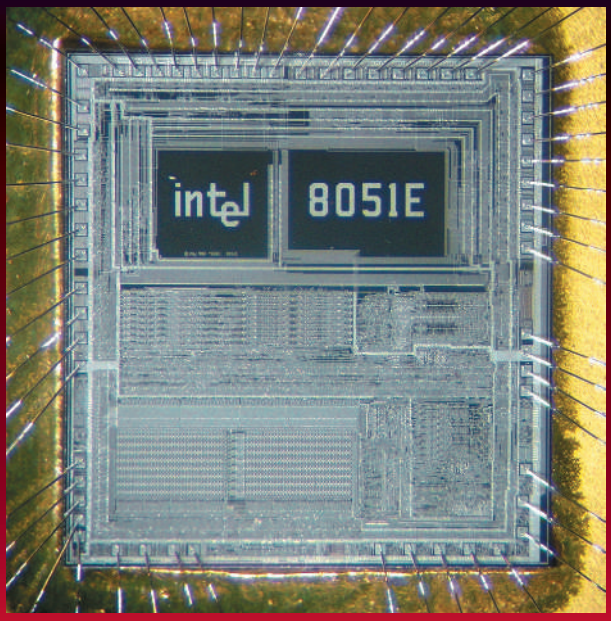
Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина
Физико-технологический институт

Е. В. МОИСЕЙКИН

МИКРОКОНТРОЛЛЕРЫ СЕМЕЙСТВА MCS-51 ТЕОРИЯ И ПРАКТИКА

Учебно-методическое пособие



Министерство образования и науки Российской Федерации
Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина

Е. В. Моисейкин

Микроконтроллеры семейства MCS-51
Теория и практика

Учебно-методическое пособие

Рекомендовано методическим советом
Уральского федерального университета
для студентов вуза, обучающихся
по направлениям подготовки
12.03.01 — Приборостроение,
11.03.04 — Электроника и наноэлектроника,
27.03.01 — Стандартизация и метрология
в приборостроении

Екатеринбург
Издательство Уральского университета
2017

УДК 621.3.049.77(075.8)
ББК 32.844.150.2я73
М74

Рецензенты:
доц., канд. физ.-мат. наук И. И. Горбачев (Уральский государственный экономический университет);
канд. физ.-мат. наук В. С. Черемных (ООО «Прософт-Системы»)

Научный редактор — проф., д-р физ.-мат. наук И. И. Мильман

На обложке изображение с сайта http://www.cpu-galaxy.at/CPU/Ram%20Rom%20Eprom/Microcontroller/Intel%208051%20section-Dateien/C8051E_Core_HiRes.jpg

Моисейкин, Е. В.

М74 Микроконтроллеры семейства MCS-51. Теория и практика : учебно-методическое пособие / Е. В. Моисейкин. — Екатеринбург : Изд-во Урал. ун-та, 2017. — 144 с.

ISBN 978-5-7996-2167-4

В пособии рассмотрены общие вопросы организации микроконтроллеров семейства MCS-51, детально показана организация и функционирование блоков, система команд. Представлено 6 лабораторных работ, посвященных изучению особенностей и программированию на языке ассемблера микроконтроллеров семейства MCS-51, рассмотрен лабораторный стенд SDK1.1 на базе микроконтроллера ADuC812 компании Analog Devices. Приведены необходимые сведения по работе в системе моделирования однокристальных микроконтроллеров Single-Chip Machine.

Пособие предназначено для студентов, обучающихся по дисциплинам «Микропроцессорная техника», «Микропроцессоры в приборах неразрушающего контроля», «Микропроцессоры приборных комплексов», «Системы автоматического управления».

Библиогр.: 9 назв. Рис. 29. Табл. 26. Прил. 4.

УДК 621.3.049.77(075.8)
ББК 32.844.150.2я73

ISBN 978-5-7996-2167-4

РАЗДЕЛ А

1. Общие сведения о микроконтроллерах семейства MCS-51

В 1980 году компанией Intel был выпущен микроконтроллер Intel 8051, являющийся первым представителем семейства MCS-51. Технологические особенности данной микросхемы были очень сложны для своего времени вследствие того, что кристалл содержал около 128 тысяч транзисторов, тогда как 16-разрядный микропроцессор Intel 8086, используемый в ПК того времени, включал около 30 тысяч полупроводниковых элементов.

Микроконтроллеры семейства MCS-51 представляют собой функционально завершенные 8-разрядные однокристальные микроЭВМ гарвардской архитектуры, содержащие все необходимые узлы для работы в автономном режиме. На основе таких микросхем могут быть реализованы алгоритмы управления различных устройств.

В настоящее время некоторые компании продолжают производство микроконтроллеров семейства MCS-51. За всю историю существования этого семейства различными производителями было выпущено несколько сотен модификаций микроконтроллеров, от простых 20-выводных устройств с одним таймером и 1 Кбайт памяти программ до сложнейших 100-выводных кристаллов с 24-разрядными АЦП и 12-разрядными ЦАП, массивами таймер-счетчиков, аппаратными 16-разрядными умножителями, 64 Кбайт программной FLASH-памяти на кристалле, системами управления приводами, USB интерфейсами и т. п.

Все микроконтроллеры из семейства MCS-51 имеют общую систему команд. Наличие дополнительного оборудования влияет толь-

ко на количество регистров специальных функций, с помощью которых осуществляется настройка параметров работы соответствующих встроенных узлов.

Микроконтроллеры данного семейства выпускаются в корпусах PLCC, DIP и QFP. Рабочие температурные диапазоны производимых микросхем:

- коммерческий (0...+70 °C);
- расширенный (–40...+85 °C);
- для военного использования (–55...+125 °C).

1.1. Структура микроконтроллеров MCS-51

Все микросхемы семейства обладают аналогичной архитектурой и имеют целый ряд общих узлов (рис. 1) [1, 2]:

- 8-разрядный центральный процессор (ЦП), ориентированный на управление исполнительными устройствами. ЦП имеет встроенную схему аппаратного умножения и деления 8-разрядных чисел. Система команд микроконтроллера содержит большое количество операций для работы с прямоадресуемыми битами, таким образом, ядро можно назвать «булевым процессором»;
- внутренняя память программ масочного или репрограммируемого типа может иметь объем 4–64 Кбайт в зависимости от модификации микроконтроллера, в некоторых кристаллах может отсутствовать, в последнем случае используется только внешняя память программ;
- резидентная память данных объемом не менее 128 байт, предназначена для организации регистровых банков, стека и хранения пользовательских данных;
- не менее четырех двунаправленных побитно настраиваемых восьмиразрядных портов ввода-вывода;
- два 16-битных программно настраиваемых многорежимных таймер-счетчика, используемых для подсчета внешних событий, организации временных задержек и синхронизации последовательного асинхронного порта;
- двунаправленный дуплексный последовательный коммуникационный порт, применяемый для организации обмена информацией между микроконтроллером и внешними устройствами;

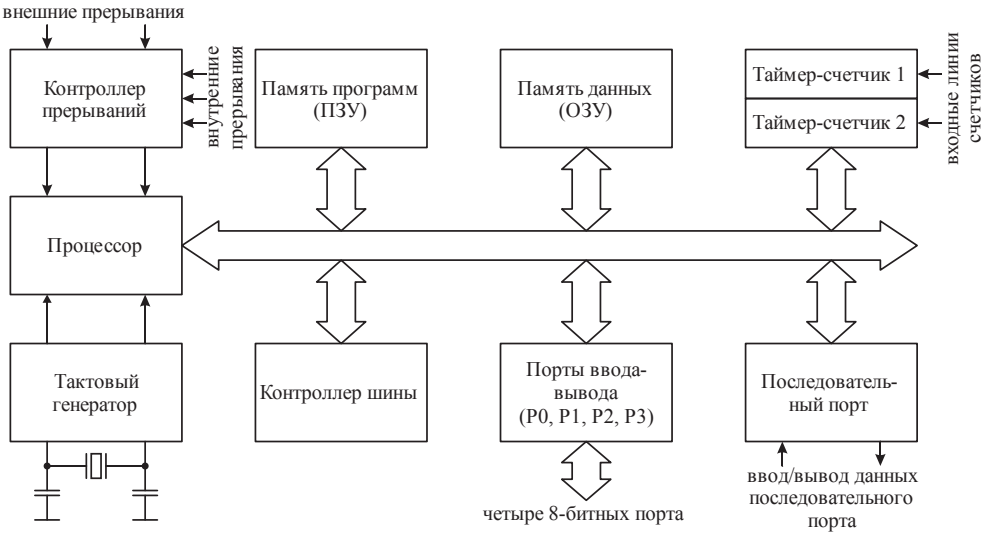


Рис. 1. Структурная схема микроконтроллера MCS-51

- система прерываний, имеющая два уровня приоритета для каждого прерывания, поддерживающая не менее 5 векторов прерываний от внутренних и внешних источников;
- встроенный тактовый генератор.

Основные технические характеристики центрального процессора микроконтроллеров семейства MCS-51:

- разрядность арифметико-логического устройства (АЛУ) — 8 бит;
- число выполняемых команд — 111;
- длина команд — 1, 2 или 3 байт;
- число регистров общего назначения (РОН) — 32;
- число прямоадресуемых битовых переменных — 128;
- число прямоадресуемых битов в области регистров специальных функций — 128;
- максимальный объем памяти программ — 64 Кбайт;
- максимальный объем памяти данных — 64 Кбайт;
- максимальный объем внутренней памяти данных — 256 байт;
- время выполнения команд при тактовой частоте 12 МГц —
  - сложение — 1 мкс;
  - пересылки «регистр — внешняя память данных» — 2 мкс;
  - умножение-деление — 4 мкс;
  - методы адресации операнда — регистровый, косвенный, прямой, непосредственный.

Различные представители семейства микроконтроллеров MCS-51 включают в себя широкий набор дополнительных средств для эффективной обработки асинхронных событий, измерения частот и временных интервалов, обработки аналоговых сигналов, управления двигателями, организации мультипроцессорных систем.

Связь основных блоков и узлов микроконтроллеров семейства MCS-51 (рис. 2), таких как резидентная память, АЛУ, блок регистров специальных функций, устройство управления, порты ввода-вывода, осуществлена с помощью внутренней двунаправленной 8-битной шины.

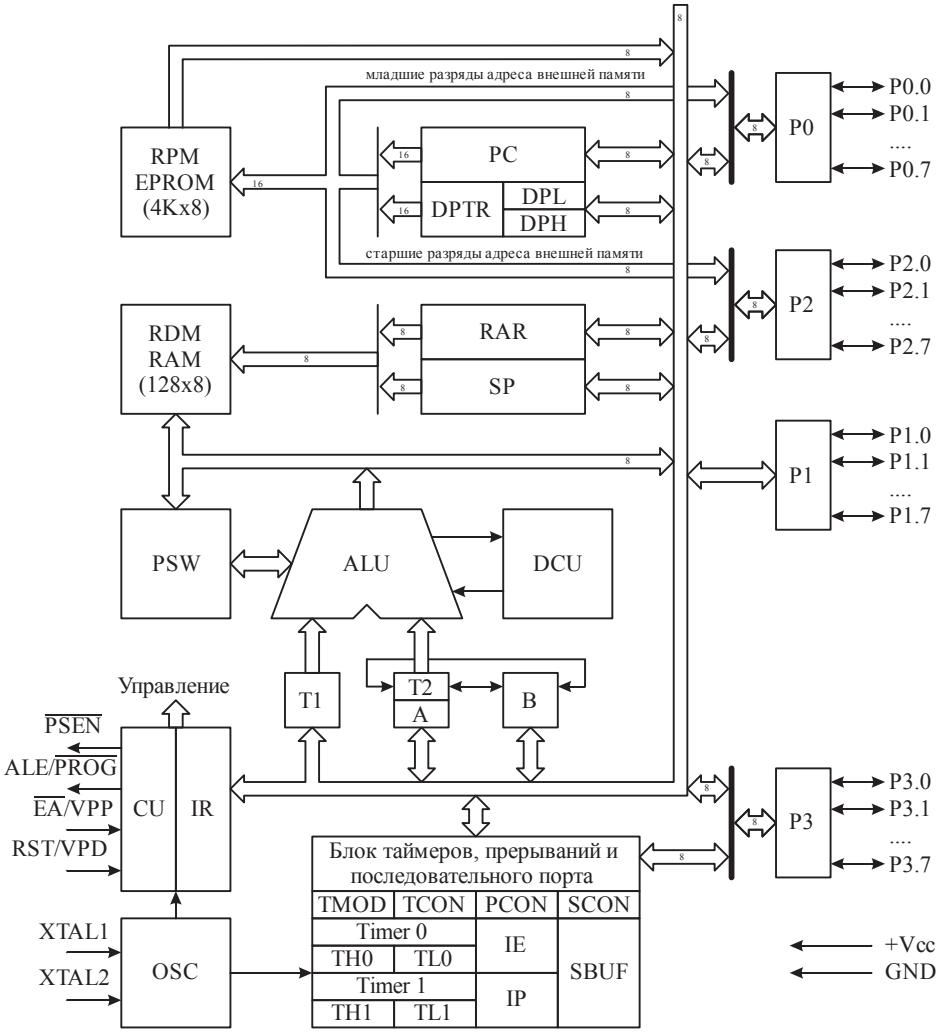


Рис. 2. Подробная структурная схема микроконтроллера Intel 8051

Ниже рассмотрены основные элементы структуры микроконтроллеров семейства MCS-51 на примере микросхемы Intel 8051:

- DCU (*Decimal Correction Unit*) — схема десятичной коррекции;
- PSW (*Program Status Word*) — схема формирования признаков результата и регистр признаков;
- RPM EPROM (*Resident Program Memory EPROM*) — встроенная (внутренняя) память программ (ПЗУ) объемом 4 Кбайт;
- RDM RAM (*Resident Data Memory RAM*) — встроенная (внутренняя) память данных (ОЗУ) объемом 128 байт;
- OSC — схема синхронизации;
- CU (*Control Unit*) — схема управления;
- IR (*Instruction Register*) — программно недоступный регистр для хранения кода операции;
- T1 (*Temp 1*), T2 (*Temp 2*) — программно недоступные регистры, служат для временного хранения операндов;
- B — восьмиразрядный регистр, используемый для операций умножения и деления;
- SP (*Stack Pointer*) — указатель стека;
- PC (*Program Counter*) — счетчик команд;
- RAR (*RAM Address Register*) — регистр-указатель данных;
- DPTR (*Data Pointer Register*) — 16-битный регистр-указатель адреса. Является совокупностью двух 8-разрядных независимых регистров DPL (младший байт DPTR) и DPH (старший байт DPTR).

В микропроцессорах, основанных на аккумуляторной архитектуре, большинство команд использует аккумулятор как источник операнда и (или) приемник результата с помощью неявной адресации. ЦП Intel 8051 основан на аккумуляторной архитектуре, но при выполнении части команд аккумулятор не используется. Например, некоторые варианты команды MOV позволяют передавать данные из любой ячейки ОЗУ в любой регистр или другую ячейку памяти, записывать константы в любой регистр и т. д. Основные логические операции могут быть произведены над ячейками памяти данных без использования аккумулятора. Кроме того, переменные могут быть инкрементированы, декрементированы и проверены без использования аккумулятора. Работа с битовыми переменными, такими как флаги, управляющие биты, осуществляется с использованием «булевого» аккумулятора, но без применения основного 8-разрядного аккумулятора.

1.2. Арифметико-логическое устройство

АЛУ представляет собой параллельное восьмиразрядное устройство, обеспечивающее выполнение арифметических операций: сложение, вычитание, умножение и деление; логических операций: И, ИЛИ, исключающее ИЛИ; операций логического сдвига, обнуления, установки, инвертирования и др.

Для выполнения некоторых команд, таких как инкрементирование ячеек памяти и регистров, автоматическое вычисление следующего адреса выполняемой команды и т. д., в АЛУ используется простейшая операция сложения. Кроме того, декрементирование регистров и сравнение переменных осуществляется в АЛУ с помощью простейшей операции вычитания.

ЦП микроконтроллеров семейства MCS-51 является типичным представителем процессоров CISC-архитектуры. Одной из характеристик таких процессоров является наличие сложных (многотактных) команд. Для рассматриваемого процессора один машинный цикл составляет 12 тактов, в зависимости от сложности команды ее время выполнения может составлять 1, 2 или 4 машинных цикла. Таким образом, простейшие операции образуют группы для выполнения более сложных операций. Другими словами, для выполнения сложных команд в АЛУ реализуется механизм каскадного исполнения простейших операций. Например, при выполнении команды DJNZ, предназначенной для условного перехода на указанный адрес по результату сравнения значения ячейки внутренней памяти программ с нулем, в АЛУ трижды инкрементируется счетчик команд, дважды производится чтение из внутренней памяти данных, выполняется арифметическое сравнение двух переменных, формируется 16-битный адрес перехода и принимается решение о том, делать или не делать переход по указанному адресу. Все перечисленные операции выполняются в АЛУ за 2 машинных цикла, что при частоте 12 МГц составляет 2 мкс.

Важной особенностью АЛУ является его способность оперировать не только байтами, но и битами. Отдельные программно-доступные биты могут быть установлены, сброшены, инвертированы, переданы, проверены и использованы в логических операциях. Способность АЛУ оперировать битами широко применяется при разработке программ управления различными объектами вследствие того, что все

параллельные порты ввода-вывода отображены в битовом пространстве данных. Это позволяет управлять каждым выводом микросхемы в отдельности, независимо от состояния других. Реализация подобных механизмов взаимодействия средствами обычных микропроцессоров вызывает определенные трудности.

Таким образом, АЛУ может оперировать четырьмя типами информационных объектов: булевскими (1 бит), цифровыми (4 бит), байтными (8 бит) и адресными (16 бит). В АЛУ выполняется 51 различная операция пересылки или преобразования этих данных. Комбинированием операций и режимов адресации (7 для данных и 4 для адресов) базовое число команд 111 увеличивается до 255 из 256 возможных при однобайтном коде операции.

1.3. Организация памяти данных, памяти программ и регистров

По способу организации пространства памяти микропроцессорные системы подразделяются на два основных типа архитектур: архитектуру фон Неймана (или принстонская архитектура) и гарвардскую архитектуру. Микроконтроллеры рассматриваемого семейства построены по гарвардской архитектуре, в этом случае память программ и память данных представляют собой независимые устройства, которые физически и логически разделены, каждое имеет собственное адресное пространство, использует собственные линии связи с АЛУ и различные управляющие сигналы.

Микроконтроллеры семейства MCS-51 позволяют использовать и управлять пятью (частично пересекающимися) адресными пространствами памяти, четыре из которых относятся к памяти данных (рис. 3) [2]:

- RSEG — пространство регистров (4×8 байт);
- DSEG — пространство внутренней (или резидентной) памяти данных (РПД) (256 байт);
- BSEG — битовое пространство данных (256 бит);
- XSEG — пространство внешней памяти данных (ВПД) (до 64 Кбайт);
- CSEG — пространство внутренней и (или) внешней памяти программ (до 64 Кбайт).

Рис. 3. Организация внутренней памяти данных (ОЗУ), пространство DSEG

Пространство DSEG предназначено для хранения данных, физически совмещено с частично пересекающимися пространствами RSEG и BSEG. Такими образом, можно использовать наиболее удобный доступ к одной и той же информации, как к ячейке памяти или регистру, битовому полю или регистру специальных функций и т. д.

Регистры общего назначения (РОН) организованы в четыре банка по 8 регистров. Физически банки расположены в начальной области DSEG, начиная с адреса 00 по 1F.

Кроме того, на пространство DSEG отображено пространство регистров специальных функций (SFR), которое содержит системные регистры (такие как 16-разрядный программный счетчик PC, регистр косвенного адреса DPTR, 8-разрядные аккумуляторы A и B, указатель стека SP и регистр слова состояния программы PSW), регистры для управления портами ввода-вывода, таймерами и другим периферийным оборудованием.

Пространство регистров представлено четырьмя банками регистров по 8 РОН (регистр общего назначения) в каждом, а также 16-разрядными программным счетчиком PC и регистром косвенного адреса DPTR, 8-разрядными аккумуляторами A и B, указателем стека SP и регистром PSW.

Физическое взаимодействие с внешней памятью данных (пространство XSEG) и внешней памятью программ (пространство CSEG) осуществляется при помощи сигналов:

- RD (*Read Data for External Memory* — чтение из внешней памяти данных) — при низком уровне сигнала RD выполняется чтение данных из XSEG;
- WR (*Write Data for External Memory* — запись во внешнюю память данных) — при низком уровне сигнала WR выполняется запись данных в XSEG;
- PSEN (*Programm Store Enable* — чтение из внешней памяти программ) — при низком уровне сигнала PSEN выполняется чтение команды из CSEG;
- ALE (*Address Lath Enable* — разрешение передачи адреса) — предназначен для фиксации младшей части адреса во внешнем регистре.

При высоком уровне управляющих сигналов выходы (ответственные за передачу данных) микросхем соответствующей памяти должны перейти в третье состояние.

1.3.1. Память программ

Память программ предназначена для хранения программ, выполняемых микроконтроллером. При выполнении тех команд, в которых принимают участие непосредственные операнды, АЛУ также обращается к памяти программ для загрузки соответствующих данных. Кроме того, система команд позволяет использовать память программ для хранения массивов. В зависимости от типа команды адрес ячейки памяти программ, из которой необходимо считать данные, может находиться в счетчике команд (PC) или регистре указателя данных (DPTR).

Наличие 16-разрядной внешней адресной шины позволяет обращаться к внешней памяти программ объемом до 64 Кбайт. Базовый микроконтроллер имеет совмещенную шину адреса данных для работы с внешней памятью. Отключение внутренней памяти программ осуществляется путем подачи на вывод EA логического уровня 0 (рис. 4). На данном примере показано распределение пространства памяти программ при объеме встроенного ПЗУ 4 Кбайт.

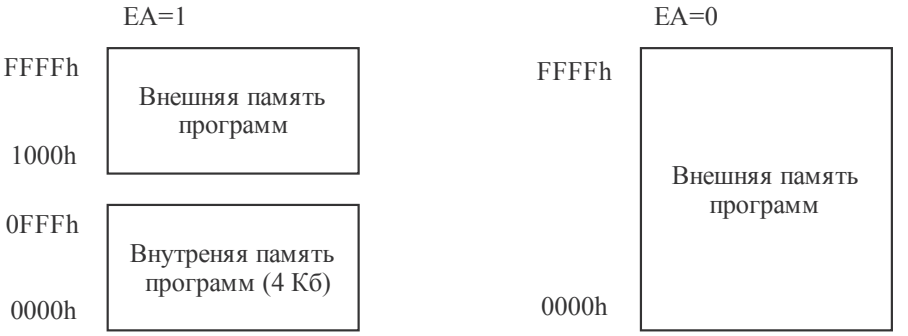


Рис. 4. Организация памяти программ в пространстве CSEG

С точки зрения программиста внешняя и внутренняя память программ представляют единое адресное пространство объемом 64 Кбайт.

1.3.2. Память данных

Внутренняя двунаправленная шина, связывающая основные узлы микроконтроллера, в том числе и резидентную память данных, является 8-разрядной, что позволяет адресовать до 256 ячеек. Однако объем встроенной памяти данных составляет 128 байт с адресами 00h–7Fh, а адресное пространство 80h–FFh предназначено для регистров специальных функций.

Первые 32 байта встроенной памяти данных с адресами 00h–1Fh, используются для 4 банков регистров общего назначения. Они обозначаются как банк 0 — банк 3, каждый из которых состоит из 8 регистров R0–R7 (см. рис. 3). В любой момент времени доступен только один банк регистров, номер активного банка указывается в слове состояния программы (PSW, см. с. 16). В некоторых командах регистры R0, R1 каждого банка могут быть использованы в качестве указателей данных.

Оставшееся пространство обычно используется для размещения стека, системных и пользовательских данных. Возможно два способа адресации к встроенной памяти данных: прямая адресация и косвенно-регистровая (регистры R0, R1).

Наличие 16-разрядной внешней адресной шины позволяет обращаться к внешней памяти данных объемом до 64 Кбайт. Для этого используется косвенно-регистровая адресация при помощи регистра DPTR.

Команды для работы с отдельными битами могут адресоваться к каждому разряду определенных ячеек внутренней памяти данных (табл. 1), при этом адрес прямоадресуемых битов можно записать либо в виде (Адрес байта). (Разряд), например 27h.2, либо в виде абсолютного битового адреса, например 3Ah. Адресуемые биты регистров специальных функций приведены в табл. 2 [2].

Таблица 1

Битовое пространство данных

Адрес байта	Адреса битов по разрядам							
	D7	D6	D5	D4	D3	D2	D1	D0
20h	07	06	05	04	03	02	01	00
21h	0F	0E	0D	0C	0B	0A	09	08
22h	17	16	15	14	13	12	11	10
23h	1F	1E	1D	1C	1B	1A	19	18
24h	27	26	25	24	23	22	21	20
25h	2F	2E	2D	2C	2B	2A	29	28
26h	37	36	35	34	33	32	31	30
27h	3F	3E	3D	3C	3B	3A	39	38
28h	47	46	45	44	43	42	41	40
29h	4F	4E	4D	4C	4B	4A	49	48
2Ah	57	56	55	54	53	52	51	50
2Bh	5F	5E	5D	5C	5B	5A	59	58
2Ch	67	66	65	64	63	62	61	60
2Dh	6F	6E	6D	6C	6B	6A	69	68
2Eh	77	76	75	74	73	72	71	70
2Fh	7F	7E	7D	7C	7B	7A	79	78

Таблица 2

Битовое пространство регистров

Адрес байта	Адреса битов по разрядам								Регистр
	D7	D6	D5	D4	D3	D2	D1	D0	
80h	87	86	85	84	83	82	81	80	P0
88h	8F	8E	8D	8C	8B	8A	89	88	TCON
90h	97	96	95	94	93	92	91	90	P1
98h	9F	9E	9D	9C	9B	9A	99	98	SCON
A0h	A7	A6	A5	A4	A3	A2	A1	A0	P2
A8h	AF			AC	AB	AA	A9	A8	IE
B0h	B7	B6	B5	B4	B3	B2	B1	B0	P3
B8h				BC	BB	BA	B9	B8	IP
D0h	D7	D6	D5	D4	D3	D2	D1	D0	PSW
E0h	E7	E6	E5	E4	E3	E2	E1	E0	A
F0h	F7	F6	F5	F4	F3	F2	F1	F0	B

1.3.3. Регистры специальных функций

Регистры специальных функций можно разделить на две группы: первая содержит регистры, предназначенные для организации управления встроенным периферийным оборудованием, а вторая — регистры, используемые для выполнения команд. Адреса, по которым расположены регистры специальных функций SFR (*Special Function Register*) базового микроконтроллера семейства MCS-51, приведены в табл. 3.

Таблица 3

Размещение регистров специальных функций в пространстве SFR

Адрес	Символ	Наименование
0E0H	*ACC	Аккумулятор (Accumulator)
0F0H	*B	Регистр-расширитель аккумулятора (Multiplication Register)
0D0H	*PSW	Слово состояния программы (Program Status Word)
080H	*P0	Порт 0 (SFR P0)
090H	*P1	Порт 1 (SFR P1)
0A0H	*P2	Порт 2 (SFR P2)
0B0H	*P3	Порт 3 (SFR P3)
081H	SP	Регистр-указатель стека (Stack Pointer)
083H	DPH	Старший байт регистра-указателя данных DPTR (Data Pointer High)
082H	DPL	Младший байт регистра-указателя данных DPTR (Data Pointer Low)
08CH	TH0	Старший байт таймера 0 (Timer High 0)
08AH	TL0	Младший байт таймера 0 (Timer Low 0)
08DH	TH1	Старший байт таймера 1 (Timer High 1)
08BH	TL1	Младший байт таймера 1 (Timer Low 1)
089H	TMOD	Регистр режимов таймер-счетчиков (Timer/Counter Mode Control Register)
088H	*TCON	Регистр управления статуса таймеров (Timer/Counter Control Register)
0B8H	*IP	Регистр приоритетов (Interrupt Priority Control Register)
0A8H	*IE	Регистр маски прерывания (Interrupt Enable Register)
087H	PCON	Регистр управления мощностью (Power Control Register)
098H	*SCON	Регистр управления приемопередатчиком (Serial Port Control Register)
099H	SBUF	Буфер приемопередатчика (Serial Data Buffer)

Примечание. Регистры, символ которых отмечен знаком *, допускают адресацию своих отдельных битов при использовании команд из группы команд операций над битами.

Адресное пространство, отведенное для регистров специальных функций, составляет 128 байт в диапазоне 80h–0FFh. Свободные адреса, не занятые регистрами, физически отсутствуют, при обращении к ним можно прочитать лишь код команды возврата.

Основные регистры специальных функций:

- регистры параллельных портов P0–P3 — служат для ввода-вывода цифровой информации через соответствующие выводы микросхемы;
- регистровые пары TH0, TL0 и TH1, TL1 — предназначены для хранения текущих значений двух программно управляемых 16-битных таймер-счетчиков;
- регистр TMOD — определяет режимы работы таймер-счетчиков;
- TCON — управляет работой таймер-счетчиков, а также используется для настройки прерываний от внешних источников;
- PCON — предназначен для управления режимами энергопотребления микроконтроллера, здесь присутствует бит управления удвоением скорости последовательного порта;
- IP и IE — применяются для задания приоритетов и выбора источников системы прерываний микроконтроллера;
- SBUF и SCON — управляют работой приемопередатчика последовательного порта;
- регистр-указатель стека SP — предназначен для организации системного стека. Стек располагается в области резидентной памяти данных и теоретически может иметь глубину до 256 байт. Реально глубина стека ограничена объемом резидентной (внутренней) памяти данных (РПД), т. е. равна 128. У микроконтроллеров семейства MCS-51 стек «растет вверх», т. е. при выполнении команды PUSH (запись в стек) или CALL (вызов подпрограммы) сначала значение регистра SP инкрементируется и затем информации записывается в стек. Соответственно при чтении данных из стека сначала производится считывание информации, а потом декремент регистра SP. При инициализации микроконтроллера после включения или в результате его перезапуска в регистр SP заносится код 07H. Таким образом, запись первого элемента стека будет производиться в ячейку резидентной памяти данных с адресом 08h;
- указатель данных DPTR — используется для обращения к пространству внешней памяти данных и при пересылке констант из внешней памяти программ в аккумулятор. Кроме того, содер-

жимое DPTR используется в качестве смещения в команде перехода. 16-битный регистр DPTR состоит из двух независимых 8-битных регистров DPL и DPH;
- аккумулятор — является источником операнда и местом фиксации результата при выполнении арифметических, логических операций и ряда операций передачи данных. Некоторые типы команд, например, операции сдвигов, проверка на нуль, могут быть выполнены только с использованием аккумулятора;
- регистр B — предназначен для выполнения операций умножения и деления, в которых используется как операнд и как приемник результата. В других командах он может применяться как дополнительный регистр;
- регистр PSW — предназначен для фиксации флагов, сформированных при выполнении ряда команд в АЛУ.

1.3.4. Регистр признаков

Регистр признаков (PSW) предназначен для хранения и анализа признаков результата операции и служебных признаков:

PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
C	AC	F0	RS1	RS0	OV		P

Поля регистра PSW:
- P (PSW.0) — флаг паритета (четности). Устанавливается и сбрасывается аппаратно в каждом цикле команды и дополняет количество единичных битов в аккумуляторе до четного;
- OV (PSW.2) — флаг переполнения. Устанавливается и сбрасывается аппаратным способом при выполнении арифметических операций. Фиксирует арифметическое переполнение при операциях над целыми числами со знаком, что позволяет выполнять математические операции в дополнительных кодах;
- RS0–RS1 (PSW.3, PSW.4) — биты выбора используемого банка регистров. Изменяются только программой, их значение указывает активный банк регистров общего назначения (табл. 4);

Таблица 4

Банки регистров

RS0	RS1	Банк	Границы адресов ОЗУ
0	0	0	00h–07h
1	0	1	08h–0Fh
0	1	2	10h–17h
1	1	3	18h–1Fh

- F0 (PSW.5) — флаг пользователя, может использоваться как дополнительный бит программой пользователя, где возможны изменения и проверка этого бита;
- AC (PSW.6) — флаг вспомогательного (межтетрадного) переноса. Модифицируется только аппаратным путем при выполнении команд сложения и вычитания. Сигнализирует о переносе или заеме в бите 3 аккумулятора;
- C (PSW.7) — флаг переноса. Устанавливается и сбрасывается как аппаратным, так и программным способом. При выполнении множества операций, включая сложение, вычитание и сдвиги, флаг C может оказывать влияние на результат и (или) сигнализировать о наличии переноса. Кроме того, в командах для работы с отдельными битами выполняет функции «булева аккумулятора».

1.4. Таймер-счетчики микроконтроллеров семейства MCS-51

Базовые модели микроконтроллеров семейства MCS-51 содержат два программируемых многорежимных таймер-счетчика (0 и 1), предназначенных для подсчета внешних событий (выводы T0 и T1), организации программно управляемых временных задержек и измерения временных интервалов [2]. Кроме того, таймер 1 применяется для определения скорости передачи последовательного порта.

Таймер-счетчик может работать в режиме таймера или в режиме счетчика. В первом случае ведется подсчет тактов деленной системной частоты (определенный промежуток времени) и при переполнении выдается запрос прерывания. В каждом машинном цикле длительностью 12 тактов регистр таймера инкрементируется только один раз, поэтому скорость счета таймера равна $f_{OSC}/12$.

В режиме счетчика ведется подсчет количества поступивших импульсов на вход микросхемы, причем идентификация импульса производится по заднему фронту. При переполнении таймерного регистра таймер-счетчика выдается запрос прерывания. Распознавание спада внешнего сигнала занимает 24 периода тактовой частоты (2 машинных цикла), поэтому максимальная скорость счета равна $f_{OSC}/24$.

Управление режимами работы таймер-счетчиков и организация их взаимодействия с системой прерываний обеспечивается двумя регистрами специальных функций TMOD и TCON. Текущее значение таймер-счетчика, соответствующее количеству подсчитанных импульсов, хранится и изменяется в таймерных регистрах TH0, TL0 и TH1, TL1 соответственно для таймер-счетчика 0 и 1. В различных режимах разрядность таймер-счетчика составляет 8–16 бит, таким образом, для подсчета используются либо только регистры TL_X, либо TH_X и TL_X, включенные последовательно, где TH_X содержит старшие биты числа, а TL_X — младшие, x — номер таймер-счетчика (0 или 1).

1.4.1. Регистр режима работы таймер-счетчика TMOD

Управление режимом работы таймер-счетчиков 0 и 1 осуществляет регистр TMOD:

Timer 1 (таймер-счетчик 1)				Timer 0 (таймер-счетчик 0)			
D7	D6	D5	D4	D3	D2	D1	D0
GATE	C/T	M1	M0	GATE	C/T	M1	M0

Поля регистра TMOD:

- C/T — выбор функции — 0 — таймер, 1 — счетчик;
- GATE — разрешение внешней блокировки. Если бит равен нулю, то включение и выключение соответствующего таймер-счетчика возможно только битом TR_X регистра TCON. В случае, когда бит равен единице, включение таймер-счетчика зависит не только от бита TR_X, но и от состояния на входе INT_X, на который необходимо подать уровень логической единицы для активации работы соответствующего таймер-счетчика;
- M [1:0] — код режима работы таймеров (табл. 5).

Таблица 5

Режим работы таймер-счетчика

M1	M0	Режим	Режим работы таймер-счетчика
0	0	0	13-битный таймер-счетчик. TH_X — 8 бит, TL_X — 5 (младших) бит
0	1	1	16-битный таймер-счетчик. TH_X и TL_X включен последовательно
1	0	2	8-битный автоперезагружаемый таймер-счетчик. TH_X хранит значение, которое должно быть перезагружено в TL_X каждый раз по переполнению
1	1	3	Таймер-счетчик 0 и 1 работают по-разному

1.4.2. Регистр управления и статуса таймера TCON

Регистр TCON управляет запуском таймер-счетчиков, содержит флаги переполнения таймер-счетчика, также используется для настройки прерываний от внешних источников. Структура регистра TCON:

TCON.7	TCON.6	TCON.5	TCON.4	TCON.3	TCON.2	TCON.1	TCON.0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Поля регистра TCON:

- TF_X — флаг переполнения таймер-счетчика. Устанавливается аппаратными средствами при переполнении таймер-счетчика, т. е. в случае перехода из максимального состояния таймерного регистра в минимальное. Сбрасывается автоматически при передаче управления подпрограмме обработки прерывания;
- TR_X — бит управления таймер-счетчика. Для активации работы таймер-счетчика 0 или 1 в соответствующий бит необходимо записать единицу. Сброс бита выключает соответствующий таймер-счетчика;
- IE_X — флаг фронта прерывания. Устанавливается аппаратно при возникновении активного сигнала на внешнем входе INT_X микроконтроллера (активный сигнал определяется битом IT_X). Сбрасывается автоматически при обслуживании прерывания;
- IT_X — бит выбора типа активного сигнала на входе INT_X. При $IT_X = 1$ активным является переход из высокого в низкий, при $IT_X = 0$ активным является низкий уровень сигнала.

1.4.3. Режимы работы таймер-счетчиков

Регистр TMOD позволяет выбрать один из четырех режимов работы для каждого таймер-счетчика, причем режимы работы 0, 1 и 2 одинаковы для обоих таймер-счетчиков, а режим 3 различен.

Схема функционирования таймер-счетчика в режиме 0 показана на рис. 5, где физические выводы микроконтроллера обозначены PIN. Для этого режима разрядность таймерного регистра составляет 13 бит, из которых 8 старших битов текущего значения содержатся в регистре TH_X, а 5 младших битов — в регистре TL_X. Модуль счета (число различных устойчивых состояний счетчика) для данного режима составляет $2^{13} = 8192$. Например, если $TH_X = 29h$ и $TL_X = 15h$, то в двоичной системе счисления этим числам соответствует запись: $TH_X = 00101001b$ и $TL_X = 00010101b$, тогда значение в таймерном регистре можно считать равным 0010100110101b, что в шестнадцатеричной системе счисления составляет 535h, а в десятичной — 1333.

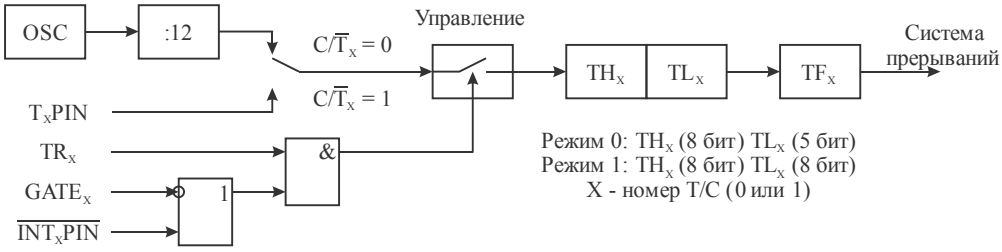


Рис. 5. Схема работы таймер-счетчика в режиме 0 и 1

При переполнении, т. е. переходе из максимального состояния, равного 8191, в минимальное — 0, в регистре TCON автоматически устанавливается флаг переполнения таймер-счетчика TF_X.

Подсчет импульсов, поступающих с внешнего входа T_X (см. рис. 5) или с генератора тактовой частоты через предделитель, осуществляется счетным узлом в двух случаях: когда управляющий бит TR_X установлен и бит разрешения внешней блокировки GATE сброшен, либо когда $TR_X = 1$, GATE = 1 и на внешнем входе микроконтроллера INT_X присутствует уровень логической 1.

Функционирование любого таймер-счетчика в режиме 1 полностью совпадает с режимом 0 за исключением того, что таймерный регистр имеет разрядность 16 бит. В этом случае модуль счета будет равен

65536, а регистры TH_X и TL_X используются полностью и также включены последовательно.

Как видно из рис. 6, управление работой таймер-счетчика в режиме 2 осуществляется аналогично режимам 0/1. Первым отличием функционирования является использование для подсчета только регистра TL_X, т. е. модуль счета равен 256. Вторым — при переполнении таймерного регистра, кроме установки флага переполнения, производится запись содержимого регистра TH_X в регистр TL_X, при этом значение TH_X не изменяется. Таким образом можно уменьшить модуль счета с 256 до любого значения, предварительно записав соответствующую разницу в регистр TH_X. Конечно, модуль счета может быть уменьшен и при использовании других режимов работы, но в данном случае перезапись определенного начального значения будет производиться автоматически.

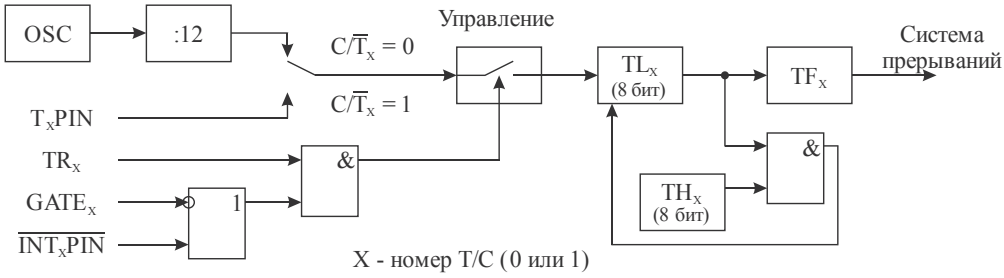


Рис. 6. Схема работы таймер-счетчика в режиме 2

Функционирование таймер-счетчиков 0 и 1 в режиме 3 различно (рис. 7). Таймер-счетчик 1 отключен, его таймерные регистры TH1 и TL1 сохраняют свое значение. Регистры TL0 и TH0 используются в качестве двух независимых таймерных регистров, причем TH0 может выполнять функции только таймера, а TL0 — таймера и счетчика. Таймер с регистром TH0 управляется только битом TR1, соответственно его можно только включить или выключить, других настроек произвести нельзя. При переполнении TH0 устанавливается флаг прерывания TF1. Работа TL0 аналогична функционированию в режимах 0 и 1, отличием является разрядность таймерного регистра, здесь она составляет 8 бит, управление производится битами таймер-счетчика 0 (C/T0, GATE0, TR0), вход для внешней блокировки — INT0 и флаг переполнения — TF0.

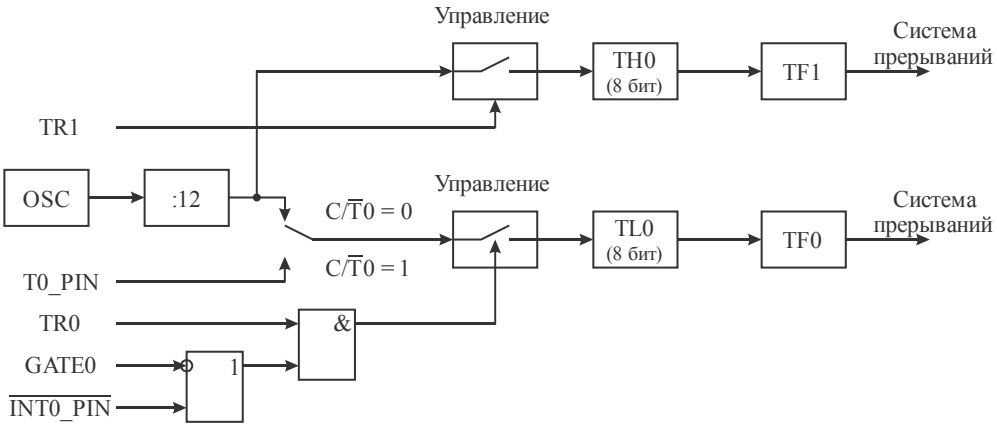


Рис. 7. Схема работы таймер-счетчика в режиме 3

В некоторых версиях микроконтроллеров семейства MCS-51 может присутствовать третий таймер-счетчик и (или) блок программных счетчиков PCA (Programmable Counter Array), которые также могут использоваться для отсчета временных интервалов.

1.5. Организация параллельных портов ввода-вывода

Базовый микроконтроллер семейства содержит один двунаправленный и три квазидвунаправленных 8-разрядных параллельных порта ввода-вывода P0—P3 с возможностью независимого индивидуального управления направлением передачи каждой линии. Выводы микроконтроллера обеспечивают обмен информацией с внешними устройствами и выполнение альтернативных функций, таких как обращение к внешней памяти, прием запросов прерываний, управление работой таймер-счетчиков, а также взаимодействие с цифровыми устройствами при помощи последовательного порта UART [2].

1.5.1. Альтернативные функции

Вследствие ограниченного количества контактов корпуса микроконтроллера, большинство выводов используется для выполнения двух функций: цифровые линии портов ввода-вывода и альтернативные функции.

Порты P0 и P2 используются для обращения к внешней памяти. Выводы порта P0 предназначены для вывода младших разрядов адреса, а также для ввода и вывода байта данных, в связи с чем значение адреса должно фиксироваться во внешнем регистре-защелке. При обращении к внешней памяти с 16-разрядным адресом для передачи старших разрядов используются выводы порта P2. Использование 8-разрядной адресации позволяет освободить порт P2 от альтернативной функции и применять его как обычный цифровой порт ввода-вывода. Обращение к внешней памяти вызывает автоматическую запись 1 во все разряды регистра P0, при этом содержимое регистра P2 остается неизменным.

Порт P3 используется для формирования и приема специальных управляющих и информационных сигналов. Альтернативная функция каждого вывода настраивается независимо от других и активируется записью 1 в соответствующий разряд регистра (табл. 6).

Таблица 6

Альтернативные функции порта ввода-вывода P3

Вывод порта	Название	Альтернативная функция
P3.0	RXD	Вход последовательного порта
P3.1	TXD	Выход последовательного порта
P3.2	INT0	Внешнее прерывание 0
P3.3	INT1	Внешнее прерывание 1
P3.4	T0	Вход таймер-счетчика 0
P3.5	T1	Вход таймер-счетчика 1
P3.6	WR	Строб записи во внешнюю память данных
P3.7	RD	Строб чтения из внешней памяти данных

1.5.2. Устройство портов

Поскольку порты ввода-вывода размещены в пространстве DSEG (внутренняя память данных), любая команда с операндом из DSEG применима к содержимому P0–P3, а совмещение с BSEG (битовое пространство данных) позволяет иметь доступ к каждому биту портов. С учетом этого обстоятельства структура портов рассматривается на уровне одного разряда.

При обращении к внешней памяти порты P0 и P2 выполняют функцию системных шин Адрес [7:0]/Данные и Адрес [15:8] соответствен-

но (см. рис. 2), в этом случае порты P0 и P2 не могут использоваться для операций ввода-вывода. Линии порта P3: P3.6 и P3.7 используются для передачи управляющих сигналов (WR и RD).

При отсутствии в системе внешней памяти, линии управления можно использовать как обычные программируемые линии порта. Можно программировать и интерфейс памяти, подавая на выходы RD, WR, PSEN импульсы произвольной длительности.

Структурные схемы аппаратных средств, обслуживающих одну линию каждого из портов P0–P3, приведены на рис. 8–11 [2]. Основой каждой схемы является регистр-защелка (D-триггер), запись информации в который осуществляется с внутренней шины данных микроконтроллера по сигналу «Запись в защелку» от ЦП. Состояние D-триггера может быть считано через внутреннюю шину по сигналу «Чтение защелки». Информация с вывода микросхемы считывается по сигналу «Чтение выводов». Кроме защелки, каждая линия ввода-вывода имеет входной буфер и формирователь выходного сигнала. Внутренний сигнал «Управление» устанавливается равным 1 только при обращении к внешней памяти, тем самым поднимая вверх переключатель MUX. В противном случае низкий уровень сигнала «Управление» закрывает транзистор VT1 порта P0 (рис. 8).

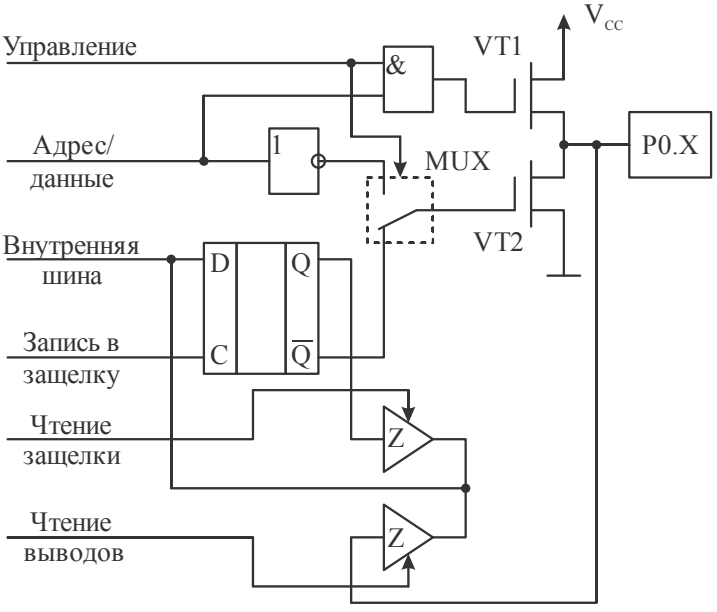


Рис. 8. Структура разряда порта P0

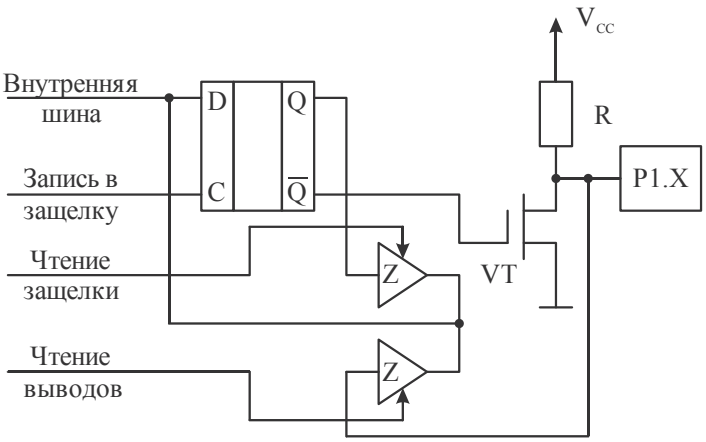


Рис. 9. Структура разряда порта P1

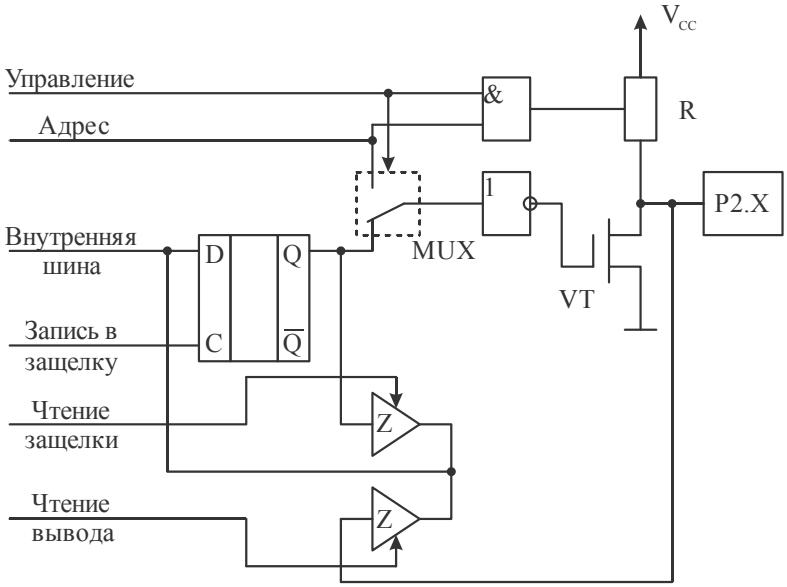


Рис. 10. Структура разряда порта P2

Выходные каскады триггеров портов P1–P3 выполнены на полевых транзисторах с внутренней нагрузкой (R — pull-up нагрузка), выходной ток — 1,6 мА, что позволяет подключать до четырех маломощных ТТЛ-входов, в то время как аналогичные каскады триггеров P0 выполнены на транзисторах с открытым стоком с нагрузочной способностью до 3,2 мА, однако для их работы требуются внешние нагрузочные резисторы.

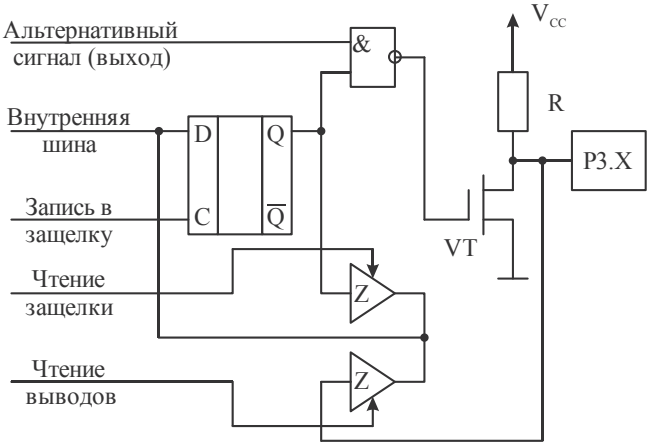


Рис. 11. Структура разряда порта P3

В режиме ввода-вывода линий портов для вывода значения логического 0 или 1 на внешний контакт микросхемы необходимо записать это значение в соответствующий разряд регистра параллельного порта. При записи логического 0 с выхода защелки подается сигнал на затвор выходного транзистора (VT2 для P0 и VT для P1–P3), открывая его, для этого используется инверсный выход триггера либо неинверсный выход и инвертор. Открытый выходной транзистор формирует низкий уровень вследствие подключения общего провода к выводу. Следуя обратной логике при записи логической 1, транзистор будет закрыт, тем самым формируется высокий уровень за счет внешнего нагрузочного резистора у порта P0 и внутренних — у портов P1, P2, P3, подключенных к источнику питания.

Если линия порта используется в качестве входа, то ее защелка должна содержать 1. При этом у портов P1, P2, P3 выходной полевой транзистор отключает общий провод от выходного контакта, а соответствующая линия внутри подтягивается к высокому уровню (рис. 9–11), но может быть внешним источником переведена в состояние 0. **Квазидвунаправленный порт** — это порт, выходной каскад которого состоит из одного транзистора и внутреннего резистора, подтягивающего линию к высокому уровню. Порты P1, P2, P3 являются квазидвунаправленными.

Порт P0 не имеет элементов постоянного подтягивания к высокому уровню. Транзистор VT1 выходного формирователя открыт только при подаче логических 1 на линию шины адрес-данные и на внутрен-

ний сигнал «Управление». При работе порта P0 на ввод-вывод данных состояние сигнала «Управление» поддерживается равным 0, а запись 1 в защелку порта P0 переводит соответствующий вывод микросхемы в высокоимпедансное состояние, в связи с этим данный порт является действительно двунаправленным — его линии должны быть подтянуты к уровню 1 внешними нагрузочными резисторами.

При взаимодействии с внешней памятью (режим альтернативных функций) сигнал «Управление» имеет значение логической 1, переключатель MUX, управляемый этим сигналом, поднят «вверх». Информационные биты, поступающие по шине адрес-данные порта P0, управляют транзисторами VT1 и VT2, включенными противофазно, тем самым формируя требуемый сигнал на выходе микросхемы. Когда необходима 16-разрядная адресация, старшая часть адреса передается через порт P2, формируя выходной сигнал с помощью полевого транзистора VT и нагрузочного резистора R.

Как было сказано выше, альтернативные функции порта P3 могут быть активированы путем занесения в соответствующие биты логических 1. Формирование выходных уровней напряжения для портов P2 и P3 производится аналогично режиму вывода линии портов.

Некоторые команды микроконтроллера позволяют читать содержимое защелки (регистры портов), а другие — значение сигналов на линиях портов.

При сбросе микроконтроллера все регистры портов устанавливаются в состояние FFh.

К портам ввода-вывода, как к ячейке РПД, можно обращаться с помощью команд, оперирующих с байтом или отдельным битом. Кроме того, использование логических команд, оперирующих с байтом, позволяет модифицировать произвольную комбинацию бит. Если разряды порта одновременно являются операндом и местом получения результата, то устройство управления реализует специальный режим, называемый *чтение-модификация-запись*, при этом считывание информации осуществляется из регистров-защелок разрядов порта, а не с внешних выводов микроконтроллера.

1.6. Последовательный асинхронный интерфейс UART

Последовательный интерфейс для передачи данных использует одну сигнальную линию, по которой информационные биты передаются друг за другом последовательно. Различают полудуплексные и дуплексные каналы последовательной связи. У полудуплексного канала данные передаются по одному и тому же каналу в обе стороны, но в каждый конкретный момент времени только в одну сторону. Дуплексный канал позволяет передавать данные по двум каналам (в одну сторону по первому, в другую — по второму). При этом появляется возможность передавать информацию в обе стороны одновременно.

Универсальный асинхронный приемопередатчик UART (Universal Asynchronous Receiver-Transmitter — УАПП) микроконтроллеров семейства MCS-51 предназначен для обмена информацией с внешними устройствами, имеющими аналогичный интерфейс. UART совместим с интерфейсом RS-232 (EIA-232), но имеет некоторые особенности, часть из них упоминаются ниже. Прием и передача данных осуществляется в полном дуплексном режиме обмена, информация представляется в виде последовательного кода (младшими разрядами вперед). Блок UART микроконтроллеров семейства MCS-51 состоит из принимающего и передающего сдвигающих регистров, буферного регистра (SBUF) приемника и передатчика, а также блока управления работой порта (регистр управления SCON и бит SMOD регистра управления мощностью PCON) [2].

После записи данных в буферный регистр производится автоматический перенос содержимого SBUF в сдвигающий регистр передатчика и начинается процесс передачи информации. Наличие буферного регистра приемника позволяет одновременно считывать ранее принятые данные из SBUF и принимать очередной байт информации. Когда новый байт принят, его значение записывается в буферный регистр. Таким образом, если за время приема очередного байта данных предыдущий из SBUF не считан, то он теряется.

Блок UART позволяет передавать данные в асинхронном или синхронном режимах. Скорости обмена приемника и передатчика должны быть предварительно согласованы, т. е. должны быть одинаковыми. Асинхронная передача (рис. 12) начинается со старт-бита (логический 0), которым приемник информируется о начале передачи. При-

емник автоматически подстраивает появление стробирующих импульсов для фиксации битов принимаемой последовательности по времени пришедшего старт-бита. Далее следуют биты данных и, возможно, бит четности (P). Завершающим битом последовательности является стоп-бит, имеющий уровень логической 1. Он гарантирует паузу между передаваемыми посылками. Следующий байт информации может передаваться после стоп-бита через любой промежуток времени.

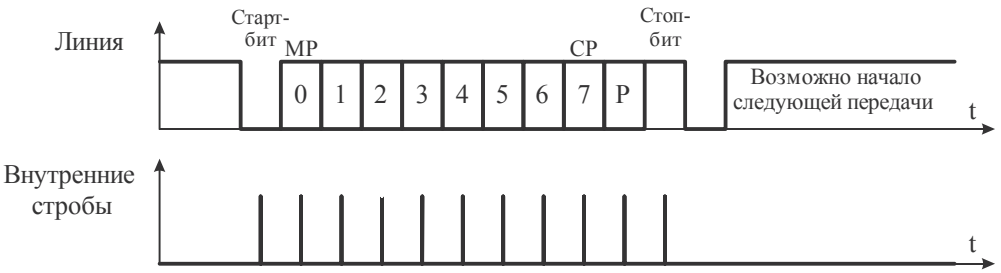


Рис. 12. Формат асинхронной передачи

В соответствии со стандартом RS-232 (EIA-232) асинхронный интерфейс поддерживает ряд стандартных скоростей обмена в диапазоне 110–115 200 бит/с (бод), в одной посылке возможна передача 5–8 информационных бит, длительность стоп-бита может составлять 1; 1,5 или 2 интервала стробирующего импульса. Интерфейс UART микроконтроллеров семейства MCS-51 имеет ряд ограничений, в частности, поддерживается не весь ряд стандартных скоростей обмена для асинхронного режима (см. 1.6.3), количество передаваемых информационных бит данных равно 8.

Интерфейс UART в синхронном режиме передачи данных позволяет обмениваться информацией только в полудуплексном режиме, при этом одна линия используется для данных, а другая — для синхронизации. Передача и прием данных осуществляется сдвиговыми регистрами.

1.6.1. Регистр управления и статуса приемопередатчика SCON

Выбор режима работы, а также настройка и определение других параметров последовательного порта осуществляется при помощи регистра SCON, функциональное назначение битов которого показано ниже:

SCON.7	SCON.6	SCON.5	SCON.4	SCON.3	SCON.2	SCON.1	SCON.0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Поля регистра SCON:

- RI — флаг прерывания приемника. Устанавливается аппаратным способом в конце приема посылки. В синхронном режиме (режим 0) — в конце периода восьмого бита данных. В асинхронных режимах (режимы 1, 2 и 3) — в середине периода приема стоп-бита. Сброс флага должен производиться в подпрограмме обслуживания прерывания;
- TI — флаг прерывания передатчика. Устанавливается аппаратно в конце передачи посылки. В синхронном режиме во время передачи 8-го бита данных, в асинхронных режимах — в начале стоп-бита. Подпрограмма обслуживания прерывания должна содержать команду очистки флага TI;
- RB8 — прием бита 8. В синхронном режиме не используется. В асинхронном режиме 1 в RB8 фиксируется стоп-бит. В режимах 2 и 3 в RB8 заносится девятый принимаемый бит данных;
- TB8 — передача бита 8. Устанавливается и сбрасывается программой в режимах 2 и 3, содержит значение девятого бита данных, который будет передаваться;
- REN — бит разрешения приема, предназначен для отключения режима приема данных. Модифицируется программой. Если бит установлен, то прием последовательных данных разрешен. Сброс бита запрещает прием через последовательный порт;
- SM2 — бит управления режимом UART. Флаг прерывания приемника (RI) не будет установлен в случае, когда SM2 = 1 в режиме 1 при отсутствии действительного стоп-бита или в режимах 2 и 3 при приеме девятого бита данных (RB8), равного 0. В режиме 0 всегда должен быть сброшен;
- SM1, SM0 — биты предназначены для программного выбора режима работы приемопередатчика (табл. 7).

Таблица 7

Режим работы приемопередатчика последовательного порта

SM0	SM1	Режим работы приемопередатчика	
0	0	0	Сдвигающий регистр расширения ввода-вывода
0	1	1	8-битовый приемопередатчик, изменяемая скорость передачи

Окончание табл. 7

SM0	SM1	Режим работы приемопередатчика	
1	0	2	9-битовый приемопередатчик. Фиксированная скорость передачи
1	1	3	9-битовый приемопередатчик, изменяемая скорость передачи

1.6.2. Регистр управления мощностью PCON

Настройка скорости передачи последовательного порта, выбор режима пониженного энергопотребления осуществляется программой путем записи соответствующего управляющего слова в регистр PCON. Кроме того, в этом же регистре присутствуют пользовательские флаги:

PCON.7	PCON.6	PCON.5	PCON.4	PCON.3	PCON.2	PCON.1	PCON.0
SMOD				GF1	GF0	PD	IDL

Поля регистра PCON:

- IDL — бит холостого хода. Запуск режима холостого хода микроконтроллера осуществляется путем занесения 1 в бит IDL;
- PD — бит пониженного потребления мощности. Установкой бита в 1 микроконтроллер переводится в соответствующий режим энергопотребления;
- GF0, GF1 — биты общего назначения, предназначены для использования по усмотрению программиста;
- SMOD — удвоенная скорость передачи последовательного порта UART. При установке бита в 1, скорость передачи увеличивается в два раза по сравнению с SMOD = 0 (см. 1.6.3). Перезапуск микроконтроллера обнуляет SMOD.

При одновременной установке в 1 битов PD и IDL будет включен режим пониженного потребления мощности (бит PD). Сброс содержимого PCON выполняется путем загрузки в него кода 0XXX0000.

1.6.3. Режимы работы последовательного порта

Регистр SCON позволяет выбрать один из четырех режимов работы последовательного порта, причем режим 0 предназначен для синхронной передачи информации, а режимы 1, 2 и 3 — для асинхронной.

В режиме 0 передача и прием данных осуществляется через внешний вывод RxD (линия порта P3.0). Синхропоследовательность, стробирующая передаваемые или принимаемые биты, передается через вывод TxD (линия порта P3.1). Длина посылки составляет 8 бит. Передача и прием данных осуществляется младшими разрядами вперед при помощи сдвигающих регистров, причем сдвиг данных производится в фазе (S6P2) каждого машинного цикла, что определяет скорость обмена, равную 1/12 частоты системы синхронизации микроконтроллера,

$$f = \frac{f_{\text{OSC}}}{12},$$

где f_{OSC} — резонансная (рабочая) частота кварцевого резонатора.

В режиме 1 передача данных производится через вывод TxD, а прием — через RxD. Длина посылки составляет 10 бит: старт-бит (логический 0), 8 бит данных и стоп-бит (логическая 1). При приеме в бите RB8 регистра управления/статуса приемопередатчика SCON.2 фиксируется стоп-бит. Скорость приема-передачи определяется частотой переполнения таймер-счетчика 1 (f_{OVLT1}) и управляющим битом SMOD регистра PCON

$$f = \frac{2^{\text{SMOD}} f_{\text{OVLT1}}}{32}.$$

В этом случае прерывание от таймер-счетчика 1 должно быть отключено. Остальные настройки таймер-счетчика не важны, он может работать в любом режиме и выполнять функцию таймера или счетчика. Его удобно использовать как таймер в режиме 2 (с автоперезагрузкой), тогда частота передачи будет определяться по выражению

$$f = \frac{2^{\text{SMOD}} f_{\text{OSC}}}{32 \cdot 12 \cdot (256 - TH1)}.$$

Набор стандартных скоростей обмена и соответствующие параметры таймер-счетчика 1 приведены в табл. 8.

В режиме 2 передача данных производится через вывод TxD, а прием — через RxD. Длина посылки составляет 11 бит: старт-бит (логический 0), 8 бит данных, программируемый девятый бит (RB8 или TB8 регистра SCON) и стоп-бит (логическая 1). Девятый передаваемый бит является программируемым, как правило, он используется для передачи контрольного бита — бита четности из слова состояния

программы (PSW.0), что позволяет осуществлять проверку правильности передачи. Девятый принимаемый бит данных помещается в бит RB8 регистра SCON, он также может использоваться для повышения достоверности приема. Скорость приема-передачи определяется битом SMOD регистра PCON и зависит от частоты системы синхронизации микроконтроллера

$$f = \frac{2^{\text{SMOD}} f_{\text{OSC}}}{64}.$$

Таким образом, скорость приема-передачи для режима 2 может составлять 1/32 или 1/64 рабочей частоты кварцевого резонатора.

Таблица 8

Настройка таймера 1 для управления частотой работы приемопередатчика

Частота приема и передачи (BAUD RATE)	Частота резонатора, МГц	Таймер-счетчик 1			
		SMOD	C/T	Режим (MODE)	Перезагружаемое число
Режим 0; 1 МГц	12	X	X	X	X
Режим 2; 375 кГц	12	1	X	X	X
Режим 1, 3; 62,2 кГц	12	1	0	2	0FFh
19,2 кГц	11,059	1	0	2	0FDh
9,6 кГц	11,059	0	0	2	0FDh
4,8 кГц	11,059	0	0	2	0FAh
2,4 кГц	11,059	1	0	2	0F4h
1,2 кГц	11,059	0	0	2	0F4h
137,5 Гц	11,059	0	0	2	1Dh
110 Гц	6	0	0	2	72h
110 Гц	12	0	0	1	0FEEBh

Функционирование последовательного порта в режиме 3 совпадает с работой в режиме 2, исключением является скорость приема и передачи данных, которая определяется переполнением таймер-счетчика 1 и задается так же, как в режиме 1.

В результате выполнения любой команды, содержащей буферный регистр SBUF как получатель байта, инициируется процесс передачи данных по последовательному интерфейсу UART. Прием данных в режиме 0 может быть осуществлен, когда RI = 0 и REN = 1, в других режимах — когда REN = 1 [2].

1.7. Система прерываний

Система прерываний предназначена для быстрой реакции на события, которыми могут быть: наличие определенного логического уровня напряжения или его изменение на входе микроконтроллера, переполнение таймер-счетчика, завершение передачи или приема байта через последовательной порт и др. Когда система прерываний сконфигурирована, при возникновении одного из событий, разрешенных к обработке, останавливается выполнение основной программы и запускается подпрограмма обработки данного прерывания, по окончании выполнения которой происходит возврат в основную программу. Информацию о возникновении какого-либо события несет флаг соответствующего прерывания. Стоит отметить, что наличие активного состояния флага прерывания не означает обязательного запуска подпрограммы обслуживания данного прерывания, для этого также необходимо соответствующее разрешение.

Архитектура системы управления прерываниями базовой модели Intel 8051 показана на рис. 13.

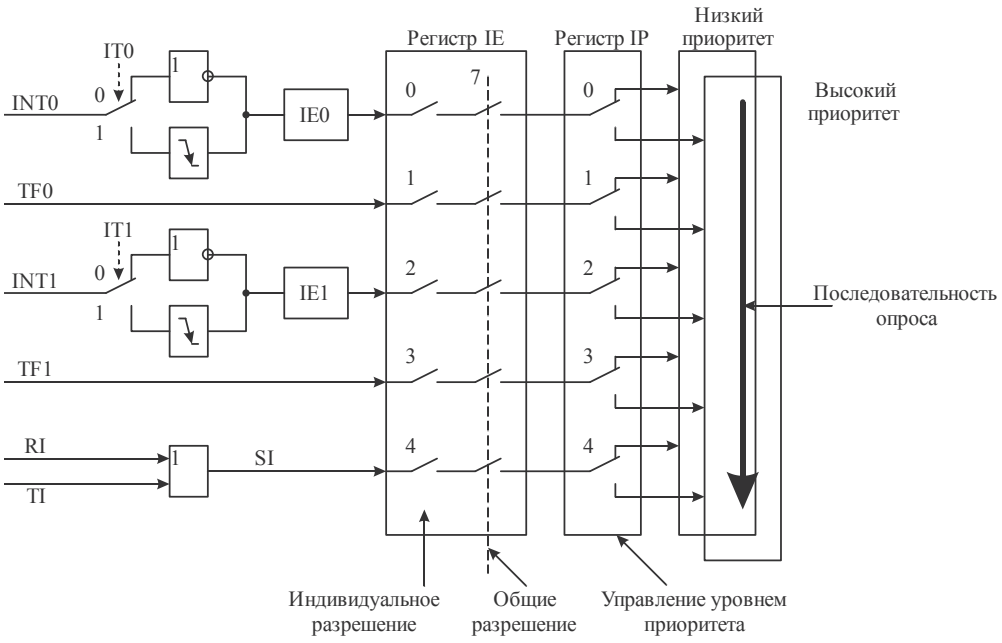


Рис. 13. Система управления прерываниями

Разрешение или запрет каждого прерывания осуществляется записью 1 или 0 в соответствующий разряд регистра специальных функций IE (Interrupt Enable). Кроме того, можно полностью отключить систему прерываний, сбросив бит общего запрещения. Структура регистра IE:

IE.7	IE.6	IE.5	IE.4	IE.3	IE.2	IE.1	IE.0
EA			ES	ET1	EX1	ET0	EX0

Поля регистра IE:

- EA — включение системы прерываний;
- ES — разрешение прерывания от последовательного порта;
- ET_X — разрешение прерывания по переполнению таймер-счетчика X;
- EX_X — разрешение прерывания по внешнему сигналу на входе $\overline{INT_X}$.

Выбор типа активного сигнала на входе INT_X: переход из уровня логической единицы в ноль или низкий уровень осуществляется программированием битов IT_X регистра TCON (см. 1.4.2, с. 19). При возникновении активного сигнала на входах INT_X автоматически устанавливаются соответствующие флаги IE_X (см. рис. 13).

По окончании приема или передачи посылки по последовательному порту будет устанавливаться флаг прерывания приемника (бит RI регистра SCON) или флаг прерывания передатчика (бит TI), которые сигналом SI (рис. 13) вызовут одну и ту же подпрограмму обслуживания прерывания по последовательному порту в случае разрешения этого прерывания. В данной подпрограмме необходимо проанализировать установленный флаг, чтобы определить источник прерывания.

Система прерываний микроконтроллеров семейства MCS-51 является приоритетной, что позволяет задать высокий или низкий уровень приоритета, указав 1 или 0 в разрядах регистра IP (Interrupt Priority):

IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
			PS	PT1	PX1	PT0	PX1

Поля регистра IP:

- PS — приоритет последовательного порта;
- PT1 — приоритет таймер-счетчика 1;

- PX1 — приоритет внешнего прерывания $\overline{\text{INT1}}$;
- PT0 — приоритет таймер-счетчика 0;
- PX0 — приоритет внешнего прерывания $\overline{\text{INT0}}$.

При одновременном возникновении двух прерываний: одного с высоким, другого с низким уровнем приоритета — сначала будет выполняться высокоприоритетное. В случае поступления прерываний с одним уровнем приоритета последовательность их обработки будет определяться последовательностью опроса, показанной на рис. 13.

Выполнение подпрограммы прерывания

Система прерываний формирует код команды безусловного перехода (LCALL) к соответствующей подпрограмме обслуживания, если нет одного из следующих условий блокировки:

- в данный момент обслуживается запрос прерывания равного или высокого уровня приоритета;
- текущий машинный цикл не последний в цикле выполняемой команды;
- выполняется команда RETI или любая команда, связанная с обращением к регистрам IE или IP.

Если во время блокировки прерывания хотя бы по одному из перечисленных условий флаг прерывания был установлен, а затем сброшен до момента окончания блокировки, то прерывание не будет обслужено, т. к. запрос нигде не сохраняется.

В начале обслуживания прерывания система автоматически формирует безусловный переход на соответствующую подпрограмму, при этом в стек помещается только содержимое счетчика команд (PC, т. е. адрес следующей команды). Затем в регистр PC загружается адрес вектора соответствующей подпрограммы обслуживания (табл. 9). **Вектор прерывания** — строго закрепленный за каждым прерыванием адрес, в ячейке памяти программ с соответствующим адресом располагают команду безусловной передачи управления (JMP) к начальному адресу подпрограммы обслуживания прерывания. Для сохранения исходных данных регистров, используемых в подпрограмме обслуживания прерывания (слова состояния программы (PSW), аккумулятора и т. д.), в ее начале располагают команды записи в стек (PUSH) соответствующих

регистров, а в конце — команды восстановления из стека (POP), при этом регистры указываются в обратном порядке. Завершение подпрограммы должно осуществляться командой RETI, которая, в отличие от команды RET, не только восстанавливает значения счетчика команд из стека, но и снимает блокировку соответствующего прерывания.

Таблица 9

Адреса векторов прерываний микроконтроллера Intel 8051

Прерывание	Наименование	Адрес вектора	Последовательность опроса
IE0	Внешнее прерывание INT0	03h	1
TF0	Переполнение таймер-счетчика 0	0Bh	2
IE1	Внешнее прерывание INT1	13h	3
TF1	Переполнение таймер-счетчика 1	1Bh	4
RI/TI	Прерывание UART	23h	5

Обработка прерывания возможна при выполнении следующих действий:

- расположите подпрограмму обслуживания прерывания по адресу соответствующего прерывания (или команду перехода на подпрограмму обслуживания);
- установите бит разрешения всех прерываний (EA) в регистре IE и бит разрешения требуемого прерывания.

Вектора прерываний базового микроконтроллера семейства MCS-51 находятся в диапазоне 0003h–0023h (табл. 9) и попадают в начальную область памяти программ. Чтобы исключить перекрытие программного кода, как правило, по адресам векторов прерываний располагают команды безусловного перехода на подпрограммы обработки прерываний, а в ячейку с адресом 0000h — на начало основной программы.

1.8. Система команд

Система команд — это набор всех команд, предусмотренных машинными кодами данной микропроцессорной системы (МПС) или ее части.

Команда — полная запись в программе, определяющая действие и данные (или адреса данных), над которыми это действие должно

выполняться. Команда подается на вход МП (для MCS-51 регистр IR, рис. 2), под ее действием производится группа операций. Подавая последовательно команды, мы можем реализовать конкретный алгоритм любой сложности.

Система команд микроконтроллеров рассматриваемого семейства довольно обширна и включает в себя 111 основных команд. Команды можно классифицировать по нескольким параметрам: по длине, по типу взаимодействующих операндов и по функциональному назначению (рис. 14). Все команды выполняются за один или два машинных цикла (соответственно 1 или 2 мкс при тактовой частоте 12 МГц), исключение — команды умножения и деления, выполняемые за четыре машинных цикла (4 мкс).

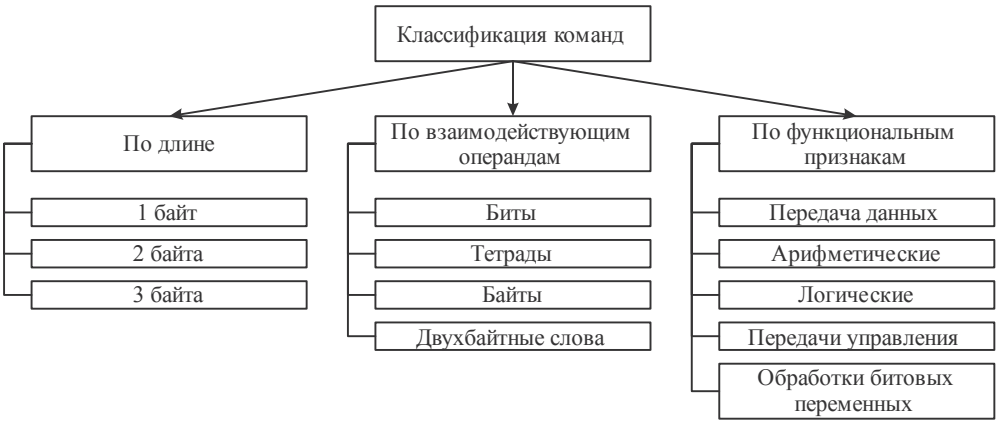


Рис. 14. Классификация команд МП

Длина команд может быть 1–3 байт. Большинство двухбайтовых команд одноцикловые, а все трехбайтовые — двухцикловые. За один машинный цикл в микроконтроллер можно вводить до двух байт программного кода. Однокристальные микроЭВМ семейства MCS-51 используют прямую, косвенную, регистровую, непосредственную, индексную, неявную и битовую адресацию данных.

В качестве операндов команд микроконтроллеры семейства MCS-51 могут использовать отдельные биты, четырехбитные цифры (тетрады), байты и двухбайтные слова.

Система команд микроконтроллеров семейства MCS-51 имеет некоторые особенности, необходимые для создания систем управления, а именно: включает команды манипуляции с отдельными битами

и имеет большое число операций ввода-вывода, что позволяет легко управлять отдельными выводами параллельных портов ввода-вывода, а также команды ветвлений и организации циклов.

Команды микроконтроллеров семейства MCS-51 по функциональному признаку можно подразделить на пять групп:

- команды пересылки данных — осуществляют пересылку данных в неизменном виде (28 команд);
- арифметические команды — позволяют производить арифметические операции над 8- и 16-битными операндами (24 команды);
- логические команды — предназначены для поразрядной обработки информации (25 команд);
- команды передачи управления — представляют возможность выполнять безусловные и условные переходы в программе, а также осуществлять вызовы подпрограмм и возврат из них (17 команд);
- команды обработки битовых переменных — предназначены для установки, сброса, выполнения логических и других операций над отдельно адресуемыми битами (17 команд).

1.8.1. Правила записи команд на языке ассемблера

Язык ассемблера — набор мнемокодов и символических адресов, допускает представление всех элементов программы в символическом виде.

Ассемблирующая программа (компилятор, транслятор) — специальная программа, преобразующая символические наименования в двоичные коды. **Дисассемблирующая программа** выполняет обратное действие.

Символические наименования, вводимые при программировании на ассемблере, имеют определения:

метка — символическое имя адреса команды;

мнемокод — символическое имя кода операции;

идентификатор — символическое обозначение объекта программы. Идентификатор может начинаться только с буквы, что позволяет отличать его от числа. В языке программирования ASM-51 имеются три категории идентификаторов:

- ключевые слова (инструкции, директивы, вспомогательные слова, операции);
- встроенные имена (обозначения регистров);

- определяемые имена (метки, внутренние и внешние переменные адресного типа и числового типа и т. д.).

Программы на языке ассемблера представляют собой последовательность команд, называемых также **операторами**. В качестве алфавита допустимых символов принят 7-разрядный код ASCII (*American Standard Code for Information Interchange*).

Ассемблер не является языком свободной формы. Каждый оператор ассемблера занимает одну строку, состоящую из четырех полей: метки, мнемокода, операндов, комментариев —

```
метка: мнемокод операнд1, операнд2; комментарий
BEG: MOV R2, #50h ; Меткой BEG помечена команда MOV
LOOP1:
LOOP2: ANL A, AFh ; Команда ANL помечена двумя метками
       LJMP LOOP1 ; Передача управления команде ANL
```

Между полями команды необходимо размещать разделители, которыми могут являться пробелы или знаки табуляции. Если в команде присутствует несколько операндов, то они разделяются запятыми. Меткой обозначается команда, на которую требуется сделать переход из другого места программы, поэтому поле метки заполняется в соответствующих случаях. Метка отделяется от мнемокода команды двоеточием. Поле мнемокода заполняется обязательно. Наличие, количество и тип операнда (операндов) определяются по записанной команде. Комментарий, располагаемый после знака точки с запятой, транслятором не обрабатывается, он предназначен для программиста, здесь можно использовать любые символы (с кодом ASCII не меньше 20h), в том числе и русские буквы. Строка должна заканчиваться парой символов — возврата каретки (0Dh) и перевода строки (0Ah).

В командах обработки информации могут указываться один или два операнда. При явном использовании двух операндов один из них называется источником S (*Source*), а второй — приемником D (*Destination*). В поле операндов первым записывается приемник, а вторым — источник. В этом случае производится чтение источника и (при необходимости) приемника, вычисление и (при необходимости) запись результата по адресу приемника. Когда в команде указывается один операнд, производится чтение операнда, его проверка или изменение и (при необходимости) запись.

Обозначения, используемые при описании команд, а также список всех команд микроконтроллера приведены в прил. 1, с. 131.

1.8.2. Команды пересылки данных

Несмотря на пересылку данных в неизменном виде, эти команды осуществляют один из способов обработки информации, например сортировку [3]. В командах пересылки используется все разнообразие способов адресации данных. Пересылка данных может осуществляться в форматах байта, половины байта, двух байтов или бита.

Данную группу команд можно подразделить на следующие подгруппы:

- команды передачи данных, использующие внутреннюю память данных;
- команды передачи данных, использующие внешнюю память данных;
- команды работы с таблицами.

Команды передачи данных, использующие внутреннюю память данных, включают такие команды, как MOV, PUSH, POP, XCH и XCHD.

Команда MOV (англ. MOVe означает «передвинуть») копирует содержимое источника в приемник (при выполнении этой команды первоначальное содержимое приемника теряется):

```
MOV A, ad  ; (ad)→(A), передача из внутреннего ОЗУ в аккумулятор
MOV ad, A ; (A)→(ad), передача из аккумулятора в память
```

Одна из команд пересылки данных записывает два байта в регистр указателя данных:

```
MOV DPTR, #d16 ; #d16→(DPTR), загрузка указателя данных
```

MOV используется для записи значения в регистр указателя данных DPTR, который применяется в командах работы с таблицами, расположенными в памяти программ, а также в командах обращения к внешней памяти данных.

```
MOV add, ads ; (ads)→(add), пересылка прямоадресуемого байта
             ; по прямому адресу
```

Команда MOV add, ads позволяет пересылать данные между ячейками внутреннего ОЗУ и (или) регистрами специальных функций без использования аккумулятора.

Как было показано в п. 1.3.2 и 1.3.3 стек размещается в резидентной памяти данных и «растет вверх». При выполнении команды PUSH сначала инкрементируется значение указателя стека SP, а затем байт данных сохраняется в стек.

```
PUSH ad ; (SP)+1→(SP), ad→((SP)), загрузка в стек
POP ad  ; ((SP))→(ad), (SP)-1→(SP), извлечение из стека
```

Команда пересылки данных XCH (eXCHange означает «обменять») осуществляет обмен содержимого источника и приемника, который можно произвести при помощи трех команд MOV. Но команды XCH делают это за один машинный цикл, занимая меньше места в памяти программ и не требуя использования дополнительной ячейки памяти данных:

```
XCH A, ad  ; (A)↔(ad), обмен аккумулятора
           ; с прямоадресуемым байтом
XCH A, @Ri ; (A)↔((Ri)), обмен аккумулятора
           ; с прямоадресуемым байтом
```

Также есть команда, обменивающая младшие половины байтов:

```
XCHD A, @Ri ; (A0..3)↔ (Ri0..3)), обмен младших тетрад
            ; аккумулятора и байта РПД.
```

Здесь D означает Digit (четыре бита используются для двоичного представления десятичной цифры).

Система команд микроконтроллеров рассматриваемого семейства содержит одну команду передачи данных, использующую внешнюю память данных (ВПД), — MOVX. Комбинирование источников и приемников данных увеличивает количество команд до четырех.

Чтение и запись данных байтового формата при обращении к внешнему ОЗУ осуществляется при помощи команд MOVX, где буква X означает eXternal (внешняя память):

```
MOVX A, @Ri    ; пересылка в аккумулятор байта из ВПД
MOVX @Ri, A    ; пересылка в ВПД из аккумулятора
MOVX A, @DPTR  ; пересылка в аккумулятор байта
               ; из расширенной ВПД
MOVX @DPTR, A  ; пересылка в расширенную ВПД из аккумулятора
```

Для доступа к внешней памяти данных используется только косвенная адресация, и обмен информацией осуществляется исключительно через аккумулятор, аккумулятор в этом случае играет роль источника либо приемника информации. Для 8-разрядных адресаций используется R0 или R1 текущего регистрового банка, а для 16-разрядных — регистр DPTR. Сигналы чтения и записи (RD и WR) активизируются только во время выполнения команд MOVX.

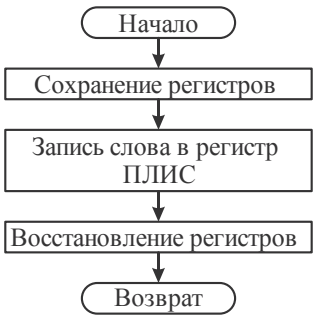


Рис. 15. Блок-схема алгоритма программы putbyte

Пример подпрограммы, используемой в лабораторных работах для пересылки управляющих слов в регистры ПЛИС, адресное пространство которых расположено на 8-й странице внешней памяти данных, а адрес регистра ПЛИС передается в подпрограмму в РОН R0, приведен ниже. Блок-схема алгоритма показана на рис. 15.

```
putbyte:
PUSH dpp       ; сохранение номера текущей страницы в стек
MOV dpp, #08d  ; выбор 8-й страницы внешней памяти
MOVX @R0, A    ; запись в регистр ПЛИС
POP dpp        ; восстановление номера страницы из стека
RET            ; возврат из подпрограммы
```

Команды для работы с таблицами — MOVC.

Чтение данных таблиц из ПЗУ осуществляется при помощи команды MOVC, буква C означает Constant (константа).

```
MOVC A, @A+DPTR ; ((A)+(DPTR))→(A), пересылка
                ; в аккумулятор байта из ПП путем
                ; использования DPTR
```

```
MOVC A, @A+PC ; (PC)+1→(PC), ((A)+(PC))→(A), пересылка
              ; в аккумулятор байта из ПП путем
              ; использования PC
```

С помощью этих команд, осуществляющих доступ исключительно к памяти программ, возможно только чтение таблиц, но не их изменение. Если таблица расположена во внешней программной памяти, то чтение байта из нее сопровождается стробом PSEN (Program Store Enable).

Обе команды MOVC предназначены для обращения к таблице с максимальным числом элементов 256 (0–255). Номер требуемого элемента таблицы загружается в аккумулятор, а адрес начала таблицы записывается в регистр DPTR или счетчик команд PC. При использовании второй команды MOVC обращение к таблице производится из подпрограммы. Сначала номер требуемого элемента таблицы загружается в аккумулятор, затем вызывается подпрограмма

```
MOV A, Num ; загрузка в аккумулятор номера требуемого
           ; элемента таблицы
CALL Table ; вызов подпрограммы
```

Подпрограмма Table будет выглядеть следующим образом:

```
Table: MOVC A, @A+PC ; загрузка в аккумулятор элемента таблицы
       RET           ; возврат из подпрограммы
```

Таблица должна находиться в памяти программ непосредственно за инструкцией RET, причем такая таблица может иметь до 255 элементов (1–255). Номер 0 не может быть использован, потому что во время выполнения инструкции MOVC A, @A+PC счетчик команд содержит адрес инструкции RET и значением элемента 0 будет код этой инструкции.

1.8.3. Арифметические операции

В данную группу входят 24 команды.

Команда сложения выполняет операцию сложения целых чисел длиной 1 байт, одним из операндов и приемник результата всегда яв-

ляется аккумулятор (мнемоника этой команды соответствует слову ADDition — сложение):

```
ADD A, ad ; (A)←(A)+(ad), сложение аккумулятора
          ; с прямоадресуемым байтом
```

Сложение многобайтовых целых чисел может осуществляться командой ADDC (ADDition with Carrier), которая учитывает перенос, полученный при сложении младших байтов складываемых многобайтовых чисел:

```
ADDC A, ad ; (A)←(A)+(ad)+(C), сложение аккумулятора
           ; с прямоадресуемым байтом и переносом
```

Существует также команда сложения, при помощи которой производится увеличение заданного операнда на единицу (INCrement):

```
INC A   ; (A)←(A)+1, инкремент аккумулятора
INC Rn  ; (Rn)←(Rn)+1, инкремент регистра
INC ad  ; (ad)←(ad)+1, инкремент прямоадресуемого байта
INC @Ri ; ((Ri))←((Ri))+1, инкремент байта во внутренней ПД
```

Также существует команда для работы с двумя байтами содержимого регистра указателя данных:

```
INC DPTR ; (DPTR)←(DPTR)+1, инкремент указателя данных
```

Эта команда позволяет инкрементировать содержимое указателя данных для чтения последовательности байтов из памяти программ или памяти данных.

Команда DAA предназначена для двоично-десятичных арифметических операций. Она не делает преобразования двоичного числа в двоично-десятичное, а лишь обеспечивает правильный результат при сложении двух двоично-десятичных чисел.

```
DAA ; десятичная коррекция аккумулятора
```

Команда вычисления разности существует только в варианте с вычитанием содержимого бита переноса

```
SUBB A, ad ; (A)←(A)-(C)-((ad)), вычитание из аккумулятора
           ; прямоадресуемого байта и заема
```

Мнемоника этой команды соответствует словам SUBtraction with Borrow (т. е. вычитание с учетом заёма, т. к. при вычитании образуется заём, а не перенос). При нахождении разности многобайтовых чисел необходимо очищать бит переноса, если нет уверенности в его содержимом, перед вычитанием младших байтов исходных чисел.

Команды вычитания, при помощи которых производится уменьшение указанного операнда на единицу (DECrement):

```
DEC ad  ; (ad)←(ad)-1, декремент прямоадресуемого байта
DEC A   ; (A)←(A)-1, декремент аккумулятора
DEC Rn  ; (Rn)←(Rn)-1, декремент регистра
DEC @Ri ; ((Ri))←((Ri))-1, декремент байта во внутренней ПД
```

Команда MUL AB производит умножение данных в аккумуляторе на данные, находящиеся в регистре B, помещая результат в регистры A (младшие 8 бит) и B (старшие 8 бит).

```
MUL AB ; (B)(A)←(A)*(B), умножение аккумулятора на регистр B
```

Команда DIV AB делит содержимое аккумулятора на значение в регистре B, оставляя остаток в B, а частное в аккумуляторе.

```
DIV AB ; (B)(A)←(A)/(B), деление аккумулятора на регистр B
```

По результату выполнения команд ADD, ADDC, SUBB, MUL и DIV устанавливаются флаги в регистре PSW, структура которого рассмотрена в п. 1.3.4.

Флаг C устанавливается при переносе из разряда D7, т. е. если результат не помещается в восемь разрядов; флаг AC устанавливается при переносе из разряда D3 в командах сложения и вычитания и служит для реализации десятичной арифметики. Этот признак используется командой DAA.

Флаг OV устанавливается при переносе из разряда D6, т. е. если результат не помещается в семь разрядов и восьмой не может быть ин-

терпретирован как знаковый. Этот признак служит для организации обработки чисел со знаком.

Наконец, флаг P устанавливается и сбрасывается аппаратно. Если число единичных бит в аккумуляторе нечетно, то P = 1, в противном случае P = 0.

Все арифметические инструкции выполняются за 1 машинный цикл за исключением команды INC DPTR, требующей 2 машинных цикла, а также операций умножения и деления, выполняемых за 4 машинных цикла. Любой байт во внутренней памяти данных может быть инкрементирован и декрементирован без использования аккумулятора.

Пример 1: Сложение значений ячейки с адресом 40h и аккумулятора:

```
mov r0, #40h ; запись в регистр R0 адреса ячейки в DSEG
             ; первого операнда
mov @r0, #10h; запись значения первого операнда
mov a, #30h  ; запись значения второго операнда в A
add a, 40h   ; сложение содержимого ячейки с адресом 40h
             ; с A и занесение результата в A
```

Пример 2: Сложение значений регистра R5 и аккумулятора:

```
mov r5,#10h ; запись значения первого операнда в R5
mov a, #30h ; запись значения второго операнда в A
add a, r5   ; сложение содержимого регистра R5 с A,
            ; занесение результата в A
```

1.8.4. Логические операции

В данной группе 25 команд, которые позволяют выполнять операции над байтами: логическое И ($\wedge$ — конъюнкция), логическое ИЛИ ($\vee$ — дизъюнкция), исключающее ИЛИ ($\oplus$), инверсию (NOT), сброс в нулевое значение и сдвиг.

Логические команды (команды поразрядной обработки информации) отличаются от команд арифметических операций тем, что работают с отдельными битами независимо от содержимого байта в целом.

Команды для вычисления функции И в байтовом формате используют разные способы адресации. Мнемоника команды расшифровывается как ANd Logical:

```
ANL A, Rn   ; (A)←(A)AND(Rn), логическое И аккумулятора
            ; и регистра
ANL A, @Ri  ; (A)←(A)AND((Ri)), логическое И аккумулятора
            ; и байта из внутренней ПД
ANL A, ad   ; (A)←(A)AND(ad), логическое И аккумулятора и
            ; прямоадресуемого байта
ANL A, #d   ; (A)←(A)AND #d, логическое И аккумулятора и
            ; константы
ANL ad, A   ; (ad)←(ad)AND(A), логическое И
            ; прямоадресуемого байта и аккумулятора
ANL ad, #d  ; (ad)←(ad)AND #d, логическое И прямоадресуемого
            ; байта и константы
```

Эти команды могут использоваться, например, для очистки отдельных битов двоичного кода или для проверки наличия 1 в некотором наборе битов.

Аналогичный набор команд имеется и для функции ИЛИ в байтовом формате с мнемоникой OR Logical:

```
ORL A, ad  ; (A)←(A)OR(ad), логическое ИЛИ аккумулятора и
           ; прямоадресуемого байта
ORL ad, A  ; (ad)←(ad)OR(A), логическое ИЛИ прямоадресуемого
           ; байта и аккумулятора
ORL ad, #d ; (ad)←(ad)OR #d, логическое ИЛИ прямоадресуемого
           ; байта и константы
```

Команды могут использоваться для установки отдельных битов двоичного кода в 1.

Для функции ИСКЛЮЧАЮЩЕЕ ИЛИ существуют команды только в байтовом формате. Команда расшифровывается как eXclusive oR Logical:

```
XRL A, Rn  ; (A)←(A)XOR(Rn), исключающее ИЛИ аккумулятора и
           ; регистра
XRL A, @Ri ; (A)←(A)XOR((Ri)), исключающее ИЛИ
           ; аккумулятора и байта из РПД
XRL ad, #d ; (ad)←(ad)XOR #d, исключающее ИЛИ прямоадресуемого
           ; байта и константы
```

Эти команды могут использоваться для изменения значения отдельных битов двоичного кода на обратное (toggle). Они могут также использоваться для проверки кодов на совпадение.

Для засылки нуля в накопитель проще использовать команду очистки CLR (CLeaR — очистить):

```
CLR A ; (A)←0, сброс аккумулятора
```

Для получения обратного кода (логическая функция НЕ) при байтовом формате данных предназначена команда

```
CPL A ; (A)←NOT(A), инверсия аккумулятора
```

Мнемоника этой команды ComPLement означает «дополнение», хотя в результате ее выполнения получается не дополнительный, а обратный код.

Команды циклического сдвига влево и вправо, работающие с 8 бит (Аккумулятор) или с 9 бит (Аккумулятор + бит переноса):

```
RL A  ; (A_{n+1})←(A_n), n=0÷6, (A_0)←(A_7), циклический сдвиг влево A
RLC A ; (A_{n+1})←(A_n), n=0÷6, (A_0)←(C), (C)←(A_7), циклический
      ; сдвиг влево с переносом
RR A  ; (A_n)←(A_{n+1}), n=0÷6, (A_7)←(A_0), циклический сдвиг вправо
      ; аккумулятора
RRC A ; (A_n)←(A_{n+1}), n=0÷6, (A_7)←(C), (C)←(A_0), циклический
      ; сдвиг вправо с переносом
```

Операции циклического сдвига перемещают содержимое аккумулятора на 1 бит вправо или влево. При циклическом сдвиге влево младший бит перемещается в старшую позицию. При циклическом сдвиге вправо происходит обратное. При участии бита переноса его содержимое включается в цепочку циклического переноса, что позволяет осуществлять сдвиги содержимого многобайтовых кодов.

Операция SWAP A осуществляет обмен младшей и старшей тетрад в аккумуляторе.

```
SWAP A ; (A_{0…3}) ↔ (A_{4…7}), обмен местами тетрад в аккумуляторе
```

Все логические команды не влияют на содержимое слова состояния программы за исключением случаев непосредственной записи результата в этот регистр или его биты.

Пример: устройство получило управляющий байт, содержащий несколько полей, биты 3–5 определяют номер режима работы устройства, необходимо выделить эти биты в отдельное число для последующей обработки.

```
mov a, #01101110b ; запись исходного числа в A
anl a, #00111000b ; выделение
rr a              ; сдвиг
rr a
rr a
```

1.8.5. Команды операций над битами

Микроконтроллеры семейства MCS-51 в системе команд содержат большое количество операций над битами, которые находятся в области резидентной памяти данных (128 бит, табл. 1) и в области регистров специальных функций (128 бит, табл. 2). Таким образом можно сказать, что «булевый» процессор выполняет свой набор команд, имеет свое побитово адресуемое ОЗУ и свои однобитные порты ввода-вывода (т. к. все регистры портов P0–P3 отображены в битовом пространстве).

Каждый из отдельно адресуемых битов может быть установлен в «1», сброшен в «0», инвертирован, передан в разряд C слова состояния программы PSW или принят из него. Между любым прямоадресуемым битом и флагом переноса могут быть произведены логические операции И и ИЛИ, т. е. бит переноса C используется как 1-битный аккумулятор булевого процессора.

Существует две команды, вычисляющие функцию И в битовом формате:

```
ANL C, bit ; (C)←(C)AND(bit), логическое И бита и переноса
ANL C,/bit ; (C)←(C)AND(NOT(bit)), логическое И
           ; инверсии бита и переноса
```

Две аналогичные команды, вычисляющие функцию ИЛИ в битовом формате:

```
ORL C, bit ; (C)←(C)OR(bit), логическое ИЛИ бита и переноса
ORL C,/bit ; (C)←(C)OR(NOT(bit)), логическое ИЛИ
           ; инверсии бита и переноса
```

Две команды пересылки информации в битовом формате. В команде MOV источником или приемником должен быть бит переноса C:

```
MOV C, bit ; (C)←(bit), пересылка бита в перенос
MOV bit, C ; (bit)←(C), пересылка переноса в бит
```

Для записи констант 0 и 1 используются команды очистки CLR и установки SETB (SET Bit означает «установить бит»):

```
CLR C    ; (C)←0, сброс переноса
CLR bit  ; (bit)←0, сброс бита
SETB C   ; (C)←1, установка переноса
SETB bit ; (bit)←1, установка бита
```

Две команды для получения обратного кода (логическая функция НЕ), работающие с отдельными битами:

```
CPL C   ; (C)←NOT(C), инверсия переноса
CPL bit ; (bit)←NOT(bit), инверсия бита
```

1.8.6. Команды передачи управления

Адреса операций переходов обозначаются на языке ассемблера меткой либо реальным значением в пространстве памяти программ.

Группа представлена командами безусловного и условного переходов, командами вызова подпрограмм и командами возврата из подпрограмм.

Существует три вида команды безусловного перехода — LJMP, AJMP, SJMP, — различающихся форматом адреса назначения.

Команда безусловного перехода LJMP (L — англ. *long* — «длинный») осуществляет переход по абсолютному 16-битному адресу, указанному в теле команды, т. е. команда обеспечивает переход в любую точку памяти программ.

```
LJMP ad16 ; (PC)←ad16, длинный переход в полном объеме памяти
          ; программ
```

Если необходимо сделать безусловный переход относительно адреса следующей операции в пределах 2 Кбайт, используют команду AJMP, длина которой составляет два байта, по этой причине она содержит только 11 младших бит адреса. При выполнении этой команды сначала содержимое счетчика команд увеличивается на 2 и затем записываются 11 младших разрядов адреса из кода команды.

```
AJMP ad11 ; (PC)←(PC)+2, (PC0-10) ← ad11, абсолютный переход
          ; внутри страницы в 2 Кб
```

Команда SJMP кодирует адрес как относительное смещение и занимает 2 байт. Дальность перехода относительно адреса команды, следующей за SJMP, ограничена диапазоном –128...+127 байт. Адрес назначения может располагаться в любом месте памяти программ.

```
SJMP rel     ; (PC)←(PC)+2, (PC)←(PC)+rel, короткий
             ; относительный переход внутри страницы
             ; в 256 байт
JMP @A+DPTR ; (PC)←(A)+(DPTR), косвенный относительный
             ; переход
```

Команда косвенного перехода JMP @A+DPTR позволяет вычислять адрес перехода в процессе выполнения самой программы. При помощи этой команды можно осуществлять переходы относительно адреса, указанного в регистре DPTR, по любому из 256 адресов. Поскольку адрес передачи управления зависит от содержимого аккумулятора, этот вариант команды безусловной передачи управления не является таковым. Данная команда может использоваться как переключатель, если в заданной области программы записать команды безусловного перехода на некоторые блоки программы. Поскольку эти команды будут занимать более одного байта, рассматриваемая команда может использоваться для передачи управления не более чем по 128 адресам.

Все команды условного перехода содержат короткий относительный адрес, т. е. переход относительно адреса следующей команды может осуществляться в пределах –128 ... +127 байт. Для каждого усло-

вия существует пара команд, одна из которых осуществляет передачу управления при его соблюдении, а другая — при несоблюдении. Условием передачи управления может быть равенство или неравенство нулю содержимого регистра аккумулятора:

```
JZ rel  ; (PC)←(PC)+2, если (A)=0, то (PC)←(PC)+rel,
        ; переход, если аккумулятор равен нулю
JNZ rel ; (PC)←(PC)+2, если (A)≠0, то (PC)←(PC)+rel,
        ; переход, если аккумулятор не равен нулю
```

В PSW отсутствует флаг нуля, поэтому инструкции JZ и JNZ проверяют условие «равно нулю» тестированием данных в аккумуляторе.

Можно также использовать в качестве условия перехода равенство бита переноса единице или нулю:

```
JC rel  ; (PC)←(PC)+2, если (C)=1, то (PC)←(PC)+rel,
        ; переход, если перенос равен единице
JNC rel ; (PC)←(PC)+2, если (C)=0, то (PC)←(PC)+rel,
        ; переход, если перенос равен нулю
```

Существуют команды, которые используют в качестве условия перехода равенство единице или нулю содержимого любого бита в пространстве SFR или адресуемого бита в ОЗУ:

```
JB bit, rel ; (PC)←(PC)+3, если (bit)=1, то (PC)←(PC)+rel,
            ; переход, если бит равен единице
JNB bit, rel; (PC)←(PC)+3, если (bit)=0, то (PC)←(PC)+rel,
            ; переход, если бит равен нулю
```

Команда передачи управления по равенству бита единице имеет вариант с очисткой содержимого этого бита:

```
JBC bit, rel ; (PC)←(PC)+3, если (bit)=1, то (bit)←0 и
             ; (PC)←(PC)+rel, переход, если бит
             ; установлен, с последующим сбросом бита
```

Перечисленные команды осуществляют переход в зависимости от результатов предыдущих вычислений. Однако есть управляющие

команды, которые сами осуществляют вычисления для получения условий передачи управления.

Команда DJNZ (*Decrement and Jump if Not Zero*) предназначена для организации программных циклов. Регистр Rn или байт по адресу ad, указанные в теле команды, содержат счетчик повторений цикла, а смещение rel — относительный адрес перехода к началу цикла. При выполнении команды содержимое счетчика уменьшается на 1 и проверяется на 0. Если значение содержимого счетчика не равно 0, то осуществляется переход на начало цикла, в противном случае выполняется следующая команда:

```
DJNZ Rn, rel ; (PC)←(PC)+2, (Rn)←(Rn)-1, если (Rn)≠0, то
             ; (PC)←(PC)+rel, декремент регистра
             ; и переход, если не нуль
DJNZ ad, rel ; (PC)←(PC)+2, (ad)←(ad)-1, если (ad)≠0, то
             ; (PC)←(PC)+rel, декремент
             ; прямоадресуемого байта и переход, если
             ; не нуль
```

Команда CJNE (*Compare and Jump if Not Equal* — «сравнить и перейти, если не равно»). Это единственная команда микроконтроллера, имеющая 3 операнда. Ее четыре разновидности отличаются способами адресации источника и приемника:

```
CJNE A, ad, rel ; (PC)←(PC)+3, если (A)≠(ad),
                ; то (PC)←(PC)+rel, если (A)<(ad),
                ; то (C)←1, иначе (C)←0, сравнение
                ; аккумулятора с прямоадресуемым байтом
                ; и переход, если не равно
CJNE A, #d, rel ; (PC)←(PC)+3, если (A)≠#d,
                ; то (PC)←(PC)+rel, если (A)<#d,
                ; то (C)←1, иначе (C)←0, сравнение
                ; аккумулятора с константой и переход,
                ; если не равно
CJNE Rn, #d, rel ; (PC)←(PC)+3, если (Rn)≠#d,
                 ; то (PC)←(PC)+rel, если (Rn)<#d,
                 ; то (C)←1, иначе (C)←0, сравнение
                 ; регистра с константой и переход,
```

```
                  ; если не равно
CJNE @Ri,#d, rel ; (PC)←(PC)+3, если ((Ri))≠#d, то
                  ; (PC)←(PC)+rel, если ((Ri))<#d,
                  ; то (C)←1, иначе (C)←0, сравнение байта
                  ; в РПД с константой и переход,
                  ; если не равно
```

Команда CJNE сравнивает два своих операнда как беззнаковые целые и производит переход по указанному в ней адресу, если сравниваемые операнды не равны. Если первый операнд меньше, чем второй, то бит переноса C устанавливается в 1.

Команда CJNE удобна для реализации процедур ожидания внешних событий. В теле команды указаны «координаты» двух байт и относительный адрес перехода rel. В качестве двух байтов могут быть использованы, например, значения содержимого аккумулятора и прямо адресуемого байта или косвенно адресуемого байта и константы. При выполнении команды значения указанных двух байтов сравниваются и, если они не одинаковы, осуществляется переход. Например, команда

```
WAIT: CJNE A, P0, WAIT
```

будет выполняться до тех пор, пока значения на линиях порта P0 не совпадут со значениями содержимого аккумулятора.

Пример: требуется содержимое 16 ячеек памяти начиная с адреса 3000h последовательно записать в порт P1.

Возможно несколько вариантов решения этой задачи, на рис. 16 приведены две блок-схемы. Ниже показан исходный текст программы, написанный к первому алгоритму:

```
      mov R5, #10h      ; пересылка в R5 количества
                        ; повторов цикла
      mov DPTR, #3000h  ; запись в DPTR начального адреса
lab1: movx A, @DPTR     ; пересылка в порт содержимого
                        ; ячейки памяти, адрес которой
      mov P1, A         ; находится в DPTR
      inc DPTR          ; инкрементирование значения DPTR
      djnz R5, lab1     ; декрементирование R5 и переход
                        ; на начало цикла, если R5≠0
```

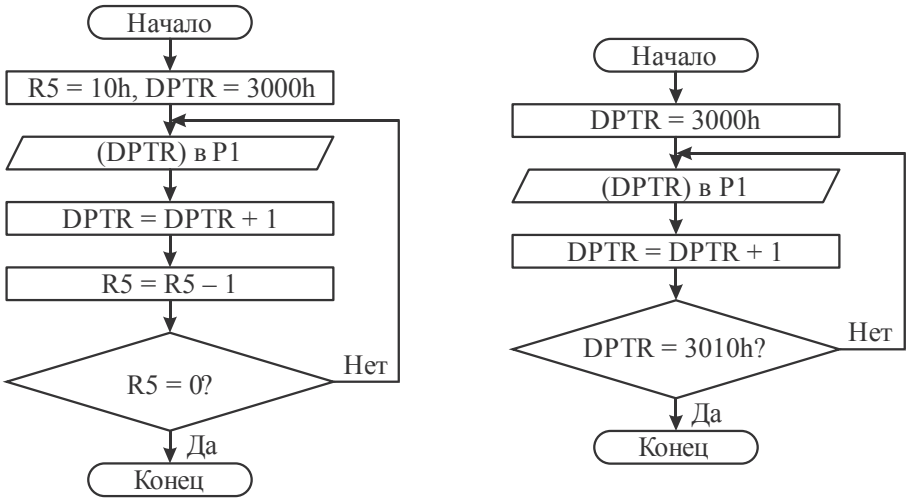


Рис. 16. Блок-схемы алгоритма записи массива в порт

Существует два вида команды вызовы подпрограммы: LCALL и ACALL. Инструкция LCALL использует 16-битный адрес вызываемой подпрограммы. В данном случае подпрограмма может быть расположена в любом месте памяти программ. Инструкция ACALL использует 11-битный адрес подпрограммы. В этом случае вызываемая подпрограмма должна быть расположена в одном 2-Кбайтном блоке с инструкцией, следующей за ACALL. Оба варианта команды записывают в стек адрес следующей команды и загружают в PC соответствующее новое значение.

```
LCALL ad16 ; (PC)←PC)+3, (SP)←(SP)+1, ((SP))←(PC0…7),
           ; (SP)←(SP)+1, ((SP))←(PC8…15), (PC)←ad16,
           ; длинный вызов подпрограммы
ACALL ad11 ; (PC)←(PC)+2, (SP)←(SP)+1, ((SP))←(PC0…7),
           ; (SP)←(SP)+1, ((SP))←(PC8…15), (PC0-10)←ad11,
           ; абсолютный вызов подпрограммы в пределах
           ; страницы в 2 Кбайт
```

Действие команд вызова подпрограмм полностью аналогично действию команд безусловного перехода. Единственное отличие команд вызова подпрограмм состоит в том, что они сохраняют в стеке адрес возврата.

Подпрограмма завершается инструкцией RET, позволяющей вернуться к инструкции, следующей за командой CALL. Эта инструкция считывает из стека адрес возврата и загружает его в PC. Инструкция RETI используется для возврата из подпрограмм обработки прерываний. Единственное отличие RETI от RET состоит в том, что RETI информирует систему о том, что обработка прерывания завершилась и разрешается прерывание обслуженного уровня. Если в момент выполнения RETI нет других прерываний, то она идентична RET.

```
RET  ; (PC8…15)←((SP)), (SP)←(SP)-1, (PC0…7)←((SP)),
     ; (SP)←(SP)-1, возврат из подпрограммы
RETI ; (PC8…15)←((SP)), (SP)←(SP)-1, (PC0…7)←((SP)),
     ; (SP)←(SP)-1, возврат из подпрограммы обработки
     ; прерывания
```

Команды RET и RETI не различают, какой командой — LCALL или ACALL — была вызвана подпрограмма, так как и в том и в другом случае в стеке сохраняется полный 16-разрядный адрес возврата.

В заключение следует отметить, что большинство трансляторов языка ассемблер допускают обобщенную мнемонику JMP для команд безусловного перехода и CALL для команд вызова подпрограмм. Конкретный тип команды определяется транслятором, исходя из «длины» перехода или вызова.

Существует одна команда, которую нельзя отнести ни к одной из групп, т. к. она не делает ничего в течение одного машинного цикла:

```
NOP; (PC)←(PC)+1, пустая операция
```

Однако эта команда (*No OPeration* — «нет операции») используется для работы в реальном масштабе времени, чтобы обеспечить кратковременную задержку перед выполнением следующей команды.

1.8.7. Общее правило расположения частей программы, размещенной в одном файле

Для удобства восприятия и анализа разрабатываемых программ, а также для корректной компиляции исходных текстов, расположение блоков небольших программ, размещаемых в одном файле, осуществляется в определенной последовательности, показанной в табл. 10.

Таблица 10

Типовое расположение блоков в программе

Блок	Пример исходного текста программы	Комментарии

1	`dpp data 84h`
	`mybyte equ 11000011b`
	`sv xdata 07h`
		Описание переменных и констант

2	`org 2003h` `ljmp prog_int0`	Адрес вектора прерывания и переход на подпрограмму обслуживания прерывания по INT0
3	`org 2050h`	Начальный адрес расположения программы во внутренней памяти программ

4	`begin: mov r0, #sv`
	`…`
	`call putbyte`
	`…`
	`jmp begin`
		Основная программа

5	`putbyte: push dpp`
	`…`
	`ret`
		Подпрограммы

6	`end`	Окончание программы

Лабораторный стенд SDK 1.1 (гл. 3) имеет встроенный загрузчик, предназначенный для инициализации всех встроенных устройств, а также выполнения некоторых других функций. Этот загрузчик располагается в области адресов 0000h–1FFFh и имеет объем 2 Кбайт. В связи с этим блок 3 (табл. 10) является обязательным, т. к. здесь определяется начальный адрес программы, которая впоследствии будет загружена в память программ стенда. В случае указания адреса меньше 2000h или отсутствия блока 3 программа загружается не полностью либо вообще не загружается. Блок 3 может предшествовать блоку 1.

Блоки 1 и 5 являются необязательными и могут отсутствовать. В блоке 1 программист указывает соответствие адресов и значений констант их символьным идентификаторам. В блоке 5 располагают подпрограммы, используемые в основной программе. В подпрограммы выделяют часто применяемую последовательность команд или какие-то большие блоки программы, например, для определения кода нажатой клавиши, вывода на жидкокристаллический индикатор (ЖКИ) числа и т. п.

В случае использования прерываний между блоками 1 и 3 располагают команды безусловных переходов по адресам векторов прерываний (блок 2).

В основной программе (блок 4) реализуется требуемый алгоритм. Программы для микроконтроллеров организуются в виде бесконечного выполнения некой части программы, например зацикленный опрос клавиатуры, с обработкой нажатий клавиш. Другой способ организации программ — применение системы прерываний. В этом случае в начальной части основной программы располагают команды инициализации всех используемых внутренних и внешних устройств, а также системы прерываний. Завершают блок 4 командой перехода на себя `jmp $`, которая заставляет микроконтроллер «зависнуть». Возможны и смешанные варианты реализации программ. Для однократного выполнения программы последней операцией также устанавливают команду `jmp $` для исключения «бесконтрольного» исполнения кода.

Последний блок, содержащий команду `end`, сообщает компилятору об остановке трансляции кода, т. е. весь последующий текст, находящийся после команды `end`, не обрабатывается.

2. Система моделирования микроконтроллера семейства MCS-51

Single-Chip Machine (SCM) представляет собой систему моделирования однокристальных микроконтроллеров (рис. 17). Существует три варианта программы [5]:

- Educational (свободная для распространения версия);
- Client (сетевая версия в сочетании с Server);
- Professional (имеет те же возможности, что и сетевая, но работает в автономном режиме).

Система моделирования Single-Chip Machine 2 предназначена:

- для моделирования работы одокристальной микроЭВМ на основе микроконтроллера семейства MCS-51 Intel 8051;
- разработки и отладки программ для микроконтроллеров Intel 8051;

- исследования поведения внутренних и внешних сигналов указанных микроконтроллеров.

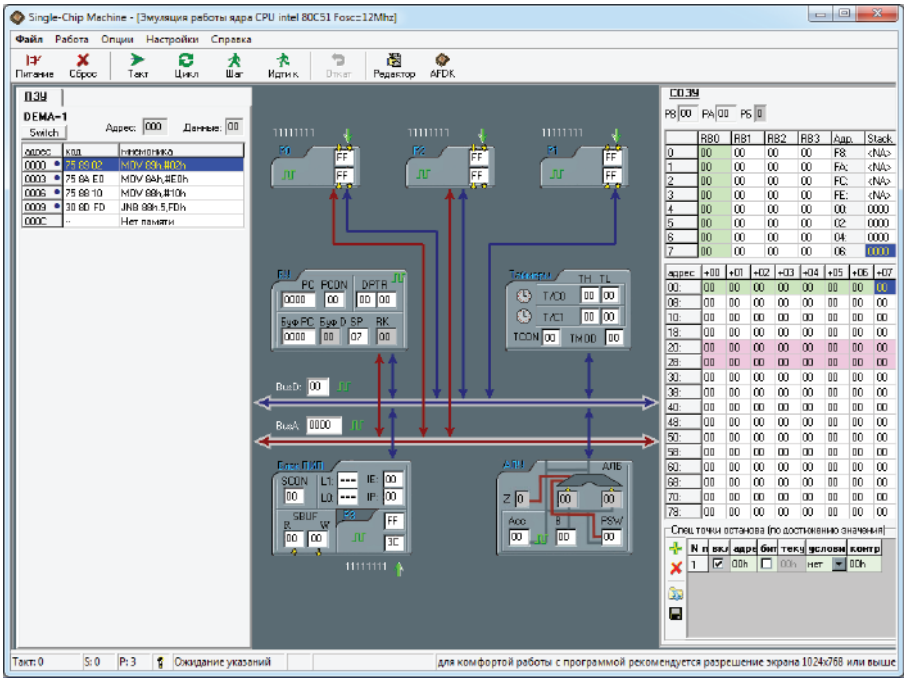


Рис. 17. Основное окно программы Single-Chip Machine 2

2.1. Запуск программы

Для запуска программы необходимо нажать кнопку Пуск, после чего в меню Программы выбрать папку с заданным при инсталляции именем (Single-Chip Machine 2 по умолчанию). В данной папке находятся два ярлыка: Single-Chip Machine Help system для вызова справочной системы и Single-Chip Machine 2 для исполняемого модуля. Запуск программы осуществляется выбором ярлыка Single-Chip Machine 2.

2.2. Основные функциональные возможности программы

Программа SCM выполнена в виде независимого запускаемого модуля, работоспособного под управлением операционных систем Microsoft Window 95/98/2000/NT/XP/7. SCM включает средства отлад-

ки и редактирования программ на ассемблере. Выполнение программы пользователя осуществляется с максимальным приближением к действительности с помощью имитационной модели. Кроме того, пользователю предоставляется такие средства, как временные диаграммы внутренних и внешних сигналов, имитация внешних сигналов, возможность изменения значений узлов микроконтроллера в процессе работы модели и др.

2.2.1. Редактор-компилятор

Ввод текста программы осуществляется во встроенном редакторе-компиляторе, вызов которого производится нажатием кнопки Редактор (), расположенной на панели инструментов (см. рис. 17) основного окна программы.

Для создания новой программы в редакторе-компиляторе (рис. 18) необходимо нажать кнопку Новая программа () на панели инструментов или при помощи меню Файл — Новый (Ctrl+N). ! По мере написания исходного текста программы, во избежание его потери, рекомендуется регулярно сохранять текст программы. Для этого достаточно нажать кнопку Сохранить текущий файл () или при помощи меню Файл — Сохранить (F2), если исходный текст сохраняется впервые, то появится диалоговое окно, в котором необходимо указать место расположения и имя создаваемого файла.

После разработки и написания программы в редакторе-компиляторе ее необходимо скомпилировать, т. е. преобразовать в машинный код. Для компиляции, а также поиска синтактических ошибок следует нажать кнопку Скомпилировать (). При обнаружении синтактических ошибок в исходном тексте программы на экране появится информационное окно с соответствующим текстом, а в нижней части редактора на закладке «Ошибки» — список всех допущенных ошибок. При выборе ошибки в списке происходит автоматическое выделение соответствующей строки в редакторе. После исправления всех ошибок программу можно запускать, для этого требуется нажать кнопку Выполнить программу () или при помощи меню Компиляция — Выполнить (F9). В результате редактор-компилятор закроется, и текст скомпилированной программы окажется в основном окне SCM (см. рис. 17).

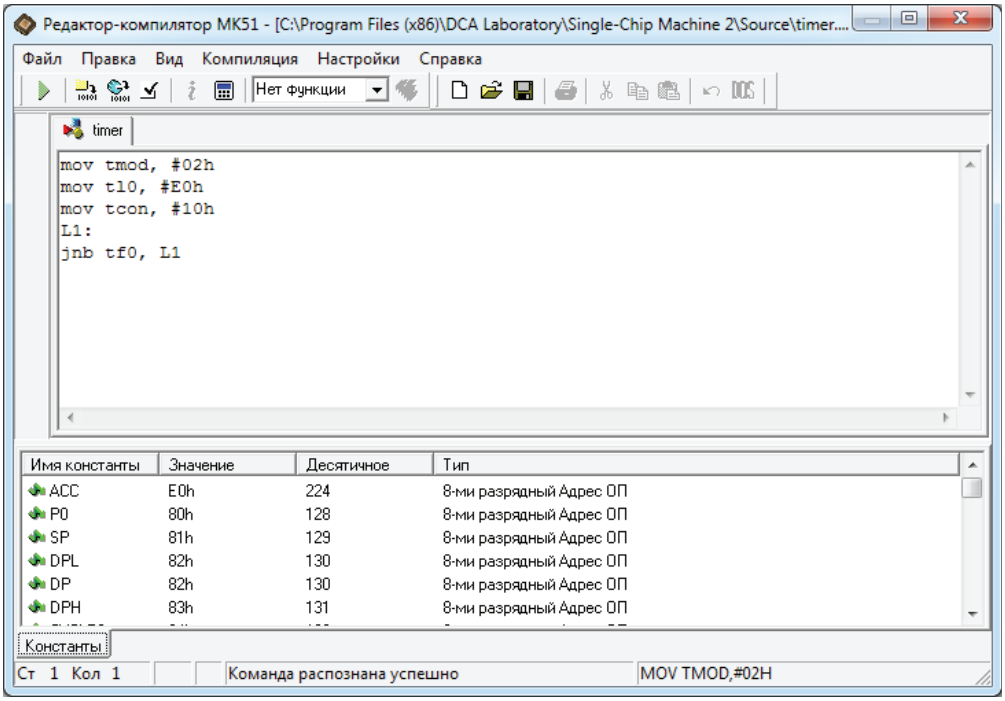


Рис. 18. Окно редактора-компилятора

Редактор-компилятор позволяет сохранять скомпилированный текст разработанной программы в формате Intel HEX, расширение hex. Intel HEX — формат файла, предназначенного для представления произвольных двоичных данных в текстовом виде, является стандартом де-факто при программировании памяти микроконтроллеров, ПЗУ, EEPROM и т. д.

2.2.2. Основное окно

Основное окно программы SCM (см. рис. 17) включает в себя главное меню, панель инструментов и основную рабочую область. В левой части основной области отображается содержимое памяти программ, в правой — содержимое внутренней памяти данных, а в центральной — имитационная модель микроконтроллера Intel 8051, которая дает наглядное представление структуры микроконтроллера семейства MCS-51 и включает в себя основные блоки:

- арифметико-логическое устройство (АЛУ), состоящее из аккумулятора, регистра B, регистра слова состояния программы (PSW) и внутренних регистров T1 и T2;
- блок управления (БУ), включающий в себя счетчик команд (PC), 16-битный регистр-указатель адреса (DPTR), регистр управления мощностью (PCON) и др.;
- 4 параллельных двунаправленных порта ввода-вывода (P0, P1, P2 и P3);
- блок последовательного интерфейса и прерываний (ПИП), содержащий регистры управления последовательным интерфейсом UART (SCON и SBUF), регистры управления системой прерываний (IE и IP) и порт ввода-вывода P3;
- блок таймер-счетчиков, состоящий из регистров управления 16-битными таймер-счетчиками.

В правой части основного окна располагается содержимое внутренней памяти данных (СОЗУ). Здесь в верхней таблице показана область регистров общего назначение (RSEG), в центральной — всей внутренней памяти данных (DSEG); в нижней — можно указывать точки останова выполнения пользовательской программы по условию.

Имитационная модель микроконтроллера Intel 8051, реализованная в программе SCM, предусматривает несколько вариантов выполнения пользовательской программы:

- такт (кнопка 

на панели инструментов) — продвижение выполнения программы на один такт;
- цикл (кнопка 

на панели инструментов) — продвижение программы на один машинный цикл;
- шаг (кнопка 

на панели инструментов) — выполнение одной команды программы;
- запуск программы до отмеченной команды (кнопка 

на панели инструментов). Для отметки команды остановки выполнения программы необходимо нажать левую кнопку мыши на требуемой команде в области памяти программ (ПЗУ) основного окна SCM, тогда выделяемая команда будет помечена красным маркером.

Перезапуск имитационной модели осуществляется при помощи кнопки Сброс. Программа SCM позволяет вернуться на один шаг назад выполняемой программы, для чего следует нажать кнопку Откат.

Все действия, касающиеся выполнения программы, представлены не только на панели инструментов, но и в меню Работа. Кроме того, в меню Опции указываются свойства выполнения и отката программы.

Основные настройки программы можно изменять при помощи меню Настройки, а также подпункта этого меню Параметры.

3. Лабораторный комплекс SDK-1.1

Учебный лабораторный комплекс SDK-1.1 предназначен для освоения архитектуры и методов проектирования [6]:

- систем на базе микропроцессоров и однокристальных микроЭВМ;
- встраиваемых контроллеров и систем сбора данных;
- периферийных блоков вычислительных систем;
- подсистем ввода-вывода встраиваемых систем.

В состав учебного стенда входят (рис. 19):

- микроконтроллер ADuC812 или ADuC842;
- внешняя E^2PROM объемом 256 байт;
- клавиатура AK1604A-WWB фирмы ACCORD;
- жидкокристаллический индикатор (ЖКИ) WH1602B-YGK-CP фирмы Winstar Display;
- часы реального времени PCF8583;
- 128 Кбайт внешней SRAM с возможностью расширения до 512 Кбайт;
- набор сигнальных светодиодов.

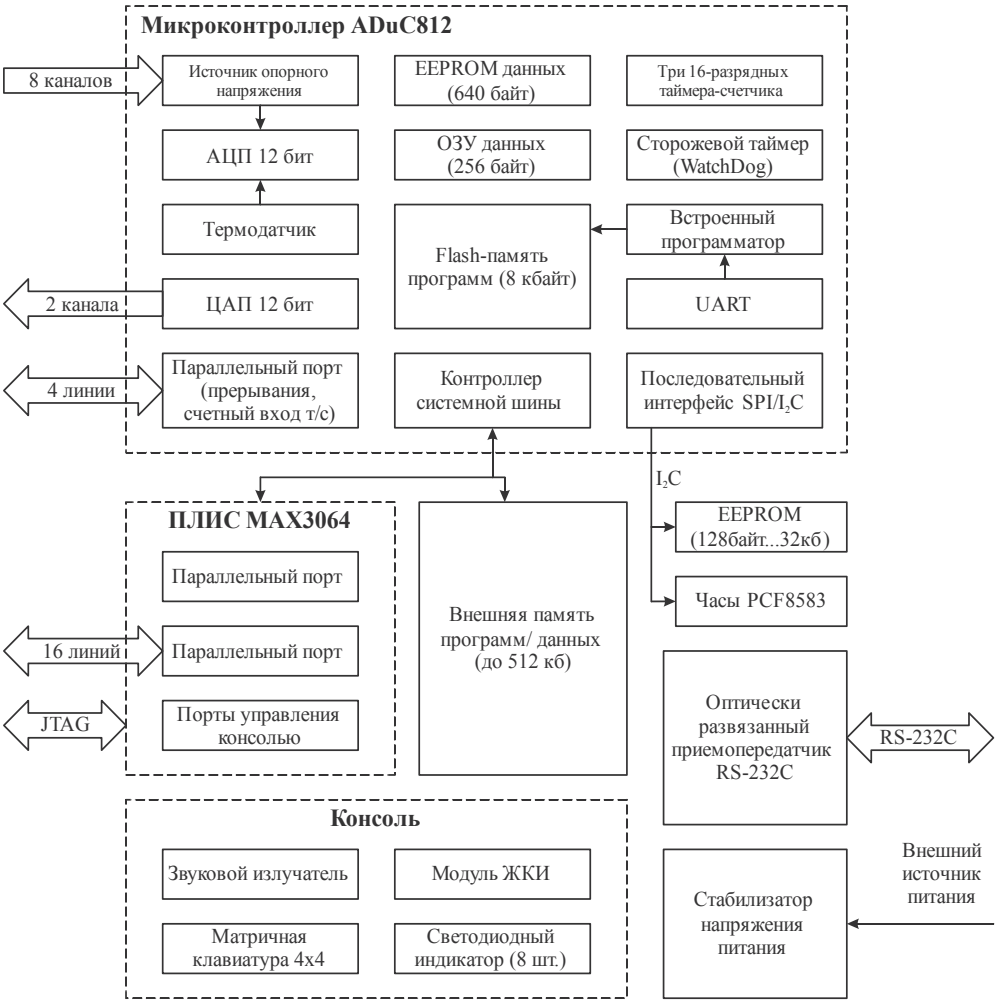


Рис. 19. Структура аппаратной части учебного стенда SDK-1.1

Далее рассмотрены основные составляющие компоненты лабораторного стенда SDK-1.1.

3.1. Микроконтроллер стенда SDK-1.1

В качестве главного элемента управления стенда используются микроконтроллеры ADuC812 или ADuC842, которые являются клонами Intel 8051 со встроенной периферией (рис. 20), основной

функционал этих микросхем совпадает за исключением некоторых особенностей.

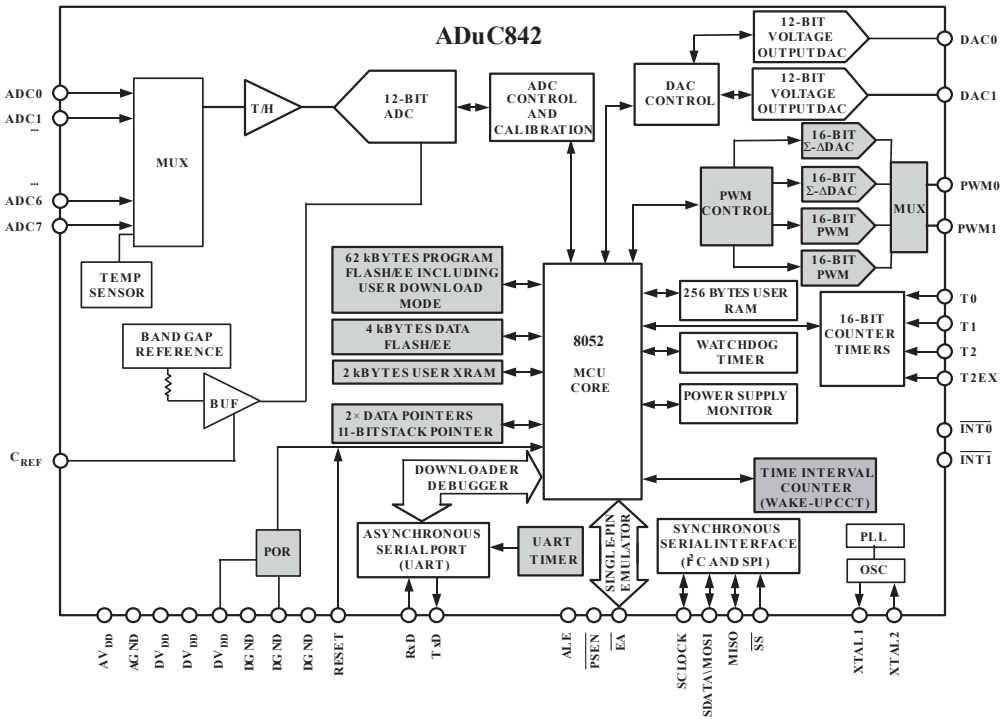


Рис. 20. Функциональная блок-схема микроконтроллера ADuC842 (серым обозначены блоки, отсутствующие в микроконтроллере ADuC812)

Основные характеристики для ADuC812 [7, 8]:

- рабочая частота 11,0592 МГц;
- 8-канальный 12-битный прецизионный АЦП с самокалибровкой со скоростью выборок до 200 К/с (в режиме прямого доступа к памяти — ПДП);
- два 12-битных ЦАП (код — напряжение);
- внутренний температурный сенсор;
- 640 байт программируемого E^2PROM со страничной организацией (256 страниц по 4 байт);
- 256 байт внутренней памяти данных;
- адресное пространство памяти данных 16 Мбайт;
- режим управления питанием;
- асинхронный последовательный порт ввода-вывода (RS-232C);

- интерфейсы SPI, I^2C;
- три 16-битных таймер-счетчика и таймер WatchDog;
- питание от источника напряжения +3 В или +5 В (отдельно для аналоговой части и отдельно для цифровой).

Микросхема ADuC842 по сравнению с ADuC812 имеет увеличенный объем резидентной памяти программ и памяти данных, встроенный программно управляемый широтно-импульсный модулятор, временной таймер (для подсчета больших интервалов времени 1/128 с — 255 ч), специальный дополнительный таймер для последовательного порта UART и схему запуска POR (Power-on Reset).

Всеми встроенными дополнительными устройствами можно управлять при помощи регистров специального назначения, адреса и значения по умолчанию приведены в прил. 2.

При обращении к внешней памяти данных используется 24-битовый адрес, младший разряды передаются через P0, а старший и средний байты адреса — через порт P2. Для обращения к внешней памяти данных используется регистр DPTR — регистр-указатель данных и DPP — указатель страницы данных.

3.1.1. Блок АЦП

Блок АЦП включает в себя 8-канальный 5-микросекундный аналого-цифровой преобразователь с однополярным питанием, который содержит многоканальный мультиплексор, устройство выборки-хранения, встроенный источник опорного напряжения (ИОН), систему калибровок и собственно АЦП. АЦП построен по схеме стандартного конвертора последовательного приближения. Источник опорного напряжения (ИОН) откалиброван на напряжение 2,5 В. Также может использоваться внешний ИОН с напряжением 2,3 В — AV_{dd}.

Все компоненты блока управляются при помощи трех интерфейсных регистров специального назначения: ADCCON1, ADCCON2 и ADCCON3. Выходной код АЦП хранится в регистрах ADCDATAH и ADCDATAL. Для указания адреса в режиме прямого доступа к памяти (ПДП) используются регистры DMAL, DMAH и DMAP. Адреса регистров указаны в прил. 2.

Регистр управления АЦП ADCCON1

Управление преобразованием, временем переключения, режимами преобразования и потреблением энергии в режиме управления АЦП ADCCON1 осуществляется программным путем. Для этого необходимо задать соответствующее управляющее слово в регистр ADCCON1, функциональное назначение бит которого отражено ниже:

ADCCON1.7	ADCCON1.6	ADCCON1.5	ADCCON1.4	ADCCON1.3	ADCCON1.2	ADCCON1.1	ADCCON1.0
MD1	MD0	CK1	CK0	AQ1	AQ0	T2C	EXC

Поля регистра ADCCON1:

- MD1, MD0 — биты, определяющие режим АЦП (табл. 11);

Таблица 11

Режим работы АЦП

MD1	MD0	Режим
0	0	Отключен
0	1	Нормальный
1	0	Дежурный, если не выполняется цикл преобразования
1	1	Холостой, если не выполняется цикл преобразования

- CK1, CK0 — биты, определяющие коэффициент деления тактовой частоты процессора для получения тактовой частоты АЦП. Получение одной выборки АЦП занимает 16 тактов (табл. 12);

Таблица 12

Делитель тактовой частоты для АЦП

CK1	CK0	Делитель
0	0	1
0	1	2
1	0	4
1	1	8

- AQ1, AQ0 — биты задержки переключения, выбирающие время, которое необходимо для перезарядки устройства выборки-хранения (УВХ) при переключении мультиплексора (табл. 13);

Таблица 13

Задержка запуска АЦП

AQ1	AQ0	Число задержки запуска АЦП
0	0	1
0	1	2
1	0	3
1	1	4

- T2C — бит запуска преобразования от таймера 2. Если бит установлен, то флаг переполнения T2 используется для запуска преобразования;
- EXC — бит разрешения внешнего запуска. Если он установлен, то вывод микроконтроллера 23 (CONVST) будет использован как сигнал запуска.

Регистр управления АЦП ADCCON2

Регистр ADCCON2 управляет выбором номера канала и режимами преобразования:

ADCCON2.7	ADCCON2.6	ADCCON2.5	ADCCON2.4	ADCCON2.3	ADCCON2.2	ADCCON2.1	ADCCON2.0
ADCI	DMA	CCONV	SCONV	CS3	CS2	CS1	CS0

Поля регистра ADCCON2:

- ADCI — бит прерывания АЦП, устанавливается аппаратным способом после окончания работы однократного цикла преобразования АЦП или по окончании передачи блока в режиме ПДП. ADCI очищается аппаратно при переходе по вектору на подпрограмму обслуживания прерывания;
- DMA — бит разрешения режима ПДП, устанавливается пользователем для начала операции ПДП со стороны АЦП;
- CCONV — бит циклического преобразования, устанавливается пользователем для перевода АЦП в режим непрерывного циклического преобразования. В этом режиме АЦП выполняет преобразование в соответствии с типом синхронизации и конфигурацией каналов, которые выбраны в других регистрах управления АЦП;
- SCONV — бит запуска однократного преобразования, устанавливается пользователем для однократного запуска АЦП. Бит сбрасывается автоматически по завершении преобразования;

- CS3, CS2, CS1, CS0 — биты выбора входных каналов, позволяют осуществить выбор номера канала АЦП под управлением программы (табл. 14). В режиме ПДП выбор номера канала осуществляется из ID-канала, записанного во внешней памяти.

Таблица 14

Выбор входных каналов АЦП

CS3	CS2	CS1	CS0	Режим
0	n2	n1	n0	Номер входного канала n2n1n0
1	0	0	0	Температурный сенсор
1	1	1	1	Останов ПДП

Регистр управления АЦП ADCCON3

Регистр управления АЦП ADCCON3 дает индикацию занятости АЦП для прикладных программ.

ADCCON3.7	ADCCON3.6	ADCCON3.5	ADCCON3.4	ADCCON3.3	ADCCON3.2	ADCCON3.1	ADCCON3.0
BUSY	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD

Поля регистра ADCCON3:

- BUSY — бит занятости АЦП, только для чтения. Устанавливается на время преобразования или калибровки АЦП. Автоматически очищается в конце циклов преобразования или калибровки;
- остальные биты RSVD зарезервированы, при чтении возвращают 0, их следует записывать только нулями.

Режимы работы АЦП

После настройки АЦП при помощи регистров ADCCON1–3 и запуска преобразования в выходных регистрах АЦП ADCDATAH и ADCDATAL начинает появляться 12-разрядный код, соответствующий напряжению на входе МК. В четырех старших разрядах ADCDATAH указан номер канала, с которого был преобразован сигнал:

ADCDATAH								ADCDATAL							
Номер канала				Старшие разряды результата				Младшие разряды результата							
D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0

Если для требуемой обработки сигнала, поступившего с АЦП, необходимо большее время, чем пауза между преобразованием, то используется режим ПДП. До включения режима ПДП размечают область внешней памяти данных, в которую будут сохраняться выборки. Разметка состоит в записи идентификаторов номеров каналов ID (четыре старших разряда) во внешней памяти (табл. 15).

Таблица 15

Разметка внешней памяти для режима ПДП

000000H	0	0	1	0					Преобразовывать канал 2
	0	1	0	1					Преобразовывать канал 5
	1	0	0	0					Преобразовывать температурный сенсор
	0	1	0	0					Преобразовывать канал 4
	0	1	0	0					Преобразовывать канал 4
00000AH	1	1	1	1					Остановка

После разметки указывается стартовый адрес блока во внешней памяти. Значение адреса заносится в регистры в следующем порядке: DMAL (*low* — младшая часть адреса), DMAH (*high* — старшая) и DMAP (*page* — страница), например, 000000 H (табл. 15). Окончание таблицы ПДП обозначается записью «1 1 1 1» в поле выбора канала. Перед запуском преобразования необходимо сконфигурировать регистры управления АЦП, выбрав необходимые параметры преобразования. Запуск АЦП в режиме ПДП осуществляется установкой бита разрешения (ADCCON2.6, DMA). По окончании преобразования установится бит прерывания АЦП ADCI (ADCCON2.7), и во внешней памяти данных будут доступны результаты работы АЦП в соответствии с предварительной разметкой памяти, как показано в табл. 16. При этом результаты разметки сохранятся.

Таблица 16

Содержание внешней памяти после окончания преобразования

000000H	0	0	1	0	x	x	x	x	Канал 2
	x	x	x	x	x	x	x	x	x — результат преобразования канала 2

Окончание табл. 16

	0	1	0	1	x	x	x	x	Канал 5
	x	x	x	x	x	x	x	x	x — результат преобразования канала 5
	1	0	0	0	x	x	x	x	Температурный сенсор
	x	x	x	x	x	x	x	x	x — результат преобразования сенсора
	0	1	0	0	x	x	x	x	Канал 4
	x	x	x	x	x	x	x	x	x — результат преобразования канала 4
	0	1	0	0	x	x	x	x	Канал 4
	x	x	x	x	x	x	x	x	x — результат преобразования канала 4
00000AH	1	1	1	1	x	x	x	x	Остановка

3.1.2. Блок ЦАП

ADuC812 на кристалле содержит два 12-разрядных цифроаналоговых преобразователя. Один SFR управления и четыре SFR данных позволяют управлять работой ЦАП:

- DAC0L/DAC1L — содержат 8 младших разрядов данных ЦАП, адреса регистров в пространстве SFR соответственно F9h и FBh;
- DAC0H/DAC1H — содержат 4 старших разрядов данных ЦАП, адреса регистров соответственно FAh и FCh;
- DACCON — содержат биты управления ЦАП общего назначения, адрес регистра FDh.

Изменение выходного напряжения каждого канала ЦАП производится после записи значения младшей части входного кода в соответствующий регистр DAC0L/DAC1L. Модуль ЦАП позволяет одновременно устанавливать выходное напряжения на обоих каналах. Для этого используется бит SYNC регистра DACCON. В случае использования 8-разрядного режима работы число, записанное в регистры DACxL, переносится в верхнюю часть 12-разрядного регистра данных ЦАП.

Структура регистра DACCON:

DACCON.7	DACCON.6	DACCON.5	DACCON.4	DACCON.3	DACCON.2	DACCON.1	DACCON.0
MODE	RNG1	RNG0	CLR1	CLR0	SYNC	PD1	PD0

Поля регистра DACCON:

- MODE — бит устанавливает режим работы обоих ЦАП. Если MODE равен 1, то 8-разрядный режим (запись восьми битов в DACxL SFR). Если равен 0, то 12-разрядный;
- RNG1 — бит выбора диапазона ЦАП1. Если равен 1, то диапазон ЦАП1 0–V_{dd}. Если равен 0, то диапазон ЦАП1 0–V_{ref};
- RNG0 — бит выбора диапазона ЦАП0. Если равен 1, то диапазон ЦАП0 0–V_{dd}. Если равен 0, то диапазон ЦАП0 0–V_{ref};
- CLR1 — бит очистки ЦАП1. Если равен 1, то выход ЦАП1 соответствует коду. Если равен 0, то выход ЦАП1 равен 0 В;
- CLR0 — бит очистки ЦАП0. Если равен 1, то выход ЦАП0 соответствует коду. Если равен 0, то выход ЦАП0 равен 0 В;
- SYNC — бит синхронизации ЦАП0/1. Если равен 1, то выходы ЦАП изменяются, как только данные попадают в регистры DACxL SFR. Можно одновременно обновить выходы обоих ЦАП путем предварительной записи данных в DACxL/H при SYNC = 0. Выходы ЦАП одновременно обновятся теперь при установке SYNC в 1;
- PD1 — бит выключения ЦАП1. Если равен 1, то ЦАП1 включен. Если равен 0, то ЦАП1 выключен;
- PD0 — бит выключения ЦАП0. Если равен 1, то ЦАП0 включен. Если равен 0, то ЦАП0 выключен.

3.1.3. Система прерываний

ADuC812 обеспечивает девять источников и два уровня прерываний. На рис. 21 приведена структурная схема системы прерываний микроконтроллера ADuC812. Здесь прерывания каждого уровня расположены в порядке убывания приоритета, также показаны все источники прерываний, флаги запросов и управления.

Инициализация и работа системы прерываний осуществляется так же, как и в базовой версии микроконтроллера семейства MCS-51 (см. 1.7.1, с. 39).

Для разрешения и установки приоритета различных прерываний используются три регистра SFR. Регистры IE и IP имеют битовую адресацию, регистр IE2 — адресуется только как байт.

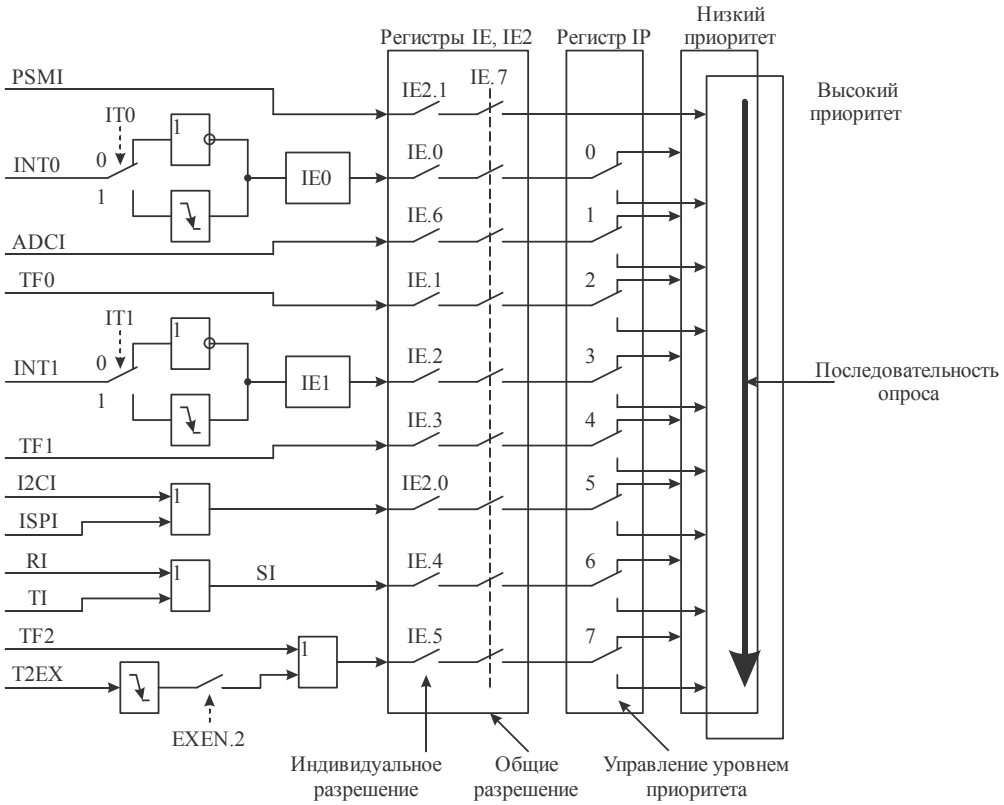


Рис. 21. Система управления прерываниями

Назначение бит регистра IE следующее:

IE.7	IE.6	IE.5	IE.4	IE.3	IE.2	IE.1	IE.0
EA	EADC	ET2	ES	ET1	EX1	ET0	EX0

Поля регистра IE:

- EADC — разрешение прерывания АЦП;
- ET2 — разрешение прерывания по переполнению таймер-счетчика 2.

Биты регистра IE.0 — IE.4 и IE.7 полностью совпадают с разрядами регистра IE базовой версии микроконтроллера (см. п. 1.7, с. 34).

Назначение бит регистра IE2 следующее:

IE2.7	IE2.6	IE2.5	IE2.4	IE2.3	IE2.2	IE2.1	IE2.0
						EPSM	ESI

Поля регистра IE2:

- EPSM — разрешение прерывания по монитору питания;
- ESI — разрешение прерывания от интерфейсов SPI/I2C.

Регистр IP устанавливает один из двух возможных уровней прерывания для различных источников прерывания. Установите соответствующий бит в 1 для присвоения высокого уровня данному прерыванию или 0 для низкого.

Назначение бит регистра IP:

IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
PSI	PADC	PT2	PS	PT1	PX1	PT0	PX1

Поля регистра IP:

- PSI — приоритет прерывания интерфейсов SPI/I2C;
- PADC — приоритет прерывания АЦП;
- PT0 — приоритет таймер-счетчика 2.

Биты PS, PT_X и PX_X полностью совпадают с одноименными разрядами регистра IP базовой версии микроконтроллера (см. п. 1.7, с. 35).

Адреса векторов прерываний микроконтроллера ADuC812 располагаются в начальной области памяти программ в диапазоне 0003h–0043h (табл. 17). На стенде SDK 1.1 в этой области расположен загрузчик (адреса 0000h–1FFFh). Пользователь не имеет возможности записывать программы в этот диапазон, поэтому разработчики стенда разместили по адресам векторов прерываний (0003h–0043h) безусловные переходы на соответствующие адреса в диапазоне 2003h–2043h. Таким образом, для организации обработки прерываний необходимо размещать подпрограммы обслуживания или команды безусловных переходов в области 2003h–2043h. Стенд SDK 1.1, оснащенный микроконтроллером ADuC842, имеет некоторые отличия, поэтому для организации обработки прерываний в области 2003h–2043h требуется использовать команды безусловного перехода LJMP.

Таблица 17

Адреса векторов прерываний микроконтроллера ADuC812

Прерывание	Наименование	Адрес вектора	Последовательность опроса
PSMI	Источник питания ADuC812	43h	1
IE0	Внешнее прерывание INT0	03h	2
ADCI	Конец преобразования АЦП	33h	3

Окончание табл. 17

Прерывание	Наименование	Адрес вектора	Последовательность опроса
TF0	Переполнение таймер-счетчика 0	0Bh	4
IE1	Внешнее прерывание INT1	13h	5
TF1	Переполнение таймер-счетчика 1	1Bh	6
I2CI/ISPI	Прерывание последовательного интерфейса (I^2C, SPI)	3Bh	7
RI/TI	Прерывание UART	23h	8
TF2/EXF2	Переполнение таймер-счетчика 2	2Bh	9

3.1.4. Порты ввода-вывода

Параллельные порты ввода-вывода

Для обмена с внешними устройствами в составе ADuC812 присутствуют четыре порта общего назначения. В дополнение к функции общего ввода-вывода, некоторые порты могут управлять взаимодействием с внешней памятью, в то время как другие мультиплексируются альтернативными функциями для периферии. В общем случае, когда альтернативная функция для контакта порта разрешена, данный контакт не может употребляться в качестве бита порта ввода-вывода общего назначения.

Порты 0, 2 и 3 двунаправленные, порт 1 служит только для ввода. Все порты содержат выходную защелку и входной буфер, порты ввода-вывода содержат также выходной буфер. Доступ к контактам портов 0–3 для чтения и записи осуществляется через регистры специального назначения P0, P1, P2 и P3 соответственно. Контакты портов 0, 2 и 3 можно конфигурировать независимо как для цифрового ввода, так и для вывода через соответствующие биты SFR. Контакты порта 1 можно конфигурировать только либо на цифровой ввод, либо на аналоговый; возможность цифрового вывода по порту 1 не поддерживается.

Асинхронный интерфейс (UART)

Последовательный порт асинхронного интерфейса UART полнодуплексный, что означает возможность одновременной передачи и приема. Имеется буфер приема, что подразумевает возможность приема второго байта до считывания из регистра приемника предыдущего.

Физический интерфейс к сети последовательных данных подключается через контакты RxD (P3.0) и TxD (P3.1), а сам порт можно конфигурировать на четыре режима работы. Режимы работы интерфейса соответствуют базовой модели микроконтроллеров семейства MCS-51 (п. 1.6, с. 28).

Интерфейс SPI

Интерфейс SPI (*Serial Peripheral Interface,* или *Microwire*) является промышленным стандартным интерфейсом синхронного последовательного обмена, допускающим одновременную синхронную передачу и прием 8 бит данных. Применяется для сопряжения микроконтроллеров с периферийными цифровыми микросхемами, устройствами и другими микроконтроллерами. За каждый такт синхронизации одновременно может и передаваться, и приниматься очередной бит данных. В основном варианте использования интерфейса предполагается, что соединяются одно ведущее устройство (Master) с одним или несколькими ведомыми устройствами (Slave).

Соединение устройств с использованием интерфейса SPI осуществляется при помощи следующих сигналов:

SCLK (*Serial Clock*) — синхросигнал, которым ведущее устройство стробирует каждый бит данных;

MOSI (*Master Output Slave Input*) — выходные данные ведущего устройства и входные данные ведомого устройства;

MISO (*Master Input Slave Output*) — входные данные ведущего устройства и выходные данные ведомого устройства;

SS (*Slave Select*) или CS (*Chip Select*) — выбор ведомого устройства, осуществляется низким уровнем. При соединении ведущего и одного ведомого эта связь между микросхемами может быть исключена путем жесткой установки низкого уровня на входе SS ведомого устройства (рис. 22). Тем не менее для некоторых типов соединяемых устройств и режимов работы эта линия связи может потребоваться, в частности, активный уровень на этом входе может использоваться для начала обмена информацией.

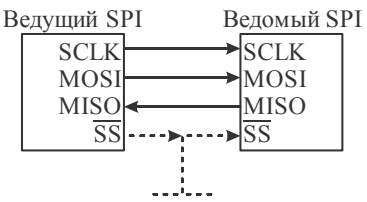


Рис. 22. Соединение ведущего и одного ведомого

Для подключения нескольких ведомых существует два варианта реализации формирования сигнала SS ведущим устройством: раздельно

для каждого ведомого (так называемое независимое, или параллельное, подключение) и одного для всех ведомых (каскадное, или последовательное подключение), последний вариант используется крайне редко. При независимом подключении (рис. 23) взаимодействие с конкретным ведомым устройством будет производиться только при низком уровне сигнала SS, а при высоком уровне выход MISO должен переводиться в высокоимпедансное состояние. При этом получается гибридная топология соединений: по сигналам SCK, MOSI и MISO топология шинная, по SS — звездообразная (центр — ведущее устройство).

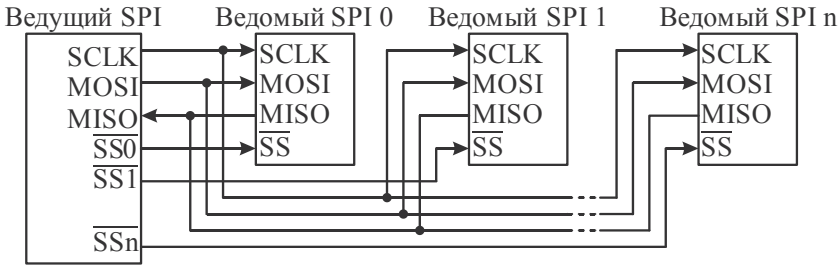


Рис. 23. Соединение ведущего и нескольких ведомых при помощи интерфейса SPI

Интерфейс SPI поддерживает 4 режима обмена (SPI Mode 0 – SPI Mode 3), отличающихся фазой (CPHA — *clock phase*) и полярностью синхросигналов (CPOL — *clock polarity*) (рис. 24). Микроконтроллеры с аппаратной поддержкой интерфейса SPI обычно имеют возможность программного выбора режима.

Форматы транзакций зависят от устройств, но общая идея такова: ведущее устройство по линии MOSI посылает код операции и адресную информацию для устройства, затем следуют данные. В операции записи они передаются ведущим устройством по той же линии, в операции чтения устройство их посылает по линии MISO. На время всей транзакции ведущее устройство сохраняет активное состояние линии SS, число требуемых синхроимпульсов зависит от формата команды.

Основные преимущества интерфейса SPI:

- простота программной и аппаратной реализации обуславливает высокую надежность и быстродействие передачи;
- полнодуплексная передача данных по умолчанию;
- возможность произвольного выбора длины пакета, длина пакета не ограничена восемью битами;

- максимальная тактовая частота ограничена только быстродействием устройств, участвующих в обмене данными;
- все линии шины SPI являются однонаправленными, что существенно упрощает решение задачи преобразования уровней и гальванической изоляции микросхем.

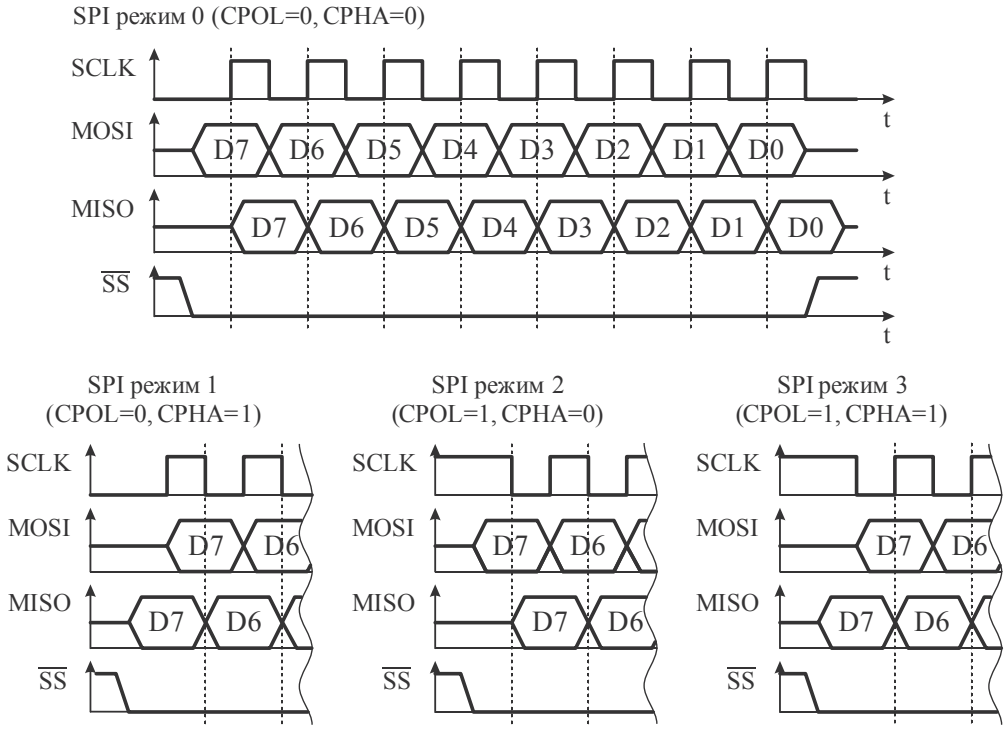


Рис. 24. Последовательность передачи по интерфейсу SPI
(вертикальная пунктирная линия определяет момент выборки информации считывающим устройством)

Основные недостатки интерфейса SPI:

- нет подтверждения приема данных со стороны ведомого устройства (ведущее устройство может передавать данные «в никуда»);
- нет определенного стандартом протокола обнаружения ошибок;
- ведомое устройство не может управлять потоком данных;
- есть множество вариантов реализации интерфейса;
- по дальности передачи данных интерфейс SPI уступает таким стандартам, как UART и CAN.

Интерфейс I²C

Шина I²C (*Inter-Integrated Circuit*) — последовательная шина данных с шинной топологией соединения. Применяется для связи интегральных схем при помощи двух линий:

SDA (*Serial DAta*) — последовательная линия данных (SDATA);

SCL (*Serial CLock*) — линии синхронизации (SCLOCK).

Обе линии являются двунаправленными, т. е. ведущий и ведомый могут быть как приемником, так и передатчиком.

В основном варианте использования интерфейс соединяет одно ведущее устройство с одним или несколькими ведомыми устройствами (рис. 25). Стандарт допускает подключение нескольких ведущих к одной шине I²C, описывает определение ошибок при одновременном обмене с участием нескольких ведущих (арбитраж). В стандартном режиме обеспечивается передача последовательных 8-битных данных со скоростью до 100 кбит/с, до 400 кбит/с в «быстром» режиме. Вторая версия стандарта увеличила скорость передачи до 3,4 Мбит/с.

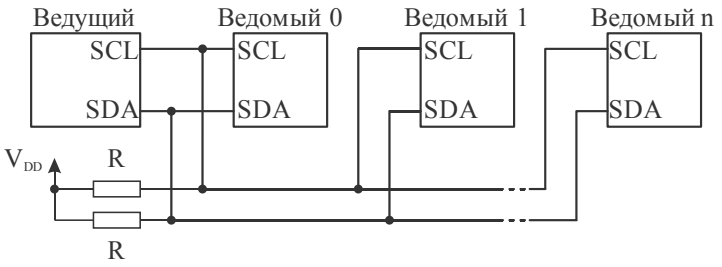


Рис. 25. Соединение ведущего и нескольких ведомых при помощи интерфейса I²C

Типичная последовательность передачи данных по интерфейсу I²C, показанная на рис. 26, начинается с условия START, которое характеризуется переходом линии SDATA из 1 в 0, при этом на линии SCLOCK удерживается высокий уровень. Затем по линии SDATA ведущий передает байт информации, 7 старших бит содержат адрес ведомого, 8-й бит (младший) является разрядом чтения/записи (R/W), который определяет направление передачи. Если бит R/W очищен, то ведущий будет передавать данные выбранному ведомому устройству, если установлен — ожидает передачу информации от ведомого. Когда адрес ведомого устройства совпадает с запрашиваемым, оно возвращает сигнал подтверждения приема ACK (по линии SDATA низкий уровень), иначе оно посылает сигнал отказа NACK (высокий уровень линии SDATA). Все передаваемые сигналы по SDATA тактируются по линии SCLOCK.

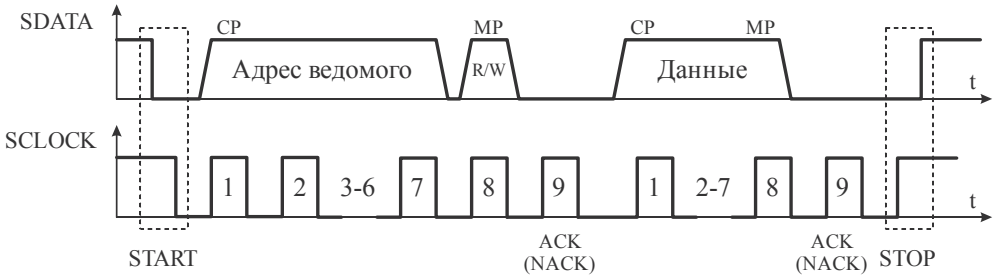


Рис. 26. Последовательность передачи по интерфейсу I^2C

После выбора ведомого происходит передача информационного байта. Во время передачи данных сигналы ACK и NACK всегда генерируются приемником. В таком случае девятый тактовый импульс, необходимый для этих сигналов, всегда генерируется ведущим. Передатчик должен освободить линию SDATA во время девятого тактового импульса.

Последовательность передачи данных заканчивается условием STOP, генерируемым ведущим устройством. STOP определяется переходом линии SDATA с низкого на высокий уровень, когда линия SCLOCK находится в высоком состоянии.

В расширенной версии устройство может иметь 10-битный адрес, что позволяет адресовать до 1008 периферийных устройств. При этом адрес ведомого передается двумя байтами. Использование 7-битного адреса допускает взаимодействие со 112 микросхемами, т. к. 16 адресов зарезервированы.

Основные преимущества шины I^2C:

- использование двух проводников для подключения нескольких устройств;
- возможна одновременная работа нескольких ведущих устройств, подключенных к одной шине I^2C;
- стандарт предусматривает «горячее» подключение и отключение устройств в процессе работы системы;
- наличие достаточно подробного и жесткого стандарта на шину I^2C обуславливает выпуск меньшего количества несовместимых I^2C-микросхем у различных производителей;
- встроенный фильтр подавляет всплески, обеспечивая целостность данных.

Основные недостатки шины I^2C:

- максимальное допустимое количество микросхем, подсоединенных к одной шине, ограничивается максимальной емкостью шины в 400 пФ;
- сложность реализации режима с несколькими ведущими;
- трудность локализации неисправности, если одно из подключенных устройств ошибочно устанавливает на шине состояние низкого уровня.

Интерфейсы SPI и I^2C микроконтроллера ADuC812

Микроконтроллер ADuC812 содержит аппаратный интерфейс SPI, поддерживающий полнодуплексный режим. Вследствие ограниченности количества контактов микроконтроллера выводы интерфейса SPI мультиплексируется с шиной I^2C и некоторыми портами ввода-вывода. Регистры, предназначенные для работы с интерфейсом SPI:

- SPIDAT — регистр данных; записанные в этот регистр данные автоматически передаются через соответствующие выводы, а полученная информация может быть считана программой;
- SPICON — регистр управления:

SPICON.7	I2CCON.6	SPICON.5	SPICON.4	SPICON.3	SPICON.2	SPICON.1	SPICON.0
ISPI	WCOL	SPE	SPIM	CPOL	CPHA	SPR1	SPR0

Поля регистра SPICON:

- ISPI — прерывание SPI интерфейса, устанавливается аппаратно после окончания каждого обмена данными. Очищается программно либо аппаратно при чтении регистра SPIDAT;
- WCOL — бит ошибки, возникающей при попытке считывания регистра SPIDAT во время процесса передачи;
- SPE — включение интерфейса SPI, устанавливается для включения интерфейса SPI, сбрасывается для включения интерфейса I^2C;
- SPIM — выбор режима работы: устанавливается — режим ведущего, сбрасывается — режим ведомого;
- CPOL — указывает полярность синхросигналов: если 0 — начинается с нулевого уровня (режимы 0 и 1), 1 — с уровня логической 1 (режимы 2 и 3);
- CPHA — фаза синхросигналов, определяющая момент выборки информации считывающим устройством: 0 — передний фронт, 1 — задний фронт;

- SPR1, SPR0 — в режиме ведущего, эти биты определяют частоту работы интерфейса (табл. 18).

Таблица 18

Частоты синхронизации SPI интерфейса

SPR1	SPR0	Частота
0	0	$f_{osc}/4$
0	1	$f_{osc}/8$
1	0	$f_{osc}/32$
1	1	$f_{osc}/64$

Данная микросхема поддерживает 2-проводной I^2C-совместимый последовательный интерфейс, который можно сконфигурировать в режиме Программно ведущего (Software Master) или как Аппаратно ведомого (Hardware Slave). Ниже приведены регистры, позволяющие взаимодействовать с интерфейсом I^2C:

- I2CADD — адресный регистр интерфейса I^2C, содержит адрес периферийного устройства;
- I2CDAT — регистр данных I^2C, при записи информации в данный регистр инициируется ее передача по интерфейсу I^2C, полученные данные считываются программно;
- I2CCON — регистр управления I^2C:

I2CCON.7	I2CCON.6	I2CCON.5	I2CCON.4	I2CCON.3	I2CCON.2	I2CCON.1	I2CCON.0
MDO	MDE	MCO	MDI	I2CM	I2CRS	I2CTX	I2CI

Поля регистра I2CCON:

- MDO — выход последовательных данных в режиме Программно Ведущий. Значение этого бита передается на внешний вывод SDATA, если установлен бит разрешения передачи данных (MDE);
- MDE — разрешение передачи данных в режиме Программно Ведущий. Устанавливается пользователем для включения вывода SDATA в качестве выхода. Сбрасывается для использования SDATA в качестве входа;
- MCO — синхронизация в режиме Программно Ведущий. Значение этого бита передается на внешний вывод SCLOCK, используется для вывода данных в режиме Программно Ведущего (Software Master);

- MDI — вход последовательных данных в режиме Программно Ведущий;
- I2CM — выбор режима Программно Ведущий/Аппаратно Ведомый;
- I2CRS — сброс последовательного порта в режиме Ведомый: 1 — перезапуск, 0 — нормальное функционирование;
- I2CTX — состояние направления передачи в режиме Ведомый: 1 — передача данных, 0 — прием;
- I2CI — прерывание I^2C-интерфейса.

На лабораторном стенде SDK1.1 к шине I^2C подключены только два встроенных периферийных устройства: часы реального времени и внешняя EEPROM данных. Адреса этих устройств приведены в табл. 19.

Таблица 19

Адреса периферийных устройств на шине I^2C

Адрес I^2C								Периферийное устройство
1	0	1	0	0	0	1	r/w	EEPROM данных
1	0	1	0	0	0	0	r/w	Часы реального времени

Примечание. r/w — выбор режимов чтения (1) или записи (0).

3.1.5. Таймер-счетчики

ADuC812 содержит три 16-разрядных таймер-счетчика: таймер 0, таймер 1 и таймер 2. Каждый таймер-счетчик состоит из двух 8-разрядных таймерных регистров THx и TLx (x = 0, 1 и 2). Все три можно сконфигурировать либо как таймеры, либо как счетчики событий.

Таймер-счетчики 0 и 1 для конфигурации используют регистры TMOD и TCON, их работа описана в п. 1.4 (с. 19). Таймер-счетчик 2 работает отлично от таймер-счетчиков 0 и 1. Управляющим регистром является T2CON, его структура приведена ниже:

T2CON.7	T2CON.6	T2CON.5	T2CON.4	T2CON.3	T2CON.2	T2CON.1	T2CON.0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	CNT2	CAP2

Поля регистра T2CON:

- TF2 — флаг переполнения таймер-счетчика 2. Устанавливается аппаратно при переполнении таймер-счетчика. Сбрасывается

программно. Не устанавливается, если RCLK или TCLK имеют значение логической 1;

- EXF2 — устанавливается аппаратно в режимах захвата или автоперезагрузки, если EXEN2 = 1 и произошел отрицательный перепад на T2EX. Очищается программно;
- RCLK — устанавливается программно, когда необходимо использовать переполнение таймер-счетчика 2 в качестве синхросигнала при приеме информации по последовательному порту в режиме 1 и 3. Сбрасывается программно;
- TCLK — аналогично предыдущему, но используется для передачи информации;
- EXEN2 — устанавливается и сбрасывается программно, предназначен для реагирования на отрицательный перепад на T2EX;
- TR2 — бит управления таймера. Устанавливается либо сбрасывается программой для запуска либо остановки;
- CNT2 — выбор функции: 0 — таймер, 1 — счетчик;
- CAP2 — бит выбора режима: захвата или автоперезагрузки.

Режимы работы таймер-счетчика 2 показаны в табл. 20.

Таблица 20

Режимы работы таймер-счетчика 2

RCLK или TCLK	CAP2	TR2	Режим
0	0	1	16-разрядный автоперезагружаемый таймер-счетчик
0	1	1	16-разрядный таймер-счетчик в режиме захвата
1	X	1	Синхросигнал для последовательного порта
X	X	0	Выключен

В режиме работы 16-разрядного автоперезагружаемого таймер-счетчика, при переполнении таймер-счетчика 2, устанавливается бит TF2, в регистры TH2 и TL2 загружаются соответственно значения из регистров RCAP2H и RCAP2L и выставляется запрос на прерывание. Если EXEN2 = 1, то при изменении значения на входе T2EX из 1 в 0 устанавливается бит EXF2 и происходит загрузка регистров TH2 и TL2 из регистров RCAP2H и RCAP2L.

В режиме захвата, при переполнении таймер-счетчика 2, устанавливается бит TF2 и генерируется запрос на прерывание. Если EXEN2 = 1,

то при изменении значения на входе T2EX из 1 в 0 устанавливается бит EXF2 и происходит запись значений регистров TH2 и TL2 в регистры RCAP2H и RCAP2L соответственно.

3.1.6. Внутренние мониторы

Для минимизации повреждения кода или данных вследствие возникновения катастрофических программных или внешних сбоев системы, ADuC812 включает в себя две мониторные функции: охранный таймер (WDT) и монитор источника питания (PSM). Обе мониторные функции конфигурируются через регистры SFR.

Назначение охранного таймера (WDT) — сгенерировать сигнал сброса устройства, если ADuC812 выполняет ошибочные действия, например, по причине сбоя программы или из-за электрических либо электромагнитных помех. Действие WDT можно запретить очисткой бита Разрешения WDE в регистре Управления охранным таймером (WDCON) SFR.

При разрешенном охранном таймере он будет генерировать системный сброс, если программа пользователя не обновляет его содержимое в интервале предустановленного времени. Интервал можно менять в диапазоне 16–2048 мс с помощью программирования специального регистра.

Монитор источника питания (PSM) генерирует прерывание, когда значение аналогового или цифрового напряжения питания падает ниже одной из пяти устанавливаемой пользователем пороговой величины (2,6–4,6 В). Бит прерывания не будет очищаться в течение 256 мс до тех пор, пока напряжение источника не станет выше порогового значения.

Эта функция гарантирует, что пользователь успеет спасти рабочие регистры во избежание возможной потери данных из-за низкого питания. Выполнение программного кода не продолжится до тех пор, пока не установится «безопасный» уровень питания. Монитор питания также защищен от импульсных помех в цепи прерывания.

3.2. Периферийное оборудование стенда SDK-1.1

На стенде SDK-1.1 ввод-вывод данных помимо портов микроконтроллера осуществляется также с помощью микросхемы ПЛИС, которая имеет 7 регистров, отображаемых во внешнее пространство памяти данных, (табл. 21).

Таблица 21

Регистры ПЛИС

Адрес	Регистр	Доступ	Назначение
080000h	KB	R/W	Регистр клавиатуры
080001h	DATA_IND	R/W	Регистр шины данных ЖКИ
080002h	EXT_LO	R/W	Регистр данных параллельного порта (разряды 0–7)
080003h	EXT_HI	R/W	Регистр данных параллельного порта (разряды 8–15)
080004h	ENA	W	Регистр управления портами ввода-вывода, звуком и сигналом INT0
080006h	C_IND	W	Регистр управления ЖКИ
080007h	SV	W	Регистр управления светодиодами

3.2.1. Клавиатура

Клавиатура AK1604A-WWB организована в виде матрицы 4x4. Доступ к колонкам и строкам организован как чтение и запись регистра KB. Структура регистра KB:

Строка, ROW (Чтение)				Колонка, COL (Запись)			
D7	D6	D5	D4	D3	D2	D1	D0

Поля регистра KB:

- COL (биты 0–3) — предназначено для сканирования клавиатуры. Сканирование производится посредством записи 0 в один из разрядов;
- ROW (биты 4–7) — предназначено для считывания данных с клавиатурной матрицы (строки). Если ни одна из кнопок на клавиатуре не нажата, то все биты этого поля содержат 1. Если кнопка нажата и на ее колонку подан 0, то в поле ROW также появится 0.

3.2.2. Жидкокристаллический индикатор (ЖКИ)

Жидкокристаллический индикатор (ЖКИ) WH1602B-YGK-CP способен отображать текстовую информацию в 2 строках по 16 символов. Имеет встроенный набор из 256 символов (ASCII + кириллица) и генератор символов с энергозависимой памятью на 8 пользовательских символов. Управление ЖКИ производится при помощи двух регистров DATA_IND и C_IND.

Регистр DATA_IND позволяет устанавливать данные на шине данных ЖКИ и считывать их оттуда. Для организации взаимодействия с ЖКИ необходимо использовать регистр C_IND, структура которого показана ниже:

C_IND.7	C_IND.6	C_IND.5	C_IND.4	C_IND.3	C_IND.2	C_IND.1	C_IND.0
					RS	RW	E

Поле регистра C_IND:

- E — бит управления входом «E» ЖКИ. Наличие положительного импульса на входе «E» позволяет зафиксировать данные на шине ЖКИ (данные, сигналы RS и RW должны быть установлены);
- RW — бит переключения шины данных ЖКИ: 0 — запись, 1 — чтение;
- RS — бит переключения типа информации: 1 — данные, 0 — команды.

Основные команды, воспринимаемые ЖКИ, приведены в табл. 22.

Таблица 22

Основные команды ЖКИ

Команда	Время выполнения	RS	RW	Регистр DATA_IND							
				D7	D6	D5	D4	D3	D2	D1	D0
Очистка дисплея	1,53 мс	0	0	0	0	0	0	0	0	0	1
Возврат в начало строки	1,53 мс	0	0	0	0	0	0	0	0	1	X
Установка позиции курсора	39 мкс	0	0	1	a6	a5	a4	a3	a2	a1	a0
Чтение флага занятости (BF) и текущей позиции курсора	0 мкс	0	1	BF	a6	a5	a4	a3	a2	a1	a0

Примечание. a6... a0 — код позиции курсора, отсчитанный слева направо: 00h...0Fh для верхней строки и 40h...4Fh для нижней строки дисплея.

3.2.3. Параллельный порт

Регистры EXT_HI и EXT_LO предназначены для обмена данными через 16-разрадный параллельный порт ввода-вывода. Старшими битами порта (8–15) управляют при помощи регистра EXT_HI, а младшими (0–7) — EXT_LO. Режим вывода данных активируется установкой соответствующих бит в регистре ENA: EN_LO и EN_HI.

3.2.4. Звуковой излучатель

Регистр ENA предназначен для управления портами ввода-вывода, звуком и сигналом INT0, его структура следующая:

ENA.7	ENA.6	ENA.5	ENA.4	ENA.3	ENA.2	ENA.1	ENA.0
		INT0	SND2	SND1	SND0	EN_HI	EN_LO

Поля регистра ENA:

- EN_LO — используется для управления младшими разрядами (0–7) 16-разрядного порта ввода-вывода. Если EN_LO = 0, то порт переводится в Z-состояние и появляется возможность считывания данных из EXT_LO. Если EN_LO = 1, то порт переключается в режим вывода и данные, содержащиеся в регистре EXT_LO, попадают на выход порта ввода-вывода;
- EN_HI — аналогичен EN_LO, но управляет старшими битами (8–15) порта;
- SND0 — SND2 — выход звукового ЦАП, задает уровень напряжения на динамике. Позволяет формировать звуковые сигналы различной тональности и громкости;
- INT0 — значение этого бита передается на вход INT0 микроконтроллера ADuC812. Бит можно использовать для формирования внешнего прерывания для микроконтроллера.

3.2.5. Часы-календарь реального времени

Лабораторный стенд SDK1.1 содержит часы-календарь типа PCF8583, подключенный к микроконтроллеру ADuC812 через шину I^2C. Это устройство с внутренней встроенной памятью объемом

256 байт, работающее от кварцевого резонатора с частотой 32,768 кГц. Микросхема может работать в режиме часов или счетчика событий.

В режиме часов первые 8 байт оперативной памяти используются для текущего времени и календаря (табл. 23). Следующие 8 байт могут быть запрограммированы на использование в качестве регистров сигнализации (будильника), или же к ним можно обращаться как к свободным адресам памяти. Остальные 240 байт относятся к оперативной памяти. Хранение времени осуществляется в двоично-десятичном коде (ДДК).

Таблица 23

Назначение ячеек памяти PCF8583

Назначение ячеек	Адрес
Управление/состояние	00h
Одна сотая доля секунды	01h
Секунды	02h
Минуты	03h
Часы	04h
Год/дата	05h
Дни недели/месяцы	06h
Таймер	07h
Управление сигналом	08h
Регистры сигнала или ячейки памяти	09h...0Fh
ОЗУ (240x8)	10h...FFh

Спецификация регистров, форматы данных и режимы работы микросхемы приведены в руководстве пользователя [6].

3.3. Интегрированная среда Keil µVision IDE

µVision IDE — отладочная среда компании ARM Group для микроконтроллеров различных семейств, в том числе MCS-51, ARM Cortex®-M и других. Сочетает в себе средства управления проектами, мощный текстовый редактор, многофункциональный отладчик, встроенную справочную систему, а также транслятор с языка C51, макроассемблер A51, редактор связей L51. Ниже показан порядок работы с µVision.

1) В своей личной студенческой папке на компьютере необходимо создать директорию Мой проект Keil. Для каждой лабораторной работы рекомендуется создавать отдельную подпапку, например, указывая номер лабораторной работы: LR1, LR2 ...В каждой из них будут размещаться многочисленные файлы, создаваемые вами и интегрированной средой в процессе разработки программ: исходные тексты, результаты трансляции программ для загрузки в учебный стенд и различные вспомогательные файлы.

2) Для запуска программы µVision служит файл

c:\Program Files\Keil Software\uv4\uv4.exe

Программу можно запустить, используя пункт меню Пуск/Программы/Keil µVision5 или одноименный ярлык на рабочем столе.

3) Разработка программы в среде µVision начинается с создания нового проекта. Для этого в меню необходимо выбрать пункт Project/New µVision Project… В появившемся окне обзора выбрать свою папку Мой проект Keil и соответствующую лабораторной работе подпапку LR1, затем ввести имя проекта без расширения. Удобно в имени проекта использовать свое имя и (или) фамилию, а также номер лабораторной работы, что в дальнейшем облегчает поиск нужных файлов. Имя файла проекта требуется писать латинскими буквами без пробелов, т. к. это название автоматически подставляется в название компилируемого файла. Например, выберем имя проекта PopovLab1.

4) После создания нового проекта требуется определить его параметры и свойства. Для этого, сразу после создания автоматически появится окно выбора микроконтроллера, которое также можно вызвать при помощи меню Project/Select device for Target …, здесь укажите тип микроконтроллера: Analog Devices — ADuC812. При возникновении вопроса: «Copy ‘START_AD.A51’ to Project Folder and Add File to Project?» — следует ответить Нет. Затем, используя пункт меню Project/Options for Target ‘Target 1’, необходимо открыть соответствующее окно, в котором на вкладке Output поставить галочку в поле Create HEX File, что позволит при компиляции формировать файл загрузочного модуля — hex-файл. Именно этот hex-файл в дальнейшем будет загружаться в лабораторный стенд. Здесь же можно изменить имя компилируемого hex-файла, указав требуемое в поле Name of Executable.

5) Чтобы открыть или создать новый текстовый файл для написания исходного текста программы на языке ассемблера, необходимо выбрать в меню File/New… (Ctrl-N). Появится окно текстового редактора.

В меню надо выбрать File/Save… (Ctr-S*)*, а в появившемся окне обзора выбрать свою папку Мой проект Keil и подпапку с текущей лабораторной работой, куда и сохранить созданный файл с расширением (**asm**) (расширение необходимо указывать обязательно!). В нашем случае удобно в качестве имени файла использовать имя проекта, т. к. в проекте будет присутствовать только один asm-файл. Например, «PopovLab1.asm».

6) Для включения созданного текстового файла в проект необходимо, используя меню Project/Manage/Project Items…, вызвать одноименное окно, где на вкладке Project Items нажать кнопку Add Files…, в появившемся окне обзора выбрать файл, созданный в предыдущем пункте (т. е. PopovLab1.asm), и нажать кнопку Add. Для добавления других файлов в проект необходимо выбирать их по очереди и нажимать кнопку Add. Завершение процедуры добавления осуществляется нажатием кнопки Close.

В проект можно добавлять не только файлы с исходными текстами программы на языке ассемблера, но и файлы с программами, написанными на языке С51 (***.c**), а также объектные файлы (***.obj**) и объектные библиотеки (***.lib**). В зависимости от расширения имени файла интегрированная среда будет вызывать соответствующую обрабатывающую программу (макроассемблер, компилятор С51, редактор связей и т. д.). В дальнейшем при выполнении некоторых лабораторных работ потребуется подключение некоторых объектных библиотек, о которых будет сказано дополнительно.

7) В текстовом окне интегрированной среды следует набрать текст программы.

8) После сохранения программы при помощи меню File/Save… (Ctrl-S) можно провести компиляцию (т. е. трансляцию в машинные коды), выбрав команду в меню Project/Build target (F7). В случае обнаружения ошибок при компиляции соответствующая информация появится в окне сообщений, тогда требуется проанализировать возникшие ошибки и внести изменения в исходный текст программы, а затем повторить компиляцию.

9) В результате успешной компиляции в подпапке LR1/Objects (соответствующей лабораторной работе) будет создан загрузочный файл с расширением .**hex**, содержащий машинные коды для загрузки в лабораторный стенд. Имя hex-файла совпадает с именем, указанном в поле Name of Executable свойств проекта, соответственно файл имеет расширение (hex), например, PopovLab1.hex.

Загрузка hex-файла в лабораторный стенд осуществляется с помощью другой программы — загрузочного монитора Т2.

3.4. Загрузочный монитор Т2

Загрузочный монитор Т2 работает на компьютере под управлением ОС Windows 9.x/NT/2k/XP/7. Основные функции монитора Т2:

- добавление стартового адреса программы в передаваемый hex-файл;
- передача загрузочных модулей в лабораторный стенд;
- получение информации из лабораторного стенда и взаимодействие с резидентным загрузчиком HEX202, работающем в лабораторном стенде;
- обеспечение элементарных операций с последовательным каналом — прием, передача, настройка скорости и т. д.

Ниже представлен порядок работы с загрузочным монитором Т2.

Запуск загрузочного монитора осуществляется вызовом программы **«t2.exe»** из папки **«c:\t2»**. В результате появится окно монитора с промптом (#). Для ввода команд используется командная строка, начинающаяся с промпта. Монитор Т2 предназначен для работы с различными лабораторными стендами, и поэтому имеет большой список команд. Для работы в рамках лабораторного практикума со стендом SDK1.1 потребуется ограниченное число команд. Полный список команд содержится в руководстве пользователя монитора.

Запись программ в стенд SDK-1.1 может осуществляться только тогда, когда он находится в режиме ожидания загрузки. Перевод стенда в состояние загрузки осуществляется его перезапуском, т. е. нажатием кнопки Reset на передней панели. В зависимости от версии стенда на дисплее может отображаться или отсутствовать надпись: «HEX202 Ожидание...».

Для определения номера последовательного порта, через который будет осуществляться обмен информацией со стендом, в окне монитора Т2 необходимо выполнить команду

opencom1 или opencom2

Первая команда используется в случае, когда стенд подключен к порту COM1, а вторая — к порту COM2. Кроме того, эта команда

открывает соответствующий порт компьютера для обмена информацией со стендом.

Ниже представлены команды, предназначенные для проведения некоторых операций с hex-файлом. Стоит обратить внимание на то, что полученный hex-файл должен находиться в папке расположения монитора T2, поэтому требуется предварительно скопировать hex-файл в соответствующую папку. В командах допускается использовать полный путь к hex-файлу, при этом путь и имя файла не должны содержать пробелы.

После определения порта вводится команда, добавляющая стартовый адрес программы в конец hex-файла,

0x02000 0x0 addhexstart <имя_файла>.hex

Например, если создан hex-файл с именем **PopovLab1.hex**, то строка добавления стартового адреса будет выглядеть следующим образом:

0x02000 0x0 addhexstart PopovLab1.hex

Резидентный загрузчик HEX202 использует этот адрес для автоматического запуска программы, загруженной в лабораторный стенд. Параметры команды addhexstart: первая цифра (**0x02000**) — стартовый адрес программы; вторая цифра (**0x0**) должна присутствовать, но при работе со стендом SDK-1.1 не используется и может иметь любое значение; последний параметр — имя hex-файла.

В случае, когда необходимо повторно загрузить hex-файл, к которому уже была добавлена строка запуска, команду addhexstart выполнять не требуется. Если была произведена компиляция исходного текста и к вновь созданному hex-файлу строка запуска еще не добавлялась, то это действие выполнять нужно обязательно, в противном случае после загрузки программы она автоматически не запустится. Необходимо следить за совпадением стартового адреса программы (число, которое записано после ключевого слова «org», расположенного в начале основной программы) и указываемым адресом в команде addhexstart.

Загрузка программы в лабораторный стенд производится командой

loadhex <имя_файла>.hex

В нашем случае

loadhex PopovLab1.hex

По окончании загрузки в окне монитора T2 появится сообщение о результате программирования памяти микроконтроллера. Программа, загруженная в лабораторный стенд, запустится с указанного адре-

са автоматически резидентным монитором HEX202, если передача информации была выполнена успешно.

В мониторе T2, используя клавиши ↑ и ↓, можно выбирать команды из введенных ранее в текущем сеансе работы.

Для выхода из монитора T2 служит команда bye. В случае зависания монитора T2 используют аварийный выход клавишами CTRL-C или CTRL-Break либо закрывают окно.

Часто повторяющийся набор команд можно оформить в виде скриптового файла, т. е. файла, содержащего команды монитора T2. Скрипт-файл может иметь любое расширение или быть без расширения. Пример скрипт-файла с именем **script1** без расширения:

```
0x2000 0x0 addhexstart kb.hex
loadhex kb.hex
```

Скрипт-файл должен находиться в одной директории с **t2.exe**. Выполнение команд, содержащихся в скрипт-файле, осуществляет специальная команда монитора T2, которой необходимо сообщить имя скрипт-файла,

```
lfile script1
```

В данном примере команда lfile загружает и выполняет скрипт-файл с именем script1. Такой способ ускоряет процесс загрузки и запуска программы после компиляции.

При выполнении команды opencom1 открывается последовательный порт со скоростью передачи 9600 бод. При необходимости использовать другие стандартные скорости применяется команда

```
19200 openchannel com1
```

Первая цифра (19200) задает скорость порта в бодах (бит в секунду). В данном примере открывается порт COM1 для работы на скорости 19 200 бод.

Закрытие канала связи осуществляется командой

```
closechannel
```

Работа монитора T2 в режиме эмуляции терминала

Приведенный выше перечень команд монитора T2 достаточен для выполнения основных функций взаимодействия с резидентным загрузчиком HEX202. Однако монитор T2 предоставляет и другие полезные возможности. Команда N term включает эмулятор терминала

```
N term
```

т. е. переводит T2 в так называемый режим терминала: все байты, приходящие по последовательному каналу, отображаются на экране монитора в символьном виде (**N = 0**) или шестнадцатеричном коде (**N = 1**), а все байты, набираемые на клавиатуре, не воспроизводятся на экране, а передаются по последовательному каналу на лабораторный стенд. Выход из этого режима производится по нажатию клавиши ESC на клавиатуре компьютера. Для записи получаемой информации в файл предусмотрена команда создания файла с заданным именем для вывода, а также команды включения и выключения копирования консольного вывода в созданный файл:

echo <имя_файла_для_вывода> — создание файла;

+echo — включение режима записи в файл;

–echo — выключение режима записи в файл.

Сохранение всей информации в файл осуществляется после выхода из монитора по команде bye. Стоит отметить, что в файл копируются только те байты, которые поступают на компьютер через COM-порт.

С использованием данного режима монитора T2 возможно выводить из лабораторного стенда и записывать в файл на компьютере большие массивы информации, например, создавать образы памяти или выводить отладочную информацию.

4. Программное управление встроенной периферией лабораторного стенда SDK-1.1

4.1. Запись данных в регистры ПЛИС

Лабораторный стенд SDK-1.1 содержит ряд периферийных устройств, взаимодействие с которыми осуществляется через регистры ПЛИС (табл. 21, с. 87). Это — клавиатура, ЖКИ-дисплей, линейка из 8 светодиодов, внешний параллельный порт и звуковой излучатель. Адреса регистров ПЛИС расположены на странице 8 внешней памяти данных. Ниже приведена подпрограмма, осуществляющая запись в регистр ПЛИС, адрес которого указывается в регистре R0, а передаваемое значение — в аккумуляторе [9].

```
;************************ PUTBYTE ******************************
; Запись байта в регистр ПЛИС (0h…7h) стенда SDK-1.1
; адрес регистра ПЛИС – в (R0), байт данных – в (A)
;***************************************************************
putbyte: push dpp      ; сохранение номера текущей страницы в стек
         mov dpp, #08d ; выбор 8-й страницы внешней памяти данных
         movx @r0, A   ; запись в регистр ПЛИС
         pop dpp       ; восстановление номера страницы
         ret           ; возврат из подпрограммы
```

4.2. Управление светодиодами

Для работы с линейкой светодиодов служит регистр SV, каждый бит которого управляет свечением одного светодиода. Единичное значение бита соответствует свечению светодиода. Программирование светодиодных излучателей сводится к выводу байта данных в регистр SV (табл. 21, с. 87). Для иллюстрации ниже приведена программа, копирующая байт данных из аккумулятора в регистр SV с использованием подпрограммы PUTBYTE [9].

```
;***************************************************************
; Вывод байта на линейку светодиодов стенда SDK1.1
; с использованием подпрограммы putbyte
;***************************************************************
org 02000h              ; начальный адрес расположения программы
                        ; во внутренней памяти программ
dpp data 84h            ; адрес регистра DPP
mybyte equ 11000011b    ; байт данных для вывода
sv xdata 07h            ; адрес регистра светодиодов
         mov r0, #sv    ; адрес регистра светодиодов в R0
         mov a, #mybyte ; загрузка в аккумулятор значения mybyte
         call putbyte   ; вывод на светодиодную линейку
         sjmp $         ; бесконечная петля
end
```

4.3. Вывод на ЖКИ

Вывод символа или отправка команды на ЖКИ выполняется подпрограммой PutChar [9], приведенной ниже. Перед вызовом подпрограммы передаваемый код символа или команды должен быть записан в аккумулятор, а тип кода — в бите RS (0 — команды, 1 — данные).

```
;*************************** PUTCHAR ****************************
; Вывод одного байта на ЖКИ стенда SDK-1.1
; Байт для вывода – в аккумуляторе.
; Значение бита RS – 0-команды, 1-данные
;****************************************************************
; определение адресов используемых регистров
dpp data 84h       ; адрес регистра DPP
data_ind xdata 01h ; регистр данных ЖКИ
c_ind xdata 06h    ; регистр управления ЖКИ
; образы битовых полей регистра C_IND
E BIT acc. 0
RW BIT acc. 1
RS BIT F0
; сохранение используемых регистров в стеке
putchar: push psw   ; сохранение слова состояния программы
anl psw, #11100111b ; выбрали банк регистра 0
push 0h             ; сохранение r0
push dpp            ; сохранение номера страницы
mov dpp, #08d       ; выбор 8-й страницы внешней памяти данных
; вывод данных в регистр данных ЖКИ
mov r0, #data_ind   ; адрес регистра данных
movx @r0, A         ; вывод данных из аккумулятора в DATA_IND
; запись кодовых полей в регистр управления ЖКИ
clr RW              ; RW=0, будет осуществляться передача
                    ; информации в ЖКИ
mov c, RS           ; перенос RS
mov acc.2, c        ; во 2-й бит аккумулятора
; формирование строба 0-1-0 на входе «E» ЖКИ для фиксации данных
mov r0, #c_ind      ; в r0 записываем адрес регистра команд
clr E               ; E=0
```

```
movx @r0, A          ; запись в C_IND
setb E               ; E=1
movx @r0, A          ; запись в C_IND
clr E                ; E=0
movx @r0, A          ; запись в C_IND
; восстановление использованных регистров
pop dpp              ; восстановление номера страницы
pop 0h               ; восстановление регистра r0
pop psw              ; восстановление слова состояния программы
ret                  ; возврат из подпрограммы
```

Пример отправки команды на ЖКИ очистки экрана:

```
clr RS; RS=0, включение режима отправки команды
mov A, #01h; команда очистки экрана
call putchar; вызов подпрограммы вывода байта в ЖКИ
```

При использовании команд управления ЖКИ не забывайте о времени их выполнения, например, приведенная выше команда очистки экрана выполняется в течение примерно 1,5 мс. Это означает, что команды или данные, отправленные в это время ЖКИ, не воспринимаются и будут проигнорированы.

4.4. Клавиатура

Для ввода информации в стенд может использоваться встроенная клавиатура. Опрос клавиатуры и определение кода нажатой клавиши осуществляется с помощью подпрограммы GetKey [9], в результате работы которой в аккумуляторе содержится код нажатой клавиши или значение 0FFh, если ни одна из кнопок не была нажата.

```
;*************************** GETKEY ****************************
; Опрос клавиатуры стенда SDK-1.1
; Код нажатой клавиши (00h…0Fh) в аккумуляторе
; Если ни одна клавиша не нажата, то код 0FFh
;***************************************************************
```

```
; определение адресов используемых регистров
dpp data 84h; адрес регистра DPP
kb xdata 00h; адрес регистра клавиатуры
; сохранение используемых в подпрограмме GETKEY регистров в стеке
getkey: push psw      ; сохранение слова состояния программы
        push 0h       ; сохранение r0
        push 1h       ; сохранение r1
        push b        ; сохранение B
        push dpp      ; сохранение номера страницы
        mov psw, 0h   ; выбор нулевого банка
        mov dpp, #08d ; выбор 8-й страницы внешней памяти данных
; сканирование клавиатуры
        mov r0, #kb   ; адрес регистра KB в r0 для последующего
                      ; взаимодействия с клавиатурой
        mov r1, #0    ; счетчик колонок = 0
        mov b,#01111111b ; сканирующий (0)
KB_lp:  mov a, b
        rl a          ; установка скан. (0) в проверяемый столбец
        mov b, a      ; сохранение скан. (0)
        movx @r0, a   ; вывод в KB скан. (0)
        movx a, @r0   ; чтение KB ROW
        cpl a         ; инверсия ROW
        anl a, #0f0h  ; ROW в старшей тетраде A, очистка поля COL
        jnz kb_cod    ; если нажата клавиша – переход на
                      ; формирование кода
        inc r1        ; выбор следующей колонки
        cjne r1, #4, KB_lp ; продолжить сканирован, если номер
                           ; колонки не 4
        mov a, #0FFh  ; нет нажатых клавиш
        sjmp kb_end
; формирование кода нажатой клавиши (00h…0Fh)
kb_cod: mov b, #0; очистка per. (B)
        mov c, acc.7
        orl c, acc.5
        mov b.2, c
        mov c, acc.7
        orl c, acc.6
        mov b.3, c
```

```
        mov a, b
        add a, r1
; восстановление использованных регистров
kb_end: pop dpp; восстановление номера страницы
        pop b    ; восстановление B
        pop 1h   ; восстановление r1
        pop 0h   ; восстановление r0
        pop psw  ; восстановление слова состояния программы
        ret      ; возврат из подпрограммы
```

При разработке программы, обрабатывающей нажатие клавиш стенда SDK 1.1, можно определить функциональное назначение используемых клавиш и их код. Например:

```
buttom1 equ 00h ; код клавиши 1 — продолжение измерений
```

В таком случае в тексте программы вместо кода клавиши 1, равного 0, (т. е. вместо цифры 0) можно использовать идентификатор buttom1.

Кроме того, необходимо учитывать, что подпрограмма getkey не устраняет дребезг контактов. Для решения этой проблемы желательно создать подпрограмму, основанную на задержке после обработки нажатой клавиши с использованием пустых циклов или прерываний от таймеров, последний вариант предпочтительнее.

4.5. Звуковой излучатель

Лабораторный стенд SDK 1.1 содержит встроенный звуковой пьезокерамический излучатель, управляемый тремя битами регистра ENA ПЛИС (см. 3.2.4, с. 89): ENA.2, ENA.3 и ENA.4. Одним из вариантов формирования звука является побитовое использование управляющих битов, для этого на любой из битов необходимо подать последовательность из логических уровней 0-1-0-1-... с заданной частотой, тогда на выходе микроконтроллера будет формироваться соответствующий меандр. Изменение громкости звука осуществляется выбором используемого бита: от тихого (ENA.2) до чуть более громкого (ENA.4).

При выводе данных в регистр ENA необходимо иметь в виду, что бит ENA.5 соединен со входом INT0 микроконтроллера и на нем должна поддерживаться логическая единица.

Приведенная ниже программа иллюстрирует формирование непрерывного звука заданного тона [9]. Задержка примерно в 300 мкс реализована методом программного цикла и позволяет получить частоту колебаний около 1.6 кГц.

```
;*****************************************************************
; Формирование непрерывного звука заданного тона
; с использованием подпрограммы PUTBYTE
;*****************************************************************
org 02000h
dpp data 84h          ; адрес регистра DPP
ena xdata 04h         ; адрес регистра ENA
buz bit acc.3         ; бит для вывода звука
        mov r0, #ena  ; указание регистра ENA для подпрограммы
                      ;  putbyte
        mov a, #20h   ; на вход INT0 записывается 1
loop:   setb buz      ; установка бита для вывода звука
        call putbyte  ; отправка в ENA 1
        call delay    ; формирование задержки
        clr buz       ; сброс бита для вывода звука
        call putbyte  ; отправка в ENA 0
        call delay    ; формирование задержки
        sjmp loop     ; переход в начало для формирования
                      ; бесконечного меандра
        delay:  mov r1, #096h; определение задержки около 300 мкс
        djnz r1, $    ; пустой цикл
        ret           ; возврат
end
```

4.6. Универсальный асинхронный приемопередатчик (UART)

Наличие в микроконтроллере ADuC812 встроенного программируемого универсального асинхронного приемопередатчика (УАПП) позволяет организовать эффективное взаимодействие лабораторного

стенда SDK1.1 с терминалом T2, функционирующим на компьютере. В режиме эмуляции терминала (см. п. 3.4, с. 95) программа T2 позволяет все набираемые на клавиатуре компьютера коды отправлять на вход УАПП стенда, а все байты, передаваемые с УАПП, отображать на терминале в текстовом или шестнадцатеричном коде. Таким образом, из лабораторного стенда можно передавать достаточно большие объемы информации для отображения на экране или для записи в файл на компьютере и, наоборот, из компьютера передавать массивы данных для размещения в памяти стенда. Это позволяет исследовать содержимое областей памяти микроконтроллера и различных периферийных устройств, входящих в состав лабораторного стенда.

Подпрограмма UART_INI иллюстрирует технологию начальной инициализации универсального асинхронного приемопередатчика (УАПП) [9]. В данном примере производится инициализация УАПП для работы со скоростью обмена 9.6 кбит/с с запретом прерываний для работы в режиме опроса флагов. Эта подпрограмма должна быть выполнена до начала любых взаимодействий с использованием УАПП.

```
;************************** UART_INI ***************************
; Пример программы инициализации UART
;***************************************************************
; определение начальных значений таймер-счетчика для стандартных
; скоростей обмена UART
S9600 equ 0fdh                  ; скорость 9.6 kb
S4800 equ 0fah                  ; скорость 4.8 kb
S2400 equ 0f4h                  ; скорость 2.4 kb
S1200 equ 0e8h                  ; скорость 1.2 kb
; начальная инициализация UART
uart_ini: mov thl, #S9600       ; скорость UART
          orl tmod, #20h        ; Таймер 1 – в режим autoreload
          anl pcon, #not(80h)   ; скорость не удваивать
          orl tcon, #40h        ; запуск таймера 1
          mov scon, #50h     ; настройка последовательного канала
          clr ie.4           ; запрет прерывание от прием/перед.
          ret
```

Следующая программа иллюстрирует технологию пересылки данных с использованием УАПП. В данном примере ASCII код клавиши,

нажатой на клавиатуре компьютера, принимается УАПП стенда, отображается на светодиодных индикаторах и отсылается обратно в COM-порт компьютера для отображения на экране монитора T2, работающего в режиме эмуляции терминала (см. п. 3.4, с. 95). Можно наблюдать принятые байты в шестнадцатеричном коде (1 term) или в текстовом формате (0 term).

```
;*****************************************************************
; Пример программы, которая принимает байт от компьютера
; через УАПП, выводит его на светодиодную линейку и отсылает
; обратно через УАПП в COM-порт компьютера (эхопечать)
;*****************************************************************
org 2000h
         call uart_ini ; инициализация UART
; прием байта с выводом на линейку светодиодов
loop:    jnb scon.0, $ ; ожидание прихода байта
         mov a, sbuf   ; прием байта
         clr scon.0    ; сброс флага приемника
         call svdisp   ; вывод на светодиоды
; передача байта обратно через УАПП в COM-порт компьютера
         mov sbuf, a   ; передача байта обратно
         jnb scon.1, $ ; ожидание конца передачи байта
         clr scon.1    ; сброс флага передатчика
         sjmp loop
end
```

4.7. Шина I^2C

Краткие описания устройства и функционирования шины I^2C приведены в 3.1.4, с. 80. Кроме того, в 3.2.5 (с. 89) кратко описаны часы-календарь реального времени, подключенные к шине I^2C стенда SDK-1.1. Более подробную информацию о применении шины I^2C стенда SDK-1.1 можно найти в соответствующем руководстве пользователя [6].

Основные операции взаимодействия с шиной I^2C подготовлены и оформлены в библиотеку I2C. LIB, которую необходимо подклю-

чить к своему проекту (см. п. 3.3, с. 92). Следует различать I^2C-адрес устройства, подключенного к шине I^2C (см. табл. 19, с. 84), и адреса внутренних регистров устройства, находящихся во внутреннем адресном пространстве этого устройства (табл. 23, с. 90). Процедуры, содержащиеся в библиотеке I2C. LIB, используют 8-разрядный внутренний адрес для доступа во внутреннее адресное пространство I^2C-устройств, т. е. адресуемый объем составляет не более 256 байт. Микроконтроллер при этом функционирует в режиме ведущего (master). Ниже приведено описание трех процедур из библиотеки I2C. LIB, используемых при доступе в регистры внутреннего адресного пространства ведомых периферийных устройств, подключенных к шине I^2C [9].

GetAck осуществляет проверку готовности ведомого (slave) I^2C-устройства к обмену данными. С этой процедуры следует начинать любое обращение к I^2C-устройству. Входной параметр в регистрах микроконтроллера A — I^2C-адрес устройства.

Результат проверки готовности фиксируется в виде состояния флага F0: 0 — устройство не готово, 1 — устройство готово к обмену.

ReceiveBlock позволяет получить блок данных от ведомого (slave) I^2C-устройства. Блок данных при получении размещается во внешней памяти данных (xdata). Входные параметры процедуры записываются в регистрах микроконтроллера:

- A — I^2C-адрес устройства;
- B — длина блока данных, т. е. количество получаемых байтов;
- R0 — адрес начала блока-источника во внутреннем адресном пространстве I^2C-устройства;
- DPTR — адрес области в xdata, где будет размещен принятый блок.

При успешном завершении процедуры приема блока данных флаг F0 сброшен. При любой ошибке во время приема F0 = 1.

SendBlock позволяет передать блок данных в ведомое (slave) I^2C-устройство. Блок данных для отправки должен быть размещен во внешней памяти данных (xdata). Входные параметры процедуры записываются в регистрах микроконтроллера:

- A — I^2C-адрес устройства;
- B — длина блока данных, т. е. количество байтов для передачи;
- R0 — адрес начала блока-приемника во внутреннем адресном пространстве I^2C-устройства;
- DPTR — адрес области в xdata, где размещен блок данных для пересылки.

При успешном завершении процедуры приема блока данных флаг F0 сброшен. При любой ошибке во время приема F0 = 1.

Ниже показаны примеры программ, осуществляющих чтение и установку времени в часах. Директива EXTRN сообщает компилятору, что подпрограммы (code) с именами «receiveblock», «sendblock», «getack» необходимо взять из внешнего файла. В данном случае они содержатся в объектной библиотеке I2C.LIB. Другими словами, подобное декларирование структуры программы требуется редактору связей для сборки загрузочного модуля из нескольких объектных модулей, в данном случае — для объединения приведенной ниже программы и библиотеки I2C.LIB.

В подпрограммах GetTime (получение времени) и PutTime (установка времени) использованы описанные выше процедуры доступа к I^2C-устройствам [9]. Подпрограммы выдают содержимое регистров без какого-либо преобразования или форматирования.

```
;*****************************************************************
; Пример организации программы для получения времени
; (4 байта) из часов по шине I2C (или для установки времени
; в часах путем отправки 4 байтов по шине I2C), где используются
; функции из библиотеки I2C.LIB для работы с шиной I2C
;*****************************************************************
extrn code (receiveblock, sendblock, getack); декларация сегмента
                                              ; программы
org 2000h                ; установка адреса начала программы
; начало программы
         mov dptr, #5000h ; адрес области в xdata для размещения
                          ; времени
         call gettime     ; получить 4 байта из часов
         call puttime     ; отправить 4 байта в часы
         jnb F0, done     ; F0=0, нет ошибки
; здесь можно разместить обработчик ошибки при F0=1
; команды обработчика
; конец обработчика ошибок по флагу F0=1
done:    sjmp $           ; бесконечная петля
```

```
;***************************** GetTime ****************************
; Получение времени в виде 4-х байтов ДДК (BCD) из часов
; dptr – адрес области в xdata для размещения 4-х байтов
; Результат в F0:
; 0 – успешно получено; 1 – часы не откликаются
;******************************************************************
gettime: push acc           ; сохранение аккумулятора
         push b             ; сохранение B
         anl psw, #1100011b ; выбор банка 0, F0=0
         push 0h            ; сохранение r0
         mov a, #0a0h       ; I2C адрес часов
         call getack        ; F0=1 – часы готовы
         cpl F0             ; иначе – не готовы
         jb F0, gt_err      ; обработка ошибки
         mov b, #4          ; длина блока байтов
         mov r0, #1h        ; адрес регистра в часах
         call receiveblock  ; получить из часов
gt_err:  pop 0h             ; восстановление r0
         pop b              ; восстановление B
         pop acc            ; восстановление аккумулятора
         ret

;***************************** PutTime ****************************
; Установка времени в виде 4-х байтов ДДК (BCD)
; dptr – адрес области в xdata исходных 4-х байтов
; Результат в F0:
; 0 – успешно установлено; 1 – часы не откликаются
;******************************************************************
puttime: push acc           ; сохранение аккумулятора
         push b             ; сохранение B
         anl psw, #1100011b ; выбор банка 0, F0=0
         push 0h            ; сохранение r0
         mov a, #0a0h       ; I2C адрес часов
         call getack        ; F0=1 – часы готовы
         cpl F0             ; иначе – не готовы
         jb F0, pt_err      ; обработка ошибки
         mov b, #4          ; длина блока байтов
         mov r0, #1h        ; адрес регистра в часах
```

```
        call sendblock  ; передать в часы
pt_err: pop 0h          ; восстановление r0
        pop b           ; восстановление B
        pop acc         ; восстановление аккумулятора
        ret
end
```

4.8. АЦП

За установку параметров и работу АЦП отвечают регистры ADCCON1, ADCCON2 и ADCCON3 (см. 3.1.1, с. 67). Для указания параметров преобразования АЦП используется регистр ADCCON1. Рекомендуется использовать режим «Нормальный». Выбор номера канала для преобразования осуществляется младшей тетрадой регистра ADCCON2.

Ниже показан пример инициализации АЦП.

```
mov adccon1, #01101000b ; «нормальный» режим
mov adccon2, #00000000b ; выбор нулевого канала
```

Запуск однократного преобразования выполняется установкой бита SCONV (ADCCON2.4), который сбрасывается автоматически по завершении преобразования.

```
setb adccon2.4 ; запуск преобразования
jb adccon2.4,$ ; ожидание конца преобразования АЦП
```

4.9. ЦАП

Для установки параметров ЦАП используется регистр DACCON (см. 3.1.2, с. 73), в котором можно указать разрядность обоих ЦАП (8 или 12 бит), отдельно для каждого канала ЦАП выбрать источник опорного напряжения (ИОН) V_{dd} или V_{ref}, включить или выключить любой из ЦАП.

Ниже показан пример команды, инициализирующей ЦАП:

```
mov daccon, #11101101b ; включен только ЦАП0 в 8-битном
                       ; режиме, ИОН = 5 В, выход ЦАП
                       ; соответствует коду
```

Установка требуемого уровня напряжения (U) на выходе ЦАП зависит от напряжения ИОН ($U_{\text{ион}}$), разрядности ЦАП (n) и числа (D), записанного во входные регистры ЦАП. Расчет значения выполняется по формуле

$$D = \frac{U(2^n - 1)}{U_{\text{ион}}}.$$

Например, если используется 8-разрядный ЦАП0, ИОН — V_{dd} (5 В) и требуется установить выходное напряжение равное 2 В, то нужно записать команду

```
mov dac01, #102 ; 102=2(В)*255/5(В)
```

4.10. Организация циклов на языке ассемблера микроконтроллеров семейства MCS-51

Циклы предназначены для многократного выполнения определенной последовательности команд (инструкций), например: для вывода информации на ЖКИ, передачи данных через порт ввода-вывода, опроса внешних датчиков и многого другого. Повторяемая последовательность называется **телом цикла**, а единичное выполнение тела цикла — **итерацией**.

Условие продолжения цикла (или условие окончания цикла) определяет, будет ли в очередной раз выполняться итерация, или необходимо закончить выполнение и выйти из цикла.

Счетчик итераций цикла (или счетчик цикла) — переменная (аккумулятор, регистр или ячейка памяти), содержащая номер выполняемой итерации. Цикл может не содержать счетчик, когда алгоритм программы предусматривает выполнение итераций до возникновения какого-либо события, например появления на внешнем входе микрокон-

троллера логической 1.

Ниже показаны примеры организации циклов с конечным количеством повторов.

1. Цикл с использованием команды djnz, счетчик — регистр R7:

```
     mov r7, #20h ; количество повторов (начальное значение
                  ; счетчика)
m1:
     …            ; тело цикла
     djnz r7, m1  ; декрементирование счетчика и переход на
                  ; начало цикла (m1), пока его содержимое
                  ; не равно нулю
```

2. Цикл с использованием команды cjne, счетчик — аккумулятор:

```
     mov a, #20h      ; начальное значение счетчика
m2:
     …                ; тело цикла
     inc a            ; увеличение счетчика
     cjne a, #40h, m2 ; сравнение значения счетчика с
                      ; конечным значением (40h) и переход на
                      ; начало цикла (m2), если они не равны
```

3. Цикл с использованием команды cjne, счетчик — ячейка с адресом 30h.

```
     mov 30h, #20h   ; начальное значение счетчика
     mov a, #40h     ; конечное значение
m3:
     …               ; тело цикла
     inc 30h         ; увеличение счетчика
     cjne a, 30h, m3 ; сравнение значения счетчика (ячейки 30h)
                     ; с конечным значением (содержимое
                     ; аккумулятора) и переход на начало цикла
                     ; (m3), если они не равны
```

РАЗДЕЛ Б

Лабораторная работа № 1. Модель микроконтроллера семейства MCS-51

Цель работы

1. Освоить выполнение основных функций в системе моделирования микроконтроллера семейства MCS-51: ввод, отладку, ассемблирование, выполнение программы.
2. Освоить команды пересылки данных и основные математические операции языка ассемблера.
3. Изучить функциональные возможности таймер-счетчика микроконтроллера.

Теоретическая часть

Ознакомьтесь с описанием базового микроконтроллера семейства MCS-51 п. 1.1–1.4. (с. 4).

Практическая часть

Используя систему команд (см. п. 1.8, с. 37 и прил. 1), выполните задания в системе моделирования микроконтроллера семейства MCS-51 «Single-Chip Machine» (гл. 2, с. 59).

А. Программа управления таймер-счетчиком

Введите исходный код программы:

```
mov tmod, #02h
mov tl0, #E0h
mov tcon, #10h
L1:
jnb tf0, L1
```

Выполните в пошаговом режиме программу, определите назначение каждой команды и общую функцию программы.

Б. Программирование таймер-счетчика

Разработайте программу, организующую отсчет установленного числа итераций, с использованием указанного таймер-счетчика в определенном режиме. Функция таймер-счетчика — таймер. Варианты заданий приведены в табл. 24.

В. Программа расчета математической функции

Разработайте программу, выполняющую математические операции. Варианты заданий приведены в табл. 24. Числа a, b, c выбирайте самостоятельно. Для выполнения операций необходимо использовать регистры R2 (число a), R3 (число b), R4 (число c) и A. В начале программы расположите команды загрузки исходных чисел в регистры. Математические операции необходимо производить с выбранными регистрами. Последней командой разместите полученный результат в ячейку внутренней памяти данных с адресом 40h.

Содержание отчета

Отчет должен включать в себя:

- титульный лист (прил. 3);
- цель работы;
- исходные тексты разработанных программ с комментариями к каждой строке, показывающие суть команды в контексте разработанной программы, а не фактическое действие с регистрами, константами и т. п.;
- к заданию А приведите комментарии к каждой строке и общее описание работы программы. К заданию Б требуется, кроме ком-

ментариев, показать расчет начального значения, загружаемого в регистры таймер-счетчика. В задании В при написании комментариев показывайте суть команды в контексте данной программы, используя названия исходных переменных a, b и c;
- заключение и выводы.

Контрольные вопросы

1. Какие основные блоки базового микроконтроллера семейства MCS-51 вы знаете?
2. Как организована система памяти изучаемого семейства микроконтроллеров?
3. Назовите функции и режимы таймер-счетчика, его регистры управления.

Таблица 24

Варианты заданий Б и В

Номер варианта	Задание 2			Задание 3
	Номер таймер-счетчика	Режим таймер-счетчика	Количество итераций	Математическая операция
1	0	0	10h	$(a+b)\cdot c-4$
2	1	0	20h	$c-a\cdot b+2$
3	0	1	30h	$b/c+a-3$
4	1	1	08h	$(a+4)/(b-c)$
5	0	0	04h	$(a+c-b)/2$
6	1	0	10	$a-c/b+3$
7	0	1	15	$a\cdot b+c-2$
8	1	1	25h	$(a-b)/c+5$
9	0	0	21	$c\cdot b+a-3$
10	1	0	40h	$2+a/c-b$
11	0	1	50	$15-a\cdot(b+c)$
12	1	1	32	$(b-2)\cdot a/c$
13	0	0	64	$a+c-b/4$
14	1	0	12	$b\cdot a+c-5$
15	0	1	05h	$(a+4)/(b-c)$
16	1	1	48	$c/(a-b)+2$

Окончание табл. 24

Номер варианта	Задание 2			Задание 3
	Номер таймер-счетчика	Режим таймер-счетчика	Количество итераций	Математическая операция
17	0	0	18h	$b-5\cdot c+a$
18	1	0	16	$a\cdot b-c+4$
19	0	1	28h	$(a+2)\cdot c-b$
20	1	1	11	$4+b\cdot c-a$
21	0	0	1Ch	$b+c\cdot a-3$
22	1	0	35h	$(4+a)\cdot c-b$
23	0	1	17	$(a+4-b)/c$
24	1	1	12	$c+b\cdot a-5$
25	0	0	13h	$b/a-c+7$
26	1	0	24	$(b+3)/(c-a)$
27	0	1	1Ah	$12/(a-b)+c$
28	1	1	19	$b-c/b+a$
29	0	0	2Bh	$a/b+c-1$
30	1	0	21	$(a-b/c)+4$
31	0	1	31h	$20-a\cdot(b+c)$
32	1	1	9h	$(b+c)\cdot c/a$

4. Для каких целей устанавливается начальное значение таймер-счетчика? Как его вычислить?
5. Какие команды можно использовать для задания режимов таймер-счетчика, установки начального значения, его запуска?
6. Укажите основные команды математических операций на языке ассемблера.
7. В каком регистре находится, а также с какой целью используется бит переноса C в математических операциях?

Лабораторная работа № 2. Программирование периферийных устройств, доступных через регистры ПЛИС

Цель работы

Изучить технологию программирования периферийных устройств стенда SDK1.1, доступных через регистры ПЛИС.

Теоретическая часть

Ознакомьтесь с описаниями базового микроконтроллера семейства MCS-51 п. 1.1–1.3 (с. 4), лабораторного стенда SDK-1.1 — п. 3, 3.1, 3.2 (с. 64) и работой последовательного порта — п. 1.6 (с. 28).

Практическая часть

Для каждого задания создавайте отдельный проект в интегрированной среде Keil μVision IDE (см. п. 3.3, с. 90). Система команд микроконтроллера описана в п. 1.8 (с. 37), список команд содержится в прил. 1 (с. 131).

А. Программа для вывода информации на линейку светодиодных индикаторов

Ознакомитесь с устройствами, доступными через регистры ПЛИС (см. п. 3.2, с. 87). Для выполнения задания необходимо: создать проект, набрать программу (см. п. 4.2, с. 97) и подпрограмму PUTBYTE (см. п. 4.1, с. 96); разобраться в организации программы, провести ассемблирование, получить HEX-файл. Используя монитор T2 (см. п. 3.4, с. 93), загрузите программу в лабораторный стенд SDK-1.1 и получите правильный результат выполнения программы. В исходном тексте измените содержимое выводимого байта данных в соответствии с вариантом табл. 25 и убедитесь в правильности результата.

Предостережение: подпрограммы в файле следует всегда располагать после головной программы (см. 1.8.7, с. 57). Это избавит от возможной проблемы со стартовым адресом программы.

Таблица 25

Варианты задания

Номер варианта	Светящиеся светодиоды	Номер варианта	Светящиеся светодиоды	Номер варианта	Светящиеся светодиоды
1	7, 5, 1	12	5, 4, 3	23	6, 4, 1
2	4, 3, 2	13	7, 4, 0	24	3, 2, 0
3	6, 3, 2	14	6, 5, 3	25	7, 6, 4
4	7, 3, 2	15	7, 3, 0	26	7, 4, 2
5	5, 4, 1	16	7, 4, 1	27	6, 1, 0
6	6, 5, 1	17	6, 3, 0	28	7, 5, 2
7	7, 6, 0	18	7, 4, 3	29	5, 3, 1
8	5, 3, 0	19	4, 3, 0	30	7, 5, 0
9	7, 6, 2	20	5, 3, 2	31	6, 2, 0
10	6, 3, 1	21	7, 5, 4	32	4, 2, 1
11	7, 6, 3	22	4, 1, 0	33	7, 6, 5

Б. Программа для вывода информации на жидкокристаллический дисплей

Изучите организацию подпрограммы PUTCHAR и технологию управления жидкокристаллическим индикатором (см. 3.2.2, с. 88 и п. 4.3, с. 98). Создайте новый проект, наберите подпрограмму PUTCHAR, составьте головную программу для вывода своего имени на ЖКИ с использованием подпрограммы PUTCHAR. Изучите команды управления курсором ЖКИ и выведите свое имя в середину первой строки ЖКИ, а фамилию в середину второй. ЖКИ имеет встроенный знакогенератор, поддерживающий символы в ASCII-кодах и кириллицу в нестандартной кодировке (прил. 4).

В. Опрос матричной клавиатуры

Изучите особенности построения и работы клавиатуры (см. 3.2.1, с. 87). Откройте новый проект, наберите подпрограмму GETKEY (см. п. 4.4, с. 99), составьте головную программу для опроса клавиатуры с помощью подпрограммы GETKEY и вывода кода нажатой клавиши на светодиодные индикаторы. Подпрограммы следует распо-

лагать после головной программы (см. 1.8.7, с. 57). Составьте таблицу кодов клавиатуры.

Г. Звуковой излучатель

Изучите способы генерации звука, используя встроенный излучатель звука (см. 3.2.4, с. 89 и п. 4.5, с. 101). Откройте новый проект, наберите программу генерации звука и получите звук. Программным способом измените громкость и высоту тона (частоту).

Д. Программирование УАПП

Изучите встроенный универсальный асинхронный приемопередатчик (см. п. 1.6, с. 28). Наберите головную программу, принимающую байт с линии, передающую этот байт обратно в линию и на светодиодные индикаторы (эхопечать), а затем подпрограмму для инициализации УАПП UART_INI (см. п. 4.6, с. 102). Изучите работу монитора Т2 в режиме эмуляции терминала (см. с. 95). Переключив монитор Т2 в режим эмуляции терминала, проверьте наличие передачи данных с клавиатуры компьютера (при нажатии клавиш) через стенд SDK1.1 на экран компьютера. Проведите данный эксперимент в двух режимах: при отображении принятых данных в символьном (бинарном) и шестнадцатеричном виде. Изучите режим работы Т2 при записи в файл информации, приходящей со стенда (см. с. 95). Запишите эхопечать в текстовый файл. Выйдя из монитора Т2 (команда bye), просмотрите содержимое этого текстового файла средствами программы Блокнот (Notepad.exe).

Содержание отчета

Отчет о работе должен содержать:

- титульный лист (прил. 3);
- цель работы;
- исходные тексты разработанных программ с комментариями к каждой строке или блоку, показывающие суть операций в контексте разработанной программы, а не фактическое действие с регистрами, константами и т. п.;
- к заданию Б приведите фото ЖКИ стенда с вашими именем и фамилией. К заданию В требуется показать таблицу кодов клави-

атуры и описать принцип работы клавиатур матричного типа. В задании Г поясните способ изменения громкости и частоты генерируемого звука. В задании Д продемонстрируйте фото или скриншот текстового файла и монитора Т2 с записанной последовательностью символов эхопечати;
- заключение и выводы.

Контрольные вопросы

1. Какие основные блоки микропроцессорного стенда SDK-1.1 вы знаете?
2. Какие устройства доступны через регистры ПЛИС? Какая подпрограмма осуществляет запись информации в регистры ПЛИС? Поясните суть ее работы.
3. Назовите регистр, управляющий светодиодной линейкой. Каким образом можно включить и выключить конкретный светодиод?
4. При помощи каких регистров ПЛИС осуществляется взаимодействие с ЖКИ? Какие линии связи используются для обмена информацией с ЖКИ? Опишите последовательность генерации этих сигналов. Чем будут отличаться передаваемые сигналы в ЖКИ при записи команды и данных?
5. Какая подпрограмма предназначена для взаимодействия с ЖКИ? Поясните ее функционал.
6. Как формируется команда позиционирования курсора? Зачем после команды очистки экрана необходимо делать задержку?
7. Опишите общий принцип организации матричных клавиатур. Какой регистр служит для работы с клавиатурой? Назовите поля этого регистра, при этом укажите направление передачи информации. Как организован опрос клавиатуры в подпрограмме getkey?
8. Каким способом генерируется звук в стенде SDK-1.1? Как изменить частоту и громкость звуковых колебаний?
9. Назовите основную цель предназначения УАПП, его основные параметры. Укажите регистры для организации взаимодействия с УАПП. Для чего используется монитор Т2 в режиме терминала? Как осуществляется переключение в режим эмуляции терминала, в режим эхопечати?

Лабораторная работа № 3. Программирование периферийных устройств, доступных через шину I^2C

Цель работы

Получить навыки программирования периферийных устройств, подключенных к микропроцессорной системе по шине I^2C.

Теоретическая часть

Ознакомьтесь с описаниями микроконтроллера семейства MCS-51 (п. 1.1–1.3, с. 4) и лабораторного стенда SDK-1.1 (с. 64): интерфейс I^2C (с. 80), регистры, доступные через ПЛИС (п. 3.2, с. 87), встроенные часы реального времени (3.2.5, с. 89).

Практическая часть

Программирование микросхемы часов реального времени

Изучите подпрограммы взаимодействия с часами через интерфейс I^2C (п. 4.7, с. 104).

Создайте новый проект в интегрированной среде Keil μVision IDE (см. п. 3.3, с. 90), подключите библиотеку I2C. LIB. Изучите технологию доступа к часам реального времени и разберитесь с форматом представления времени в часах. Напишите программу, позволяющую:

- применяя подпрограмму PutTime, установить в часах текущее время;
- используя бесконечный цикл, на основе подпрограммы GetTime считывать содержимое часов и выводить полученное значение:

вариант 1 — на экран компьютера средствами Т2 в режиме эмуляции терминала. Здесь необходимо применить созданные в лабораторной работе 2 программы работы с УАПП.

вариант 2 — на ЖКИ стенда, используя созданные в лабораторной работе 2 программы работы с ЖКИ.

Содержание отчета

В отчет о лабораторной работе должны быть включены:

- титульный лист (прил. 3);
- цель работы;
- исходные тексты разработанных программ с комментариями к каждой строке или блоку, показывающие суть операций в контексте разработанной программы, а не фактическое действие с регистрами, константами и т. п.;
- в практической части приведите фото ЖКИ стенда или экрана компьютера с выведенным временем;
- заключение и выводы.

Контрольные вопросы

1. Какие линии связи используются в интерфейсе I^2C? Опишите формат передачи данных по шине I^2C.
2. Сколько ведомых и ведущих устройств позволяет подключать шина I^2C?
3. Укажите основные отличия интерфейсов I^2C и SPI.
4. Какие устройства подключены к шине I^2C в лабораторном стенде SDK-1.1?
5. Назовите регистры управления шиной I^2C. Какие режимы и параметры шины позволяют конфигурировать эти регистры?
6. Опишите предназначение, входные и выходные параметры процедур, содержащихся в библиотеке I2C.LIB. Поясните функционал процедур, используемых в работе для взаимодействия с часами реального времени (gettime и puttime).
7. В каком формате представляются данные в микросхеме часы-календарь типа PCF8583? Как осуществить вывод времени на ЖКИ, в порт УАПП?

Лабораторная работа № 4. Система прерываний

Цель работы

Изучить работу системы прерываний микроконтроллеров семейства MCS-51 и получить навыки ее программирования.

Теоретическая часть

Ознакомьтесь с описанием микроконтроллера семейства MCS-51 (п. 1.1, с. 4), системы прерываний базового микроконтроллера (п. 1.7, с. 34) и микроконтроллера ADuC812 (3.1.3, с. 73).

Практическая часть

Создайте новый проект в интегрированной среде Keil μVision IDE (см. п. 3.3, с. 90). Разработайте программу, выводящую текущее значение определенной ячейки памяти (например, 40h) на светодиодную линейку стенда SDK-1.1 (см. п. 4.2, с. 97), при возникновении прерывания по INT0 или INT1 значение выбранной ячейки должно соответственно увеличиваться или уменьшаться.

Предлагаемая структура программы:

1) основная программа — установка значений регистров ENA, TCON, IE для использования прерываний INT0 и INT1;

2) подпрограмма вывода значения выбранной ячейки на светодиодную линейку;

3) подпрограмма обработки прерывания INT0 — инкрементирование значения выбранной ячейки и его вывод;

4) подпрограмма обработки прерывания INT1 — декрементирование значения выбранной ячейки и его вывод.

Из-за ручной генерации прерывания с помощью подключенных внешних кнопок, следует при разработке предусмотреть программное

исключение «дребезга контактов». Для этого создайте программную задержку с помощью вложенного цикла (рис. 27) или путем многократного запуска таймер-счетчика в режиме прерывания (рис. 28 и рис. 29).

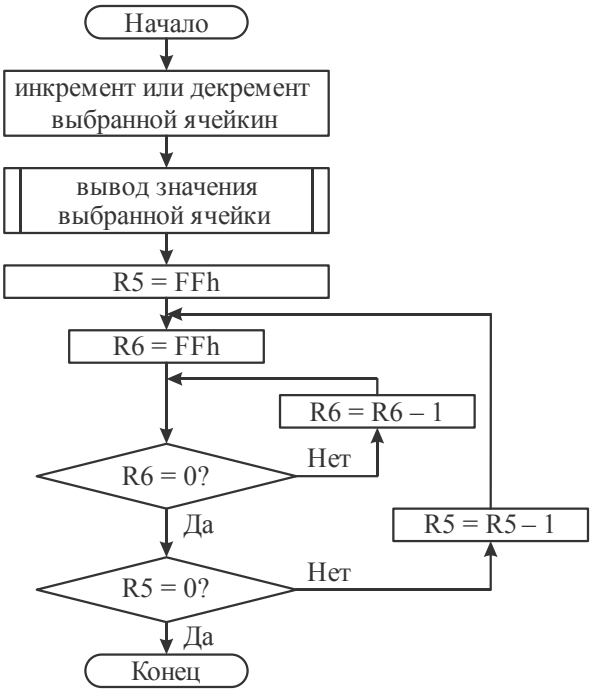


Рис. 27. Блок-схема алгоритма подпрограммы обработки прерывания с учетом исключения «дребезга контактов» при помощи вложенного цикла

Рис. 28. Блок-схема алгоритма подпрограммы обработки прерывания с учетом исключения «дребезга контактов» при помощи таймер-счетчика

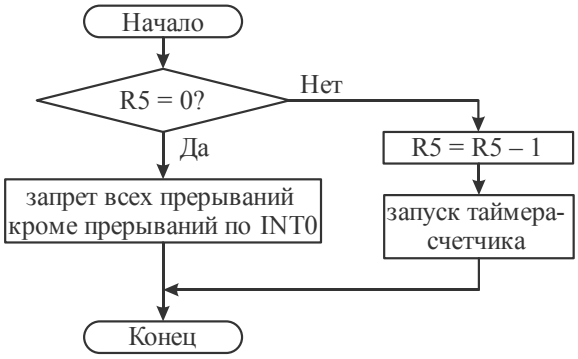


Рис. 29. Блок-схема алгоритма подпрограммы обработки прерывания таймер-счетчика

Содержание отчета

Отчет о работе должен содержать:

- титульный лист (прил. 3);
- цель работы;
- исходные тексты разработанных программ с комментариями к каждой строке или блоку, показывающие суть операций в контексте разработанной программы, а не фактическое действие с регистрами, константами и т. п.;
- заключение и выводы.

Контрольные вопросы

1. Опишите предназначение, общий принцип работы и основные возможности системы прерываний микроконтроллеров семейства MCS-51.
2. Какое из двух одновременно возникших разноуровневых прерываний будет выполняться, а при возникновении одноуровневых?
3. Назовите регистры базового микроконтроллера, конфигурирующие систему прерываний, последовательность их настройки, а также процесс инициализации системы прерываний.
4. В чем заключаются отличия систем прерываний микроконтроллеров Intel 8051 и ADuC812?
5. Покажите способы и поясните отличия записи значений в регистры ENA, TCON и IE.

Лабораторная работа № 5. Генератор импульсного сигнала

Цель работы

Приобрести навыки программирования ЦАП микроконтроллера ADuC812 и освоить разработку программ генерации импульсных сигналов.

Теоретическая часть

Ознакомьтесь с описанием микроконтроллера семейства MCS-51 (п. 1.1, с. 4), блока ЦАП микроконтроллера ADuC812 (3.1.2, с. 72).

Практическая часть

Изучите способы организации циклов на языке ассемблер (см. п. 4.10, с. 109), рассмотрите пример инициализации ЦАП (п. 4.9, с. 108).

Создайте новый проект в интегрированной среде Keil μVision IDE (см. п. 3.3, с. 90). В соответствии со своим вариантом (табл. 26) разработайте программу для формирования импульсного сигнала.

При написании программы используйте 8-разрядный режим работы и 1 канал ЦАП, выберите оптимальный для вашего варианта источник опорного напряжения, задав соответствующее значение в регистре DACCON.

Содержание отчета

В отчете должны содержаться:

- титульный лист (прил. 3);
- цель работы;
- параметры и форма сигнала с указанием напряжения и соответствующего кода в контрольных точках, исходные тексты разрабо-

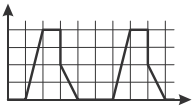
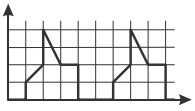
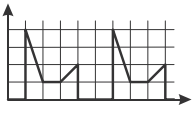
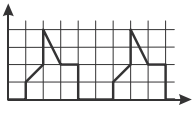
танных программ с комментариями к каждой строке или блоку, показывающие суть операций в контексте разработанной программы, а не фактическое действие с регистрами, константами и т. п.;
- фотография экрана осциллографа с полученным сигналом;
- заключение и выводы.

Контрольные вопросы

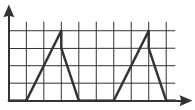
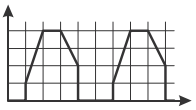
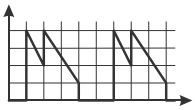
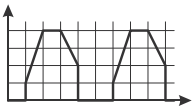
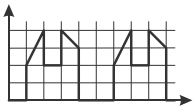
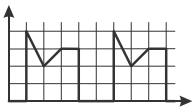
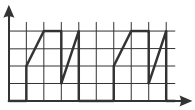
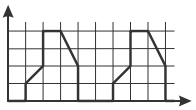
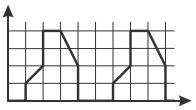
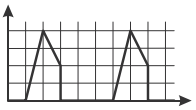
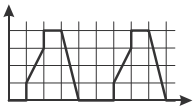
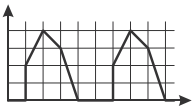
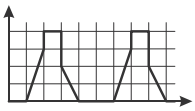
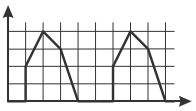
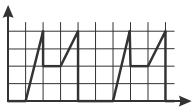
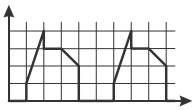
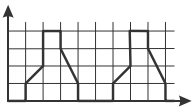
1. Назовите основные параметры ЦАП микроконтроллера ADuC812. Какие регистры позволяют управлять ЦАП?
2. С какой целью в ЦАП микроконтроллера ADuC812 могут использоваться два источника опорного напряжения — V_{ref} (2,5 В) и V_{dd} (5 В)? Как выбрать оптимальный источник опорного напряжения?
3. Как сформировать увеличение или уменьшение амплитуды сигнала в реальном времени на выходе ЦАП? Каким приемом можно воспользоваться для варьирования длительности сигнала без изменения его формы? Предложите способ программной генерации паузы между импульсами.
4. Подготовьте пример исходного кода программы, предназначенной для реализации линейного увеличения амплитуды выходного напряжения ЦАП в диапазоне 1–1,5 В за период времени 250 мкс.

Таблица 26

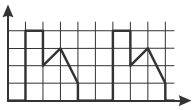
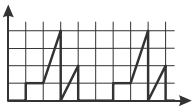
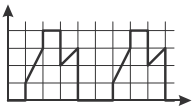
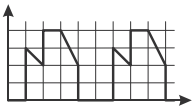
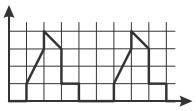
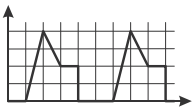
Варианты задания

Номер варианта	Форма сигнала	Параметры сигнала	Номер варианта	Форма сигнала	Параметры сигнала
1		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 0{,}5$ В	19		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 0{,}5$ В
2		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1$ В	20		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1$ В

Продолжение табл. 26

Номер варианта	Форма сигнала	Параметры сигнала	Номер варианта	Форма сигнала	Параметры сигнала
3		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1{,}5$ В	21		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1{,}5$ В
4		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 2$ В	22		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 2$ В
5		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 2{,}5$ В	23		$T = 1250$ мкс $\tau_{min} = 500$ мкс $U_{max} = 2{,}5$ В
6		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 3$ В	24		$T = 1250$ мкс $\tau_{min} = 500$ мкс $U_{max} = 3$ В
7		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 3{,}5$ В	25		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 3{,}5$ В
8		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 4$ В	26		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 4$ В
9		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 4{,}5$ В	27		$T = 1250$ мкс $\tau_{min} = 500$ мкс $U_{max} = 4{,}5$ В
10		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 5$ В	28		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 5$ В
11		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 0{,}5$ В	29		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 0{,}5$ В
12		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1$ В	30		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1$ В
13		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1{,}5$ В	31		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1{,}5$ В

Окончание табл. 26

Номер варианта	Форма сигнала	Параметры сигнала	Номер варианта	Форма сигнала	Параметры сигнала
14		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 2$ В	32		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 2$ В
15		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 2{,}5$ В	33		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 2{,}5$ В
16		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 3$ В	34		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 3$ В
17		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 3{,}5$ В	35		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 3{,}5$ В
18		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 4$ В	36		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 4$ В

Лабораторная работа № 6. Измерение напряжения

Цель работы

Приобрести навыки разработки программ измерения напряжений при помощи АЦП микроконтроллера ADuC812.

Теоретическая часть

Ознакомьтесь с описанием микроконтроллера семейства MCS-51 (п. 1.1, с. 4) и блоком АЦП микроконтроллера ADuC812 (3.1.1, с. 67).

Практическая часть

Рассмотрите пример инициализации и запуска преобразования АЦП (см. п. 4.8, с. 108). Создайте новый проект в интегрированной среде Keil μVision IDE (см. п. 3.3, с. 90). Разработайте программу, выводящую на экран ЖКИ стенда входное напряжение измеренное на канале 0 встроенного АЦП.

Предлагается следующая структура программы.

1) Основная программа:

а) инициализация АЦП — 1 режим (нормальный), делитель тактовой частоты для АЦП — 4 (т. к. оптимальный рабочий частотный диапазон АЦП составляет 400 кГц — 3 МГц), число задержки запуска АЦП — 3;

б) основной цикл программы — измерение кода АЦП, преобразование полученного кода в значение напряжения и вывод его на ЖКИ;

2) подпрограмма вывода одного байта на ЖКИ (PutChar).

В данном задании не требуется большой точности, поэтому достаточно использовать 8-битное значение измеренного напряжения, кроме того, такой подход облегчает математические операции. С этой целью необходимо полученное 12-битное число преобразовать в 8-разрядное, убрав наименее значимую часть — 4 младших бита исходного числа. Затем полученное 8-битное значение кода преобразовать в напряжение, учитывая, что источником опорного напряжения является V_{ref} (2,5 В), и вывести на ЖКИ, не забыв перекодировать в ASCII последовательность. Обратите внимание: каждый канал АЦП имеет входной резистивный делитель на 2, выполненный при помощи двух последовательно соединенных резисторов номиналом 10 кОм.

Содержание отчета

Отчет о работе включает:

- титульный лист (прил. 3);
- цель работы;
- исходные тексты разработанных программ с комментариями к каждой строке или блоку, показывающие суть операций в кон-

тексте разработанной программы, а не фактическое действие с регистрами, константами и т. п.;
- в практической части описание алгоритма преобразования измеренного кода АЦП в напряжение;
- заключение и выводы.

Контрольные вопросы

1. Назовите основные параметры АЦП микроконтроллера ADuC812.
2. Какие регистры позволяют управлять АЦП?
3. Опишите режимы работы, поддерживающие АЦП микроконтроллером ADuC812.
4. Предложите алгоритм перевода числа 0–255 в напряжение 0–2,5 В, а затем в ASCII-код для вывода на ЖКИ.

Использованный библиографический список

1. Справочник. Микроконтроллеры: архитектура, программирование, интерфейс / В. Б. Бродин, М. И Шагурин. Москва: ЭКОМ, 1999. 400 с.
2. MCS 51. Microcontroller Family User's Manual [Электронный ресурс]. Режим доступа: http://web.mit.edu/6.115/www/document/8051.pdf. Загл. с экрана.
3. Каспер Э. Программирование на языке ассемблера для микроконтроллеров семейства i8051 / Э. Каспер. Москва : Горячая линия-Телеком, 2003. 192 с.
4. Огородников И. Н. Микропроцессорная техника: учебник / И. Н. Огородников. 2-е изд., перераб. и доп. Екатеринбург : УГТУ-УПИ, 2007. 380 с.
5. Описание программы Single-Chip Machine [Электронный ресурс]. Режим доступа: http://dca.narod.ru/simulation/mk51.htm (файл SCMHELP. HLP). Загл. с экрана.
6. Учебный стенд SDK-1.1 : руководство пользователя [Электронный ресурс]. Санкт-Петербург : ЛМТ, 2006. Режим доступа: http://lmt.cs.ifmo.ru. (файл sdk11_userm_v1_0_11.pdf). Загл. с экрана.
7. Спецификация ADuC812 [Электронный ресурс] / пер. с англ. Б. Л. Горшкова, Ю. М. Зайцева, В. И. Силантьева. Санкт-Петербург : АВТЭКС, 1999. Режим доступа: http://www.autex.spb.ru. (файл aduc812_rus.pdf). Загл. с экрана.
8. ADuC812. MicroConverter®, Multichannel 12-Bit ADC with Embedded Flash MCU [Электронный ресурс]. Режим доступа: http://www.analog.com/(файл ADUC812.pdf). Загл. с экрана.
9. Огородников И. Н. Микропроцессорная техника : практический курс [учеб. пособие] / И. Н. Огородников. Екатеринбург : УрФУ, 2012. 137 с.

Приложение 1
Система команд микроконтроллеров семейства MCS-51

Rn (n = 0, 1, ..., 7) — регистр общего назначения в выбранном банке регистров;

@Ri (i = 0, 1) — регистр общего назначения в выбранном банке регистров, используемый в качестве регистра косвенного адреса;

ad — адрес прямоадресуемого байта;

ads — адрес прямоадресуемого байта-источника;

add — адрес прямоадресуемого байта-получателя;

ad11–11-разрядный абсолютный адрес перехода;

ad16–16-разрядный абсолютный адрес перехода;

rel — относительный адрес перехода;

#d — непосредственный операнд;

#d16 — непосредственный операнд (2 байта);

bit — адрес прямоадресуемого бита;

/bit — инверсия прямоадресуемого бита;

A — аккумулятор;

PC — счетчик команд;

DPTR — регистр указатель данных;

() — содержимое ячейки памяти или регистра.

В таблице указаны название команды, мнемокод, длина в байтах (Б), время выполнения в машинных циклах (Ц), а также производимая операция.

Название команды	Мнемокод	Б	Ц	Операция
Команды передачи данных				
Пересылка в аккумулятор из регистра (n=0...7)	MOV A, Rn	1	1	(A) ← (Rn)
Пересылка в аккумулятор прямоадресуемого байта	MOV A, ad	2	1	(A) ← (ad)
Пересылка в аккумулятор байта из РПД (i=0,1)	MOV A, @Ri	1	1	(A) ← ((Ri))
Загрузка в аккумулятор константы	MOV A, #d	2	1	(A) ← #d
Пересылка в регистр из аккумулятора	MOV Rn, A	1	1	(Rn) ← (A)

Название команды	Мнемокод	Б	Ц	Операция
Пересылка в регистр прямоадресуемого байта	MOV Rn, ad	2	2	(Rn) ← (ad)
Загрузка в регистр константы	MOV Rn, #d	2	1	(Rn) ← #d
Пересылка по прямому адресу аккумулятора	MOV ad, A	2	1	(ad) ← (A)
Пересылка по прямому адресу регистра	MOV ad, Rn	2	2	(ad) ← (Rn)
Пересылка прямоадресуемого байта по прямому адресу	MOV add, ads	3	2	(add) ← (ads)
Пересылка байта из РПД по прямому адресу	MOV ad, @Ri	2	2	(ad) ← ((Ri))
Пересылка по прямому адресу константы	MOV ad, #d	3	2	(ad) ← #d
Пересылка в РПД из аккумулятора	MOV @Ri, A	1	1	((Ri))←(A)
Пересылка в РПД прямоадресуемого байта	MOV @Ri, ad	2	2	((Ri)) ← (ad)
Пересылка в РПД константы	MOV @Ri, #d	2	1	((Ri)) ← #d
Загрузка указателя данных	MOV DPTR, #d16	3	2	(DPTR) ← #d16
Пересылка в аккумулятор байта из ПП	MOVC A, @ A+DPTR	1	2	(A) ← ((A)+(DPTR))
Пересылка в аккумулятор байта из ПП	MOVC A, @ A+PC	1	2	(PC) ← (PC)+1, (A) ← ((A)+(PC))
Пересылка в аккумулятор байта из ВПД	MOVX A, @Ri	1	2	(A) ← ((Ri))
Пересылка в аккумулятор байта из расширенной ВПД	MOVX A,@DPTR	1	2	(A) ← ((DPTR))
Пересылка в ВПД из аккумулятора	MOVX @Ri, A	1	2	((Ri)) ← (A)
Пересылка в расширенную ВПД из аккумулятора	MOVX @DPTR, A	1	2	((DPTR)) ← (A)
Загрузка в стек	PUSH ad	2	2	(SP) ← (SP)+1, ((SP)) ← (ad)
Извлечение из стека	POP ad	2	2	(ad) ← (SP), (SP) ← (SP)–1
Обмен аккумулятора с регистром	XCH A, Rn	1	1	(A) ↔ (Rn)

Название команды	Мнемокод	Б	Ц	Операция
Обмен аккумулятора с прямоадресуемым байтом	XCH A, ad	2	1	(A) ↔ (ad)
Обмен аккумулятора с байтом из РПД	XCH A, @Ri	1	1	(A) ↔ ((Ri))
Обмен младших тетрад аккумулятора и байта РПД	XCHD A, @Ri	1	1	$(A_{0...3}) \leftrightarrow ((Ri)_{0...3})$
Арифметические операции				
Сложение аккумулятора с регистром (n=0...7)	ADD A, Rn	1	1	(A) ← (A)+(Rn)
Сложение аккумулятора с прямоадресуемым байтом	ADD A, ad	2	1	(A) ← (A)+(ad)
Сложение аккумулятора с байтом из РПД (i = 0,1)	ADD A, @Ri	1	1	(A) ← (A)+((Ri))
Сложение аккумулятора с константой	ADD A, #d	2	1	(A) ← (A)+#d
Сложение аккумулятора с регистром и переносом	ADDC A, Rn	1	1	(A) ← (A)+(Rn)+(C)
Сложение аккумулятора с прямоадресуемым байтом и переносом	ADDC A, ad	2	1	(A) ← (A)+(ad)+(C)
Сложение аккумулятора с байтом из РПД и переносом	ADDC A, @Ri	1	1	(A) ← (A) + ((Ri)) + (C)
Сложение аккумулятора с константой и переносом	ADDC A, #d	2	1	(A) ← (A)+# d+(C)
Десятичная коррекция аккумулятора	DAA	1	1	Если $(A_{0...3}) > 9$ или ((AC)=1), то $(A_{0...3}) \leftarrow$ $\leftarrow (A_{0...3})+6$, затем если $(A_{4...7}) > 9$ или ((C)=1), то $(A_{4...7}) \leftarrow (A_{4...7})+6$
Вычитание из аккумулятора регистра и заема	SUBB A, Rn	1	1	(A) ← (A)−(C)−(Rn)
Вычитание из аккумулятора прямоадресуемого байта и заема	SUBB A, ad	2	1	(A) ← (A)−(C)−((ad))
Вычитание из аккумулятора байта РПД и заема	SUBB A, @Ri	1	1	(A) ← (A)−(C)−((Ri))
Вычитание из аккумулятора константы и заема	SUBB A, #d	2	1	(A) ← (A)−(C)−#d
Инкремент аккумулятора	INC A	1	1	(A) ← (A)+1

Название команды	Мнемокод	Б	Ц	Операция
Инкремент регистра	INC Rn	1	1	(Rn) ← (Rn)+1
Инкремент прямоадресуемого байта	INC ad	2	1	(ad) ← (ad)+1
Инкремент байта в РПД	INC @Ri	1	1	((Ri)) ← ((Ri))+1
Инкремент указателя данных	INC DPTR	1	2	(DPTR) ← (DPTR)+1
Декремент аккумулятора	DEC A	1	1	(A) ← (A)–1
Декремент регистра	DEC Rn	1	1	(Rn) ← (Rn)–1
Декремент прямоадресуемого байта	DEC ad	2	1	(ad) ← (ad)–1
Декремент байта в РПД	DEC @Ri	1	1	((Ri)) ← ((Ri))–1
Умножение аккумулятора на регистр B	MUL AB	1	4	(B) (A) ← (A)*(B)
Деление аккумулятора на регистр B	DIV AB	1	4	(A).(B) ← (A)/(B)
Логические операции				
Логическое И аккумулятора и регистра	ANL A, Rn	1	1	(A) ← (A) AND (Rn)
Логическое И аккумулятора и прямоадресуемого байта	ANL A, ad	2	1	(A) ← (A) AND (ad)
Логическое И аккумулятора и байта из РПД	ANL A, @Ri	1	1	(A) ← (A) AND ((Ri))
Логическое И аккумулятора и константы	ANL A, #d	2	1	(A) ← (A) AND #d
Логическое И прямоадресуемого байта и аккумулятора	ANL ad, A	2	1	(ad) ← (ad) AND (A)
Логическое И прямоадресуемого байта и константы	ANL ad, #d	3	2	(ad) ← (ad) AND #d
Логическое ИЛИ аккумулятора и регистра	ORL A, Rn	1	1	(A) ← (A) OR (Rn)
Логическое ИЛИ аккумулятора и прямоадресуемого байта	ORL A, ad	2	1	(A) ← (A) OR (ad)
Логическое ИЛИ аккумулятора и байта из РПД	ORL A, @Ri	1	1	(A) ← (A) OR ((Ri))
Логическое ИЛИ аккумулятора и константы	ORL A, #d	2	1	(A) ← (A) OR #d
Логическое ИЛИ прямоадресуемого байта и аккумулятора	ORL ad, A	2	1	(ad) ← (ad) OR (A)

Название команды	Мнемокод	Б	Ц	Операция
Логическое ИЛИ прямоадресуемого байта и константы	ORL ad, #d	3	2	(ad) ← (ad) OR #d
Исключающее ИЛИ аккумулятора и регистра	XRL A, Rn	1	1	(A) ← (A) XOR (Rn)
Исключающее ИЛИ аккумулятора и прямоадресуемого байта	XRL A, ad	2	1	(A) ← (A) XOR (ad)
Исключающее ИЛИ аккумулятора и байта из РПД	XRL A, @Ri	1	1	(A) ← (A) XOR ((Ri))
Исключающее ИЛИ аккумулятора и константы	XRL A, #d	2	1	(A) ← (A) XOR #d
Исключающее ИЛИ прямоадресуемого байта и аккумулятора	XRL ad, A	2	1	(ad) ← (ad) XOR (A)
Исключающее ИЛИ прямоадресуемого байта и константы	XRL ad, #d	3	2	(ad) ← (ad) XOR #d
Сброс аккумулятора	CLR A	1	1	(A) ← 0
Инверсия аккумулятора	CPL A	1	1	(A) ← NOT (A)
Сдвиг аккумулятора влево циклический	RL A	1	1	$(A_{n+1}) \leftarrow (An)$, n=0÷6, $(A_0) \leftarrow (A_7)$
Сдвиг аккумулятора влево через перенос	RLC A	1	1	$(A_{n+1}) \leftarrow (A_n)$, n=0÷6, $(A_0) \leftarrow (C)$, $(C) \leftarrow (A_7)$
Сдвиг аккумулятора вправо циклический	RR A	1	1	$(A_n) \leftarrow (A_{n+1})$, n=0÷6, $(A_7) \leftarrow (A_0)$
Сдвиг аккумулятора вправо через перенос	RRC A	1	1	$(A_n) \leftarrow (A_{n+1})$, n=0÷6, $(A_7) \leftarrow (C)$, $(C) \leftarrow (A_0)$
Обмен местами тетрад в аккумуляторе	SWAP A	1	1	$(A_{0...3}) \leftrightarrow (A_{4...7})$
Операции с битами				
Сброс переноса	CLR C	1	1	(C) ← 0
Сброс бита	CLR bit	2	1	(bit) ← 0
Установка переноса	SETB C	1	1	(C) ← 1
Установка бита	SETB bit	2	1	(bit) ← 1
Инверсия переноса	CPL C	1	1	(C)←NOT (C)
Инверсия бита	CPL bit	2	1	(bit) ← NOT (bit)
Логическое И бита и переноса	ANL C, bit	2	2	(C) ← (C) AND (bit)

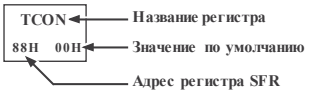
Название команды	Мнемокод	Б	Ц	Операция
Логическое И инверсии бита и переноса	ANL C,/bit	2	2	(C)←(C) AND (NOT (bit))
Логическое ИЛИ бита и переноса	ORL C, bit	2	2	(C) ← (C) OR (bit)
Логическое ИЛИ инверсии бита и переноса	ORL C,/bit	2	2	(C) ← (C) OR (NOT (bit))
Пересылка бита в перенос	MOV C, bit	2	1	(C) ← (bit)
Пересылка переноса в бит	MOV bit, C	2	2	(bit) ← (C)
Команды передачи управления				
Длинный переход в полном объеме ПП	LJMP ad16	3	2	(PC) ← ad16
Абсолютный переход внутри страницы в 2 Кб	AJMP ad11	2	2	(PC) ← (PC)+2, (PC_{0-10}) ← ad11
Короткий относительный переход внутри страницы в 256 байт	SJMP rel	2	2	(PC) ← (PC)+2, (PC) ← (PC)+rel
Косвенный относительный переход	JMP @A+DPTR	1	2	(PC) ← (A)+(DPTR)
Переход, если аккумулятор равен нулю	JZ rel	2	2	(PC) ← (PC)+2, если (A)=0, то (PC) ← (PC)+rel
Переход, если аккумулятор не равен нулю	JNZ rel	2	2	(PC) ← (PC)+2, если (A)≠0, то (PC) ← (PC)+rel
Переход, если перенос равен единице	JC rel	2	2	(PC) ← (PC)+2, если (C)=1, то (PC) ← (PC)+rel
Переход, если перенос равен нулю	JNC rel	2	2	(PC) ← (PC)+2, если (C)=0, то (PC) ← (PC)+rel
Переход, если бит равен единице	JB bit, rel	3	2	(PC)←(PC)+3, если (b)=l, то (PC) (PC)+rel
Переход, если бит равен нулю	JNB bit, rel	3	2	(PC) ← (PC)+3, если (b)=0, то (PC) ← (PC)+rel
Переход, если бит установлен, с последующим сбросом бита	JBC bit, rel	3	2	(PC) ← (PC)+3, если (b)=1, то (b) ← 0 и (PC) ← (PC)+rel
Декремент регистра и переход, если не нуль	DJNZ Rn, rel	2	2	(PC) ← (PC)+2, (Rn) ← (Rn) — 1, если (Rn) ≠ 0, то (PC) ← ←(PC)+rel

Название команды	Мнемокод	Б	Ц	Операция
Декремент прямоадресуемого байта и переход, если не нуль	DJNZ ad, rel	3	2	(PC) ← (PC)+2, (ad) ← (ad)−1, если (ad) ≠ 0, то (PC) ← ←(PC)+rel
Сравнение аккумулятора с прямоадресуемым байтом и переход, если не равно	CJNE A, ad, rel	3	2	(PC) ← (PC)+3, если (A) ≠ (ad), то (PC)←(PC)+ +rel, если (A) < (ad), то (C) ← 1, иначе (C) ← 0
Сравнение аккумулятора с константой и переход, если не равно	CJNE A, #d, rel	3	2	(PC) ← (PC)+3, если (A) ≠ #d, то (PC) ← (PC)+ +rel, если (A) < #d, то (C) ← 1, иначе (C) ← 0
Сравнение регистра с константой и переход, если не равно	CJNE Rn, #d, rel	3	2	(PC) ← (PC)+3, если (Rn) ≠ #d, то (PC) ← (PC)+ + rel, если (Rn) < #d, то (C) ← 1, иначе (C) ← 0
Сравнение байта в РПД с константой и переход, если не равно	CJNE @Ri,#d, rel	3	2	(PC) ← (PC)+3, если ((Ri)) ≠ #d, то (PC) ← (PC)+ + rel, если ((Ri)) < #d, то (C) ← 1, иначе (C) ← 0
Длинный вызов подпрограммы	LCALL adl6	3	2	(PC) ← (PC)+3, (SP) ← (SP)+1, ((SP)) ← ($PC_{0...7}$), (SP) ← (SP)+1, ((SP)) ← (PC8...15), (PC) ← ad16
Абсолютный вызов подпрограммы в пределах страницы в 2 Кб	ACALL ad11	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	2	($PC_{8...15}$) ← ((SP)), (SP) ← (SP)−1, ($PC_{0...7}$) ← ((SP)), (SP) ← (SP)−1
Возврат из подпрограммы обработки прерывания	RETI	1	2	($PC_{8...15}$) ← ((SP)), (SP) ← (SP)−1, ($PC_{0...7}$) ← ((SP)), (SP) ← (SP)−1
Пустая операция	NOP	1	1	(PC) ← (PC)+1

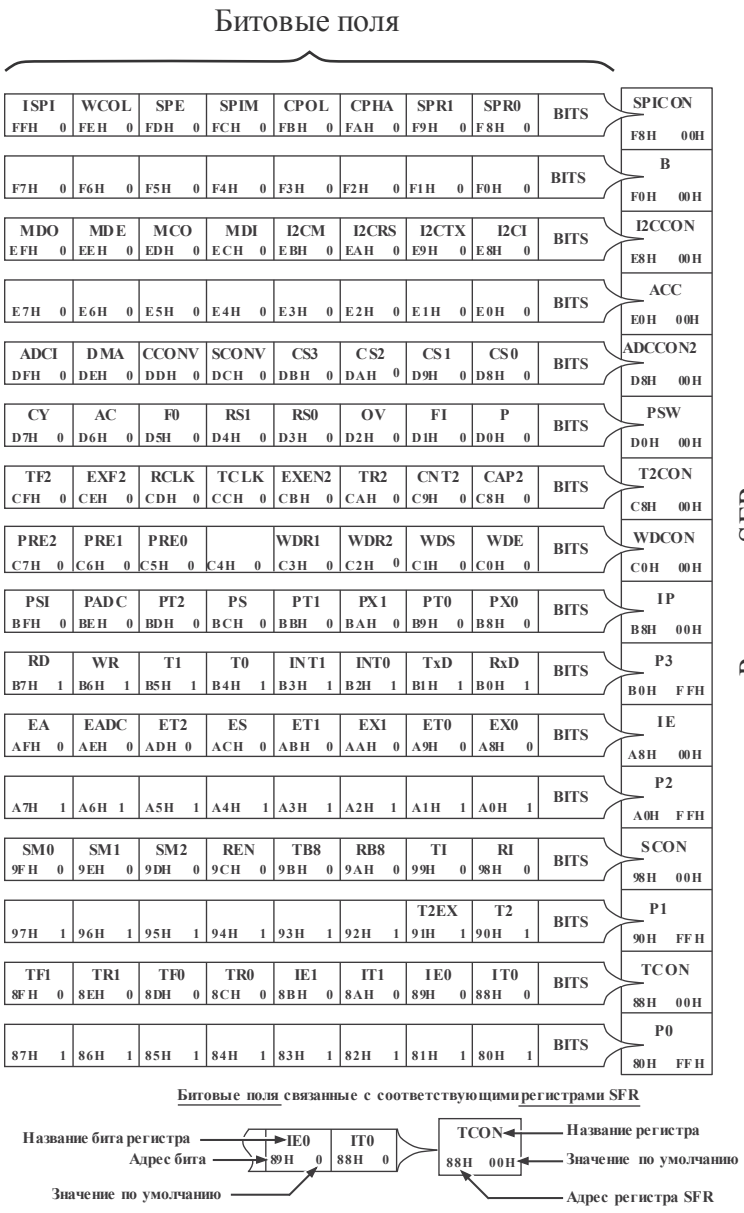
Приложение 2
Адреса и значения по умолчанию регистров специальных функций микроконтроллера ADuC812

Регистры SFR

SPICON[1] F8H 00H	DAC0L F9H 00H	DAC0H FAH 00H	DAC1L FBH 00H	DAC1H FCH 00H	DACCON FDH 04H	RESERVED	NOT USED
B[1] F0H 00H	ADCOFSL[2] F1H 00H	ADCOFSH[2] F2H 20H	ADCGAINL[2] F3H 00H	ADCGAINH[2] F4H 00H	ADCCON3 F5H 00H	RESERVED	SPIDAT F7H 00H
I2CCON[1] E8H 00H	RESERVED	RESERVED	RESERVED	RESERVED	RESERVED	RESERVED	ADCCON1 EFH 20H
ACC[1] E0H 00H	RESERVED	RESERVED	RESERVED	RESERVED	RESERVED	RESERVED	RESERVED
ADCCON2[1] D8H 00H	ADCDATAL D9H 00H	ADCDATAH DAH 00H	RESERVED	RESERVED	RESERVED	RESERVED	PSMCON DFH DEH
PSW[1] D0H 00H	RESERVED	DMAL D2H 00H	DMAH D3H 00H	DMAP D4H 00H	RESERVED	RESERVED	RESERVED
T2CON[1] C8H 00H	RESERVED	RCAP2L CAH 00H	RCAP2H CBH 00H	TL2 CCH 00H	TH2 CDH 00H	RESERVED	RESERVED
WDCON[1] C0H 00H	NOT USED	NOT USED	NOT USED	ETIM3 C4H C9H	RESERVED	EDARL C6H 00H	RESERVED
IP[1] B8H 00H	ECON B9H 00H	ETIM1 BAH 52H	ETIM2 BBH 04H	EDATA1 BCH 00H	EDATA2 BDH 00H	EDATA3 BEH 00H	EDATA4 BFH 00H
P3[1] B0H FFH	NOT USED	NOT USED	NOT USED	NOT USED	NOT USED	NOT USED	NOT USED
IE[1] A8H 00H	IE2 A9H 00H	NOT USED	NOT USED	NOT USED	NOT USED	NOT USED	NOT USED
P2[1] A0H FFH	NOT USED	NOT USED	NOT USED	NOT USED	NOT USED	NOT USED	NOT USED
SCON[1] 98H 00H	SBUF 99H 00H	I2CDAT 9AH 00H	I2CADD 9BH 55H	NOT USED	NOT USED	NOT USED	NOT USED
P1[1,3] 90H FFH	NOT USED	NOT USED	NOT USED	NOT USED	NOT USED	NOT USED	NOT USED
TCON[1] 88H 00H	TMOD 89H 00H	TL0 8AH 00H	TL1 8BH 00H	TH0 8CH 00H	TH1 8DH 00H	NOT USED	NOT USED
P0[1] 80H FFH	SP 81H 07H	DPL 82H 00H	DPH 83H 00H	DPP 84H 00H	RESERVED	RESERVED	PCON 87H 00H



[1] Регистры SFR у которых адрес заканчивается на 0H или 8H являются побитно адресуемыми.
[2] Калибровочные коэффициенты устанавливаются при включении питания значениями подобранными на заводе.
[3] Первичная функция Порта P1 - аналоговый ввод, следовательно для включения цифровой дополнительной функции необходимо в соответствующие разряды порта записать 0


Битовые поля
ISPI FFH 0 | WCOL FEH 0 | SPE FDH 0 | SPIM FCH 0 | CPOL FBH 0 | CPHA FAH 0 | SPR1 F9H 0 | SPR0 F8H 0 | BITS | SPICON F8H 00H
F7H 0 | F6H 0 | F5H 0 | F4H 0 | F3H 0 | F2H 0 | F1H 0 | F0H 0 | BITS | B F0H 00H
MDO EFH 0 | MDE EEH 0 | MCO EDH 0 | MDI ECH 0 | I2CM EBH 0 | I2CRS EAH 0 | I2CTX E9H 0 | I2CI E8H 0 | BITS | I2CCON E8H 00H
E7H 0 | E6H 0 | E5H 0 | E4H 0 | E3H 0 | E2H 0 | E1H 0 | E0H 0 | BITS | ACC E0H 00H
ADCI DFH 0 | DMA DEH 0 | CCONV DDH 0 | SCONV DCH 0 | CS3 DBH 0 | CS2 DAH 0 | CS1 D9H 0 | CS0 D8H 0 | BITS | ADCCON2 D8H 00H
CY D7H 0 | AC D6H 0 | F0 D5H 0 | RS1 D4H 0 | RS0 D3H 0 | OV D2H 0 | FI D1H 0 | P D0H 0 | BITS | PSW D0H 00H
TF2 CFH 0 | EXF2 CEH 0 | RCLK CDH 0 | TCLK CCH 0 | EXEN2 CBH 0 | TR2 CAH 0 | CNT2 C9H 0 | CAP2 C8H 0 | BITS | T2CON C8H 00H
PRE2 C7H 0 | PRE1 C6H 0 | PRE0 C5H 0 | C4H 0 | WDR1 C3H 0 | WDR2 C2H 0 | WDS C1H 0 | WDE C0H 0 | BITS | WDCON C0H 00H
PSI BFH 0 | PADC BEH 0 | PT2 BDH 0 | PS BCH 0 | PT1 BBH 0 | PX1 BAH 0 | PT0 B9H 0 | PX0 B8H 0 | BITS | IP B8H 00H
RD B7H 1 | WR B6H 1 | T1 B5H 1 | T0 B4H 1 | INT1 B3H 1 | INT0 B2H 1 | TxD B1H 1 | RxD B0H 1 | BITS | P3 B0H FFH
EA AFH 0 | EADC AEH 0 | ET2 ADH 0 | ES ACH 0 | ET1 ABH 0 | EX1 AAH 0 | ET0 A9H 0 | EX0 A8H 0 | BITS | IE A8H 00H
A7H 1 | A6H 1 | A5H 1 | A4H 1 | A3H 1 | A2H 1 | A1H 1 | A0H 1 | BITS | P2 A0H FFH
SM0 9FH 0 | SM1 9EH 0 | SM2 9DH 0 | REN 9CH 0 | TB8 9BH 0 | RB8 9AH 0 | TI 99H 0 | RI 98H 0 | BITS | SCON 98H 00H
97H 1 | 96H 1 | 95H 1 | 94H 1 | 93H 1 | 92H 1 | T2EX 91H 1 | T2 90H 1 | BITS | P1 90H FFH
TF1 8FH 0 | TR1 8EH 0 | TF0 8DH 0 | TR0 8CH 0 | IE1 8BH 0 | IT1 8AH 0 | IE0 89H 0 | IT0 88H 0 | BITS | TCON 88H 00H
87H 1 | 86H 1 | 85H 1 | 84H 1 | 83H 1 | 82H 1 | 81H 1 | 80H 1 | BITS | P0 80H FFH
Регистры SFR
Битовые поля связанные с соответствующими регистрами SFR
Название бита регистра
Адрес бита
Значение по умолчанию
IE0 89H 0
IT0 88H 0
TCON 88H 00H
Название регистра
Значение по умолчанию
Адрес регистра SFR

Приложение 3
Пример заполнения титульного листа отчета о лабораторной работе

Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Уральский федеральный университет имени первого Президента России Б. Н. Ельцина»
Физико-технологический институт
Кафедра физических методов и приборов контроля качества

Отчет о лабораторной работе № 1

Модель микроконтроллера семейства MCS-51

Преподаватель
_____ (Ф. И.О.)

Студент
группы _____ (Ф. И.О.)

Екатеринбург
2017

Приложение 4
Образы символов, хранящихся в ПЗУ ЖКИ

Upper 4 bit / Lower 4 bit	LLLL	LLLH	LLHL	LLHH	LHLL	LHLH	LHHL	LHHH	HLLL	HLLH	HLHL	HLHH	HHLL	HHLH	HHHL	HHHH
LLLL	CG RAM (1)															
LLLH	CG RAM (2)															
LLHL	CG RAM (3)															
LLHH	CG RAM (4)															
LHLL	CG RAM (5)															
LHLH	CG RAM (6)															
LHHL	CG RAM (7)															
LHHH	CG RAM (8)															
HLLL	CG RAM (1)															
HLLH	CG RAM (2)															
HLHL	CG RAM (3)															
HLHH	CG RAM (4)															
HHLL	CG RAM (5)															
HHLH	CG RAM (6)															
HHHL	CG RAM (7)															
HHHH	CG RAM (8)															

Примечание. H — высокий уровень (1), L — низкий уровень (0).

Оглавление

Учебное издание

Моисейкин Евгений Витальевич

Микроконтроллеры семейства MCS-51
Теория и практика

Редактор И. В. Меркурьева
Верстка О. П. Игнатьевой

Подписано в печать 19.09.2017. Формат 70×100/16.
Бумага офсетная. Цифровая печать. Усл. печ. л. 11,6.
Уч.-изд. л. 6,5. Тираж 50 экз.
Заказ 266

Издательство Уральского университета
Редакционно-издательский отдел ИПЦ УрФУ
620049, Екатеринбург, ул. С. Ковалевской, 5
Тел.: +7 (343) 375-48-25, 375-46-85, 374-19-41
E-mail: rio@urfu.ru

Отпечатано в Издательско-полиграфическом центре УрФУ
620083, Екатеринбург, ул. Тургенева, 4
Тел.: +7 (343) 358-93-06, 350-58-20, 350-90-13
Факс: +7 (343) 358-93-06
http://print.urfu.ru


ДИАЕР УНИВЕРСИТЕТСКОГО КНИГОИЗДАНИЯ
УНИВЕРСИТЕТСКАЯ КНИГА
2015


9 785799 621674