит *CHR*9 регистра управления *UCR* установлен), то бит 0 регистра *UCR* (обозначен *TXB*8) предается в бит 9 регистра сдвига.

При передаче первым выдвигается старт бит, за ним – младший бит символа и т.д. После выдвижения стоп бита регистр сдвига заполняется новым символом из регистра *UDR*. Если этот регистр пуст, то бит *TXC* (*TX Complete Flag*) регистра *USR* устанавливается в «1».

Включением/выключением передатчика управляет бит *TXEN* регистра *UCR*. Если передатчик включен (*TXEN*=1) то вывод МК *PD*1 используется как выход передатчика *UART* независимо от настроек порта. В противном случае этот *pin* работает как порт.

Структурная схема приемника приведена на рис.3.26.

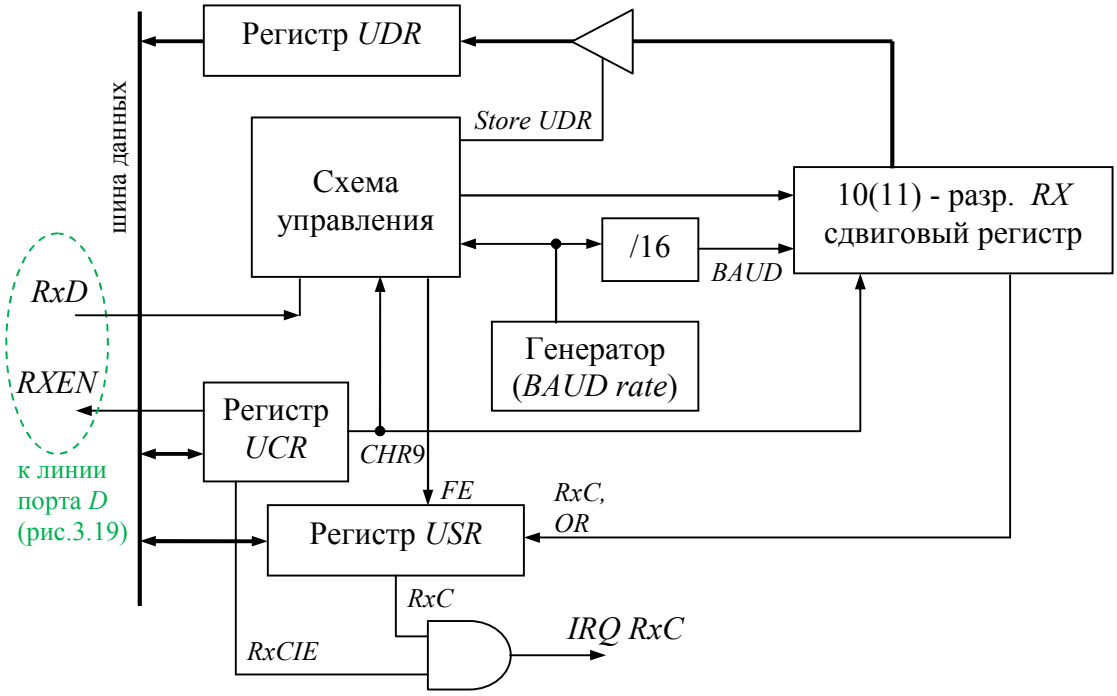

Рис. 3.26 – Структурная схема приемника *UART*

Входные цепи приемника непрерывно следят за состоянием линии с частотой дискретизации в 16 раз большей частоты передачи данных. Это позволяет отследить фронт с лог. «1» в лог. «0», т.е. обнаружить старт бит. Однако для начала приема этого недостаточно. Например, пусть 1 отсчет оказался низким уровнем, тогда приемник, продолжая наблюдать сигнал, считывает 8, 9 и 10 отсчеты. Если любые два или более этих отсчетов будут содержать лог. «1», то принятый фронт – помеха и приемник перейдет в режим поиска очередного спадающего фронта. Если после обнаружения фронта хотя бы два отсчета из трех (8, 9, 10 отсчеты) содержат низкие уровни, то принимаемая последовательность – данные, которые будут приниматься по тому же правилу: два из трех. Это обеспечивает дополнительную помехоустойчивость приема данных – рис.3.27, где "черточками" показаны моменты чтения линии *RxD*, длинные "черточки" соответствуют 8, 9, 10 отсчетам (нумерация с единицы).

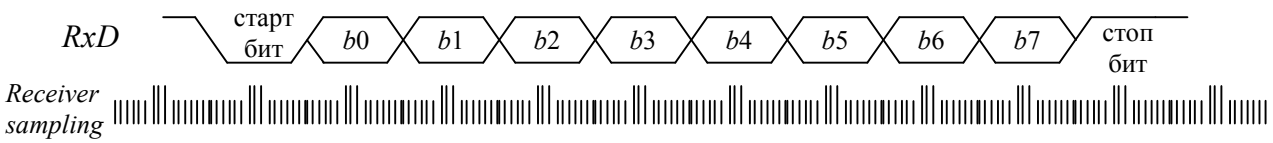

Рис. 3.27 – Мажоритарный принцип приема сигнала из цифрового канала

При окончании приема ожидается получение отсчетов содержащих лог. «1» (стоп бит). Если при этом из трех отсчетов хотя бы два содержат низкий уровень, то фиксируется ошибка кадра *FE* (*Framing Error*) и соответствующий бит регистра статуса *USR* устанавливается в «1».

В независимости от того был ли принят стоп бит или произошла ошибка при его приеме данные из сдвигового регистра приемника передаются в регистр *UDR*, а бит окончания приема *RXC* в регистре статуса *USR* устанавливается в «1».

Если при приеме установлен режим 9-разрядных данных (бит *CHR*9=1 регистра *UCR*), то бит *RXB*8 регистра *UCR* принимает значение бита 9 регистра сдвига.

Если при приеме очередного символа оказывается, что регистр *UDR* не был прочитан фиксируется ошибка перегрузки *OR* (*OverRun*) и соответствующий бит регистра *USR* устанавливается в «1». Это означает, что принятый символ не может быть передан из сдвигового регистра в *UDR* и его значение будет потеряно первым же обнаруженным фронтом старт бита. Значение бита *OR* буферизировано, т.е. обновляется при чтении регистра *UDR* (проверять ошибку перегрузки надо после чтения принятого символа).

Включением/выключением приемника управляет бит *RXEN* регистра *UCR*. Если приемник включен (*RXEN*=1) то вывод МК *PD*0 используется как вход приемника *UART* независимо от настроек порта. В противном случае этот *pin* работает как порт.

UART обслуживается четырьмя регистрами: *RG* данных приема/передачи, *RG* управления, *RG* статуса и *RG* управления скоростью обмена.

UDR (*UART I/O Data RG*) – регистр данных устройства.

	7	6	5	4	3	2	1	0
UDR $0C ($2C)	b7	b6	b5	b4	b3	b2	b1	b0
	R/W	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

Под одним адресом физически скрыты два регистра – приемника по чтению и передатчика по записи.

USR (*UART Status RG*) – статусный регистр устройства.

	7	6	5	4	3	2	1	0
USR $0B ($2B)	*RxC*	*TxC*	*UDRE*	*FE*	*OR*	-	-	-
	R	*R/W*	*R/W*	*R/W*	*R/W*	*R*	*R*	*R*
Начал. знач.	0	0	1	0	0	0	0	0

RxC (*UART Receive Complete*) – устанавливается в «1» когда принимаемый символ передан из сдвигового регистра в регистр данных *UDR*, установка бита происходит независимо от наличия ошибки кадра. Бит сбрасывается при чтении регистра. Если в регистре управления разрешены прерывания, то установка этого бита вызывает соответствующее прерывание.
TxC (*UART Transmitte Complete*) – устанавливается в «1» когда последний бит текущего символа, включая служебные биты, был выдвинут из сдвигового регистра в линию и регистр данных *UDR* не содержит нового символа для передачи. Если в управляющем регистре разрешены прерывания, то установка бита вызывает соответствующее прерывание. Бит сбрасывается автоматически при начале выполнении обработчика прерывания. Бит можно сбросить программно записью в него лог. «1».
UDRE (*UART Data RG Empty*) – устанавливается когда символ из регистра *UDR* передан в освободившейся сдвиговый регистр и означает готовность передатчика к приему очередного символа для передачи. Если в управляющем регистре разрешены прерывания, то установка этого бита вызывает соответствующее прерывание до тех пор пока он не будет сброшен. Сброс бита осуществляется записью в регистр *UDR*. При сбросе МК этот бит установлен, показывая готовность передатчика.
FE (*Framing Error*) – устанавливается когда на месте стоп бита принимаемого символа оказывается ноль.
OR (*Overrun*) – устанавливается когда в регистре *UDR* имеется не прочитанный символ и в сдвиговом регистре принят очередной символ. Бит буферизирован, что означает, что его обновление происходит при чтении регистра *UDR*, бит сбрасывается при перезаписи принятых данных в *UDR*.

UCR (*UART Control RG*) – регистр управления устройством.

	7	6	5	4	3	2	1	0
UCR $0A ($2A)	*RxCIE*	*TxCIE*	*UDRIE*	*RXEN*	*TXEN*	*CHR*9	*RXB*8	*TXB*8
	R/W	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R*	*W*
Начал. знач.	0	0	0	0	0	0	1	0

RxCIE (*RX Complete Interrupt Enable*) – установка в «1» разрешает прерывание при окончании приема (по установке бита *RXC* статусного регистра *USR*).
TxCIE (*TX Complete Interrupt Enable*) – установка в «1» разрешает прерывание при окончании передачи (по установке бита *TXC* статусного регистра *USR*).
UDRIE (*UART Data RG Empty Interrupt Enable*) – установка в «1» разрешает прерывание при окончании передачи и пустоте регистра данных (по установке бита *UDRE* статусного регистра *USR*).
RXEN (*RX Enable*) – установка в «1» включает приемник. При сброшенном бите установка битов *RXC*, *OR*, *FE* невозможна. При сбросе этого бита влияние на эти разряды не оказывается.
TXEN (*TX Enable*) – установка в «1» включает передатчик. Если сброс бита произошел во время передачи символа, то передатчик не выключится до тех пор пока не будет передан текущий символ из сдвигового регистра и плюс все символы записанные в *UDR*.
CHR9 (9 *Bit Characters*) – установка в «1» устанавливает длину символа 9 бит плюс старт бит и стоп бит. Девятый символ принимается в разряде *RXB*8, а передается через разряд *TXB*8 управляющего регистра. Данный бит позволяет эмулировать дополнительный разряд данных или дополнительный стоп бит или бит четности.

UBRR (*UART Band Rate RG*) – регистр управления частотой приемопередатчика.

	7	6	5	4	3	2	1	0
UBRR $09 ($29)	b7	b6	b5	b4	b3	b2	b1	b0
	R/W	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

Частота приемопередатчика определяется на основании формулы и нижеследующей таблицы.
$BAUD=F_{CK}/(16(UBRR+1))$, где F_{CK} – частота кварца.

Baud Rate	1 MHz		%Error	1.8432 MHz		%Error	2 MHz		%Error	2.4576 MHz		%Error
2400	UBRR=	25	0.2	UBRR=	47	0.0	UBRR=	51	0.2	UBRR=	63	0.0
4800	UBRR=	12	0.2	UBRR=	23	0.0	UBRR=	25	0.2	UBRR=	31	0.0
9600	UBRR=	6	7.5	UBRR=	11	0.0	UBRR=	12	0.2	UBRR=	15	0.0
14400	UBRR=	3	7.8	UBRR=	7	0.0	UBRR=	8	3.7	UBRR=	10	3.1
19200	UBRR=	2	7.8	UBRR=	5	0.0	UBRR=	6	7.5	UBRR=	7	0.0
28800	UBRR=	1	7.8	UBRR=	3	0.0	UBRR=	3	7.8	UBRR=	4	6.3
38400	UBRR=	1	22.9	UBRR=	2	0.0	UBRR=	2	7.8	UBRR=	3	0.0
57600	UBRR=	0	7.8	UBRR=	1	0.0	UBRR=	1	7.8	UBRR=	2	12.5
76800	UBRR=	0	22.9	UBRR=	1	33.3	UBRR=	1	22.9	UBRR=	1	0.0
115200	UBRR=	0	84.3	UBRR=	0	0.0	UBRR=	0	7.8	UBRR=	0	25.0

Baud Rate	3.2768 MHz		%Error	3.6864 MHz		%Error	4 MHz		%Error	4.608 MHz		%Error
2400	UBRR=	84	0.4	UBRR=	95	0.0	UBRR=	103	0.2	UBRR=	119	0.0
4800	UBRR=	42	0.8	UBRR=	47	0.0	UBRR=	51	0.2	UBRR=	59	0.0
9600	UBRR=	20	1.6	UBRR=	23	0.0	UBRR=	25	0.2	UBRR=	29	0.0
14400	UBRR=	13	1.6	UBRR=	15	0.0	UBRR=	16	2.1	UBRR=	19	0.0
19200	UBRR=	10	3.1	UBRR=	11	0.0	UBRR=	12	0.2	UBRR=	14	0.0
28800	UBRR=	6	1.6	UBRR=	7	0.0	UBRR=	8	3.7	UBRR=	9	0.0
38400	UBRR=	4	6.3	UBRR=	5	0.0	UBRR=	6	7.5	UBRR=	7	6.7
57600	UBRR=	3	12.5	UBRR=	3	0.0	UBRR=	3	7.8	UBRR=	4	0.0
76800	UBRR=	2	12.5	UBRR=	2	0.0	UBRR=	2	7.8	UBRR=	3	6.7
115200	UBRR=	1	12.5	UBRR=	1	0.0	UBRR=	1	7.8	UBRR=	2	20.0

Baud Rate	7.3728 MHz		%Error	8 MHz		%Error	9.216 MHz		%Error	11.059 MHz		%Error
2400	UBRR=	191	0.0	UBRR=	207	0.2	UBRR=	239	0.0	UBRR=	287	-
4800	UBRR=	95	0.0	UBRR=	103	0.2	UBRR=	119	0.0	UBRR=	143	0.0
9600	UBRR=	47	0.0	UBRR=	51	0.2	UBRR=	59	0.0	UBRR=	71	0.0
14400	UBRR=	31	0.0	UBRR=	34	0.8	UBRR=	39	0.0	UBRR=	47	0.0
19200	UBRR=	23	0.0	UBRR=	25	0.2	UBRR=	29	0.0	UBRR=	35	0.0
28800	UBRR=	15	0.0	UBRR=	16	2.1	UBRR=	19	0.0	UBRR=	23	0.0
38400	UBRR=	11	0.0	UBRR=	12	0.2	UBRR=	14	0.0	UBRR=	17	0.0
57600	UBRR=	7	0.0	UBRR=	8	3.7	UBRR=	9	0.0	UBRR=	11	0.0
76800	UBRR=	5	0.0	UBRR=	6	7.5	UBRR=	7	6.7	UBRR=	8	0.0
115200	UBRR=	3	0.0	UBRR=	3	7.8	UBRR=	4	0.0	UBRR=	5	0.0

3.3.7 *EEPROM* – энергонезависимая память данных

Разработчики МК часто встраивают в свои изделия энергонезависимую память *EEPROM* (*Electrically Erasable Programmable Read-Only Memory*). Это позволяет сохранить данные на период пропадания питания, что характерно устройствам с батарейным или комбинированным питанием. Также общей чертой обращения к *EEPROM* является то, что процедуры записи и чтения требуют посылки последовательности определенных команд, а сам процесс обращения занимает несколько миллисекунд, требуя полной остановки выполнения программы на этот период. Это обеспечивает сохранность данных от случайной записи, стирания при различных сбоях – отказ генератора, снижение напряжения питания и т.д.

Программно *EEPROM* доступно с помощью трех адресов: регистр адреса (*EEPROM Address RG*), регистр данных (*EEPROM Data RG*) и регистр управления (*EEPROM Control RG*).

	7	6	5	4	3	2	1	0
EEAR $1E ($3E)	-	b6	b5	b4	b3	b2	b1	b0
	R	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
EEDR $1D ($3D)	b7	b6	b5	b4	b3	b2	b1	b0
	R/W	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
EECR $1C ($3C)	-	-	-	-	-	*EEMWE*	*EEWE*	*EERE*
	R	*R*	*R*	*R*	*R*	*R/W*	*R/W*	*R/W*
Начал. знач.	0	0	0	0	0	0	0	0

EEMWE (*EEPROM Master Write Enable*) – разрешает запись в память, если этот бит равен нулю, то запись не возможна (после программной установки этого бита, МК автоматически его сбросит после четырех тактов МК).
EEWE (*EEPROM Write Enable*) – назначение это бита отождествлено со стробом записи *EEPROM*, который посылается после корректной установки адреса и данных, при этом бит *EEMWE* должен быть установлен в «1».
EERE (*EEPROM Read Enable*) - назначение это бита отождествлено со стробом чтения *EEPROM*, который посылается после корректной установки адреса и данных; при этом процессор останавливается на 4 такта перед выполнением следующей команды; в конце операции бит автоматически сбрасывается; попытка чтения во время записи приведет к прекращению цикла записи и неопределенности данных при чтении.

Таким образом, процедура записи в энергонезависимую память следующая:

- дождаться когда бит *EEMWE* управляющего регистра *EECR* сбросится в ноль;
- записать адрес в регистр *EEAR* (если нужно), записать данные в регистр *EEDR* (если нужно);
- установить в «1» бит *EEMWE* управляющего регистра *EECR*, при этом бит *EEWE* должен быть сброшен;
- в течение 4 тактов после установки *EEMWE* послать команду записи установкой в «1» бита *EEWE*.

Доступ при записи в память занимает примерно 2.5 мс при *Vcc*=5 В и 4 мс при *Vcc*=2.7 В. После успешной записи бит *EEWE* автоматически установится в ноль. После установки бита *EEWE* процессор останавливается на два такта перед выполнением следующей инструкции. Особое внимание при записи в *EEPROM* следует уделять прерываниям, которые могут прервать цикл записи и привести к потере данных.

Сводная таблица портов

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C
$3E ($5E)	Reserved								
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
$3C ($5C)	Reserved								
$3B ($5B)	GIMSK	INT1	INT0	–	–	–	–	–	–
$3A ($5A)	GIFR	INTF1	INTF0						
$39 ($59)	TIMSK	TOIE1	OCIE1A	–	–	TICIE1	–	TOIE0	–
$38 ($58)	TIFR	TOV1	OCF1A	–	–	ICF1	–	TOV0	–
$37 ($57)	Reserved								
$36 ($56)	Reserved								
$35 ($55)	MCUCR	–	–	SE	SM	ISC11	ISC10	ISC01	ISC00
$34 ($54)	Reserved								
$33 ($53)	TCCR0	–	–	–	–	–	CS02	CS01	CS00
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)							
$31 ($51)	Reserved								
$30 ($50)	Reserved								
$2F ($4F)	TCCR1A	COM1A1	COM1A0	–	–	–	–	PWM11	PWM10
$2E ($4E)	TCCR1B	ICNC1	ICES1	.	–	CTC1	CS12	CS11	CS10
$2D ($4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte							
$2C ($4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte							
$2B ($4B)	OCR1AH	Timer/Counter1 – Compare Register High Byte							
$2A ($4A)	OCR1AL	Timer/Counter1 – Compare Register Low Byte							
$29 ($49)	Reserved								
$28 ($48)	Reserved								
$27 ($47)	Reserved								
$26 ($46)	Reserved								
$25 ($45)	ICR1H	Timer/Counter1 – Input Capture Register High Byte							
$24 ($44)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte							
$23 ($43)	Reserved								
$22 ($42)	Reserved								
$21 ($41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0
$20 ($40)	Reserved								
$1F ($3F)	Reserved								
$1E ($3E)	EEAR	–	EEPROM Address Register						
$1D ($3D)	EEDR	EEPROM Data Register							
$1C ($3C)	EECR	–	–	–	–	–	EEMWE	EEWE	EERE
$1B ($3B)	Reserved								
$1A ($3A)	Reserved								
$19 ($39)	Reserved								
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
$15 ($35)	Reserved								
$14 ($34)	Reserved								
$13 ($33)	Reserved								
$12 ($32)	PORTD	–	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
$11 ($31)	DDRD	–	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
$10 ($30)	PIND	–	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
…	Reserved								
$0C ($2C)	UDR	UART I/O Data Register							
$0B ($2B)	USR	RXC	TXC	UDRE	FE	OR	–	–	–
$0A ($2A)	UCR	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8
$09 ($29)	UBRR	UART Baud Rate Register							
$08 ($28)	ACSR	ACD	–	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
…	Reserved								
$00 ($20)	Reserved								

3.4 ПРОГРАММИРОВАНИЕ МК И ЕГО ИДЕНТИФИКАЦИЯ

В настоящее время практически все МК имеют в своем составе управляющие биты конфигурации, которые называются обычно *Lock*, *Fuse* битами. Назначением этих бит является конфигурирование МК, например, разрешение/запрещение программирования памяти и т.п.

В серии МК *AT*90 *Lock* биты предназначены для управления возможностью записи во *Flash* и *EEPROM*, а также возможностью чтения программы из вне. При этом восстановление значений этих битов в исходное состояние (все разрешено) возможно только операцией *erase chip* – полным удалением программы и данных.

МК *AT*90*S*2313 имеются два *Fuse* бита: *SPIEN*, *FSTRT*. Когда значение первого равно нулю, программирование по последовательному интерфейсу и загрузка данных разрешены (по умолчанию *SPIEN*=0). Когда *FSTRT*=0 включается режим ускоренного старта – за счет сокращения времени ожидания окончания переходных процессов при включении генератора (значение по умолчанию *FSTRT*=1) – см. рис.3.7.

Также *AT*90*S*2313 как и другие содержит нестираемые биты опознавания: *signature bits*, чтение которых позволяет идентифицировать марку МК, его производителя, определить объем *Flash* памяти и получить некоторые другие сведения. Отметим, что с помощью *Lock* битов чтение этих бит может быть запрещено.

Программирование *Flash* возможно двумя путями: по параллельному или последовательному интерфейсу. В исходном состоянии вся память готова к записи (каждая ячейка содержит 0FFh). При этом параллельное программирование требует наличия +12 В, которое используется только для перевода чипа в режим параллельного программирования. В документации на каждый МК имеются подробные алгоритмы программирования и схемы включения.