МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное
образовательное учреждение высшего образования
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

МИКРОКОНТРОЛЛЕРНЫЕ СИСТЕМЫ

Лабораторный практикум

Санкт-Петербург
2024

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Целью выполнения лабораторных работ является закрепление теоретического материала по курсу «Микроконтроллерные системы» и получение практических навыков программирования микроконтроллеров.

По результатам выполнения работы студентом оформляется отчет.

Отчет должен содержать титульный лист, соответствующий образцу, размещенному на сайте университета.

Текст отчета в соответствии с требованиями ГОСТ 7.32 – 2017 набирается шрифтом Times New Roman кеглем не менее 12, строчным, без выделения, с выравниванием по ширине; абзацный отступ должен быть одинаковым и равным по всему тексту 1,25 см; строки разделяются полуторным интервалом; поля страницы: верхнее и нижнее – 20 мм, левое – 30 мм, правое – 15 мм; полужирный шрифт применяется только для заголовков разделов и подразделов; разрешается использовать компьютерные возможности акцентирования внимания, применяя шрифты разной гарнитуры.

Основную часть отчета следует делить на разделы и подразделы в соответствии с пунктом «Содержание отчета». Разделы и подразделы должны иметь порядковую нумерацию в пределах всего текста. Номер подраздела включает номер раздела и порядковый номер подраздела, разделенные точкой, после номера раздела и подраздела в тексте точку не ставят. Разделы и подразделы должны иметь заголовки; заголовки разделов и подразделов следует печатать с абзацного отступа с прописной буквы, полужирным шрифтом, без точки в конце, не подчеркивая; если заголовок состоит из двух предложений, их разделяют точкой; переносы слов в заголовках не допускаются. Каждый раздел основной части отчета начинают с новой страницы.

Страницы отчета следует нумеровать арабскими цифрами, соблюдая сквозную нумерацию по всему тексту; титульный лист включают в общую нумерацию страниц; номер страницы на титульном листе не проставляют; номер страницы проставляют в центре нижней части листа без точки.

На все рисунки должны быть ссылки в тексте (например, …в соответствии с рисунком 1); рисунки следует нумеровать арабскими цифрами сквозной нумерацией; рисунки могут иметь наименование и пояснительные данные (подрисуночный текст).

На все таблицы должны быть ссылки в тексте; таблицы, следует нумеровать арабскими цифрами сквозной нумерацией; наименование таблицы следует помещать над таблицей слева, без абзацного отступа.

Полностью оформленный отчет допускается к защите. Для успешной защиты отчета обучающемуся необходимо уметь аргументированно обосновывать приведенные в отчете утверждения, положения и выводы.

Лабораторная работа 1
АРХИТЕКТУРА ЯДРА И СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРОВ MCS-51

Цель работы: изучение архитектуры ядра и системы команд микроконтроллеров семейства MCS-51; приобретение навыков программирования микроконтроллеров.

1. Основные сведения о микроконтроллерах MCS-51

MCS-51 (или 8051) – это семейство 8-разрядных микроконтроллеров (МК), разработанное компанией Intel. В настоящее время микроконтроллеры этого семейства выпускают различные производители. Одним из крупнейших является компания Microchip Technology Inc., выпускающая микроконтроллеры серии AT89 (до 2016 года микроконтроллеры этой серии AT89 производились компанией Atmel Corporation) [1-3].

1.1. Основные характеристики микроконтроллеров MCS-51

Большинство микроконтроллеров семейства MCS-51 обладает следующими основными характеристиками:

- 8-разрядный процессор,
- внутренняя (встроенная) память программ (RAM), объем которой зависит от модели МК,
- внутренняя память данных (RAM) объемом 265 байт,
- возможность подключения внешней памяти данных и программ до 64 Кбайт каждая,
- глубина стека 256 байт,
- 32 программируемых вывода (4 8-разрядных порта ввода-вывода),
- полнодуплексный последовательный порт,
- три 16-разрядных таймера/счетчика,
- 8 источников прерываний,
- битовый процессор (для работы с битами информации),

– 256 прямоадресуемых бит,

– есть операции умножения и деления,

– стандартная тактовая частота – 12 МГц.

Приведенные особенности характерны для большинства современных микроконтроллеров семейства MCS-51, однако следует помнить, что в зависимости от конкретной модели может изменяться объем памяти программ и данных, число таймеров и источников прерываний, а также число внешних выводов.

Большинство МК серии MCS-51 имеет 40 внешних выводов (рис. 1).

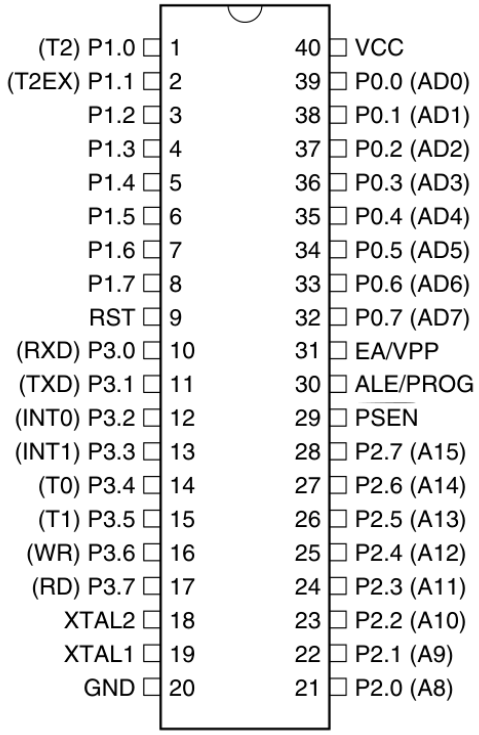

Рис. 1. Внешние выводы 40-выводных микроконтроллеров семейства MCS-51

Функциональное назначение выводов:

V_{CC} – напряжение питания;

GND – земля (0 В);

RST – вход сброса; подача напряжения высокого уровня на этот вывод МК в течение двух машинных циклов приводит к перезагрузке МК;

XTAIL1, XTAIL2 – выводы для подключения внешнего тактового генератора;

ALE/PROG, EA/VPP, PSEN, WR, RD – выводы для передачи управляющих сигналов при использовании внешней памяти программ и данных (подробнее о подключении внешней памяти см. в разделе 1.3), а также для записи в память программ;

INT0, INT1 – входы внешних прерываний;

T0, T1, T2, T2EX – выводы для подачи внешних сигналов таймерам-счетчикам 0, 1 и 2;

RxD, TxD – приемник и передатчик последовательного порта;

P1.0…P1.7, P3.0…P3.7 – выводы двунаправленных 8-битных портов 1 и 3, имеют альтернативные функции, связанные с периферийными устройствами;

P0.0…P0.7, P2.0…P2.7 – выводы двунаправленных 8-битных портов 0 и 2, выполняют альтернативные функции при подключении внешней памяти программ и данных.

Большинство микроконтроллеров имеет гарвардскую архитектуру, то есть отдельные блоки для памяти программ и памяти данных, а соответственно и отдельные шины адреса и данных для связи с каждым блоком. Обобщенная структурная схема МК представлена на рис. 2.

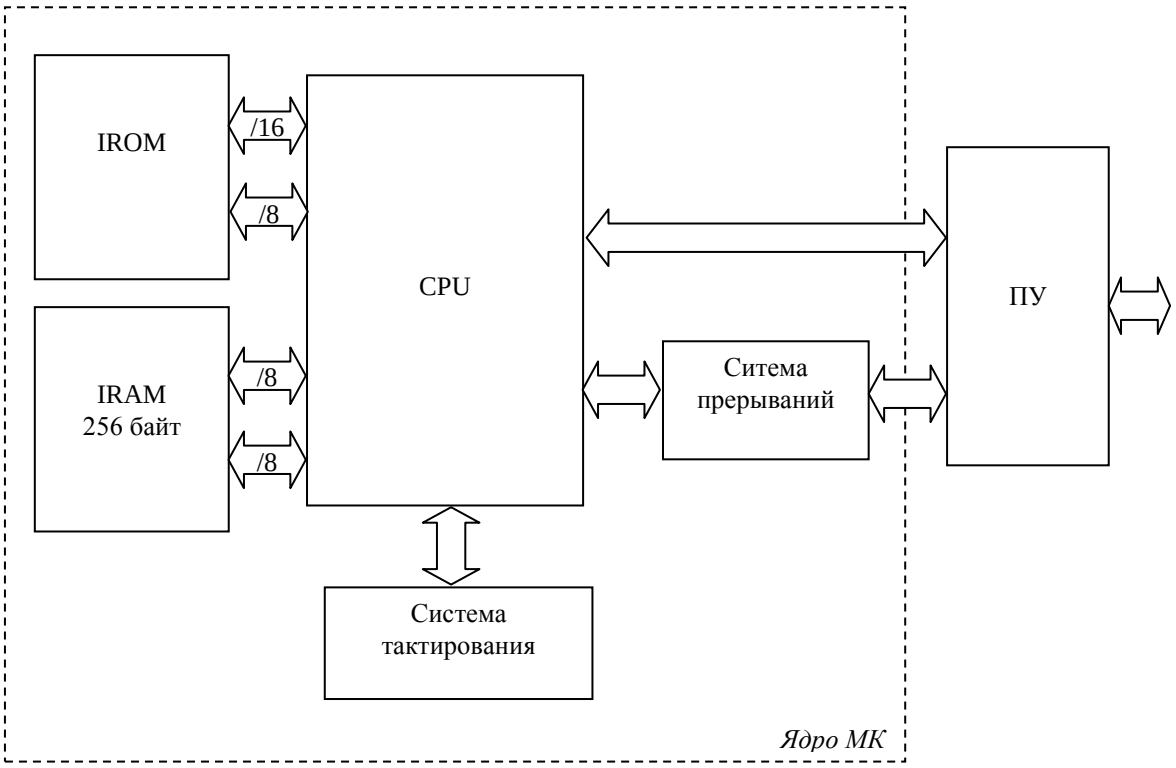

Рис. 2. Обобщенная структурная схема МК

За связь микроконтроллера с внешними устройствами отвечают периферийные устройства (ПУ): порты, таймеры и др. Подробные сведения о периферийных устройствах МК приведены в описании работ, посвященных использованию этих устройств.

В состав ядра МК входит центральный процессор (CPU), внутренняя память программ, внутренняя память данных, система тактирования, а также система прерываний.

1.2. Архитектура центрального процессора

Центральный процессор большинства микроконтроллеров состоит из дешифратора команд, блока арифметико-логического устройства (АЛУ) и блока управления программой.

Дешифратор команд производит декодирование команды и формирование управляющих сигналов, необходимых для ее выполнения.

Блок АЛУ производит большую часть операций над данными и состоит из арифметико-логического устройства, аккумулятора A, регистров B и PSW (подробнее см. раздел 1.3). АЛУ производит операции как над 8-битными данными, так и над отдельными битами.

Блок управления программой управляет последовательностью, в которой выполняются команды, расположенные в памяти программ. 16-битный программный счетчик PC (*Program counter*) содержит адрес следующей для выполнения команды.

Длительность машинного цикла в микроконтроллерах семейства MCS-51 составляет 12 тактовых импульсов. Все команды выполняются в течение одного или двух машинных циклов за исключением команд умножения и деления, выполнение которых занимает 4 машинных цикла (подробнее см. раздел 1.4).

1.3. Организация памяти

Так как микроконтроллеры семейства MCS-51 реализованы в соответствии с гарвардской архитектурой, они имеют раздельные (физически и логически)

адресные пространства памяти – память программ и память данных. Полное пространство памяти МК приведено на рис. 3.

1.3.1. Память программ

Память программ предназначена для ранения кода исполняемой микроконтроллером программы. Эта память, как правило, является энергонезависимой (информация в ней сохраняется при выключении питания) и доступной только для чтения.

Объем внутренней памяти программ (IROM) семейства MCS-51 зависит от конкретной модели и составляет от 0 до 64 Кбайт. Для работы моделей, не имеющих внутренней памяти программ, необходимо подключение внешней памяти программ.

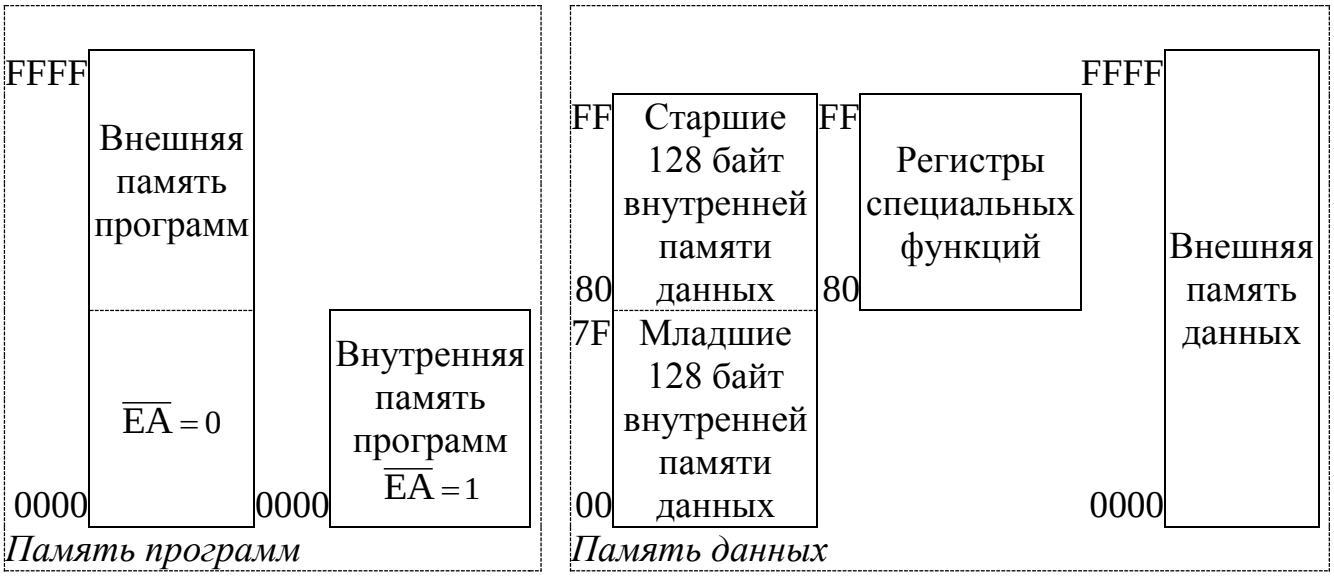

Рис. 3. Полное пространство памяти микроконтроллеров семейства MCS-51

У любых МК семейства MCS-51 имеется возможность подключения внешней памяти программ объемом до 64 Кбайт.

Если на выводе $\overline{EA}$ микроконтроллера высокий уровень напряжения и значение программного счетчика PC не превышает максимальный адрес внутренней памяти программ, то МК обращается к внутренней памяти.

Если на выводе $\overline{EA}$ микроконтроллера низкий уровень напряжения или значение PC превышает максимальный адрес внутренней памяти программ, то МК обращается к внешней памяти программ.

При включении или сбросе МК, содержимое PC обнуляется и программа начинает выполняться с адреса 0h. Этот адрес называется *вектором сброса* МК.

Блок памяти программ с адресами 03h – 2Bh обычно содержит *векторы прерываний*. Нижняя граница блока векторов прерываний может отличаться для разных моделей МК, подробнее см. в описании работы, посвященной системе прерываний.

1.3.2. Память данных

Большинство МК семейства MCS-51 имеют внутреннюю память данных объемом 256 байт. Кроме этого, также имеется возможность подключения внешней памяти данных объемом до 64 Кбайт.

На кристалле МК размещаются три физически разделенных блока внутренней памяти данных: блок младших 128 байт IRAM с адресами 00 – 7Fh, блок старших 128 байт IRAM с адресами 80h – FFh и блок регистров специальных функций SFR, размещенный по адресам 80h – FFh.

Так как старшие 128 байт внутренней памяти данных и область SFR имеют одинаковое адресное пространство, для обращения к ним используют разные способы адресации (табл. 1, подробнее см. раздел 1.4).

Таблица 1

Название области	**Адреса**	**Способ адресации**
Младшие 128 байт IRAM	00 – 7Fh	прямая и косвенная
Старшие 128 байт IRAM	80h – FFh	косвенная
Регистры специальных функций	80h – FFh	прямая

Регистры общего назначения

Младшие 128 байт IRAM в отличие от других областей внутренней памяти данных в свою очередь подразделяются на три области (рис. 4).

Первые (младшие) 32 байта IRAM (с адреса 00 по 1Fh) сгруппированы в четыре банка по 8 регистров общего назначения (РОН) с именами R0 – R7 в каждом. Выбор рабочего (текущего) банка регистров осуществляется программно с помощью управляемых бит RS0 и RS1 слова состояния PSW (табл. 3). Отличительной особенностью блока рабочих регистров является возможность

использования, наряду с прямой и косвенной адресацией, прямой регистровой адресации. При старте МК рабочим является банк 0.

Следующие 16 байт IRAM (с адреса 20h по 2Fh) представляют собой область 128 прямо адресуемых бит (может быть использована для хранения 128 программно-управляемых пользовательских флагов). Обращение к каждому из указанных флагов может быть произведено двумя способами: указанием прямого адреса бита (например, 7Ch) или указанием номера бита в байте (например, 2Fh.4). Каждый из 16 байт этой области допускает прямую и косвенную побайтовую адресацию.

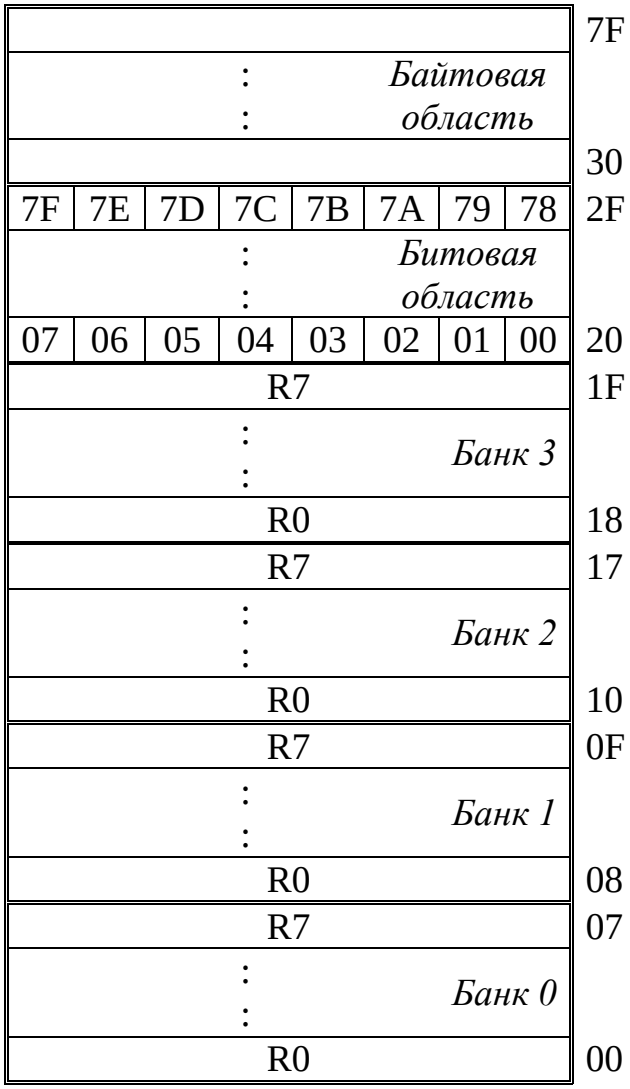

Рис. 4. Структура младших 128 байт IRAM

Оставшиеся 80 байт из области младших 128 байт IRAM и вся область старших 128 байт IRAM адресуется побайтно.

Регистры специального назначения

Блок регистров специального назначения (*Special function register*, SFR) содержит различное число регистров (в зависимости от модели МК). В качестве примера в табл. 2 приведены регистры SFR микроконтроллера AT89C52. Большинство регистров, приведенных в табл. 2, есть у любого МК семейства MCS-51.

Регистры с адресами, кратными восьми, допускают побитную адресацию (отмечены * в табл. 2).

Таблица 2

Адрес	**Имя**	**Значение после старта**	**Назначение**
* 80	P0	11111111	Регистр порта 0
81	SP	00000111	Указатель стека
82	DPL	00000000	Младший байт DPTR
83	DPH	00000000	Старший байт DPTR
87	PCON	0xxx0000	Управление питанием
* 88	TCON	00000000	Управление таймерами 0 и 1
89	TMOD	00000000	Режимы таймеров 0 и 1
8A	TL0	00000000	Младший байт таймера 0
8B	TL1	00000000	Младший байт таймера 1
8C	TH0	0000000	Старший байт таймера 0
8D	TH1	0000000	Старший байт таймера 1
* 90	P1	11111111	Регистр порта 1
* 98	SCON	00000000	Управление последовательным портом
99	SBUF	xxxxxxxx	Регистр данных последовательного порта
* A0	P2	11111111	Регистр порта 2
* A8	IE	0x000000	Разрешение прерываний
* B0	P3	11111111	Регистр порта 3
* B8	IP	xx000000	Приоритеты прерываний
* C8	T2CON	00000000	Управление таймером 2
C9	T2MOD	xxxxxx00	Режимы таймера 2
CA	RCAP2L	00000000	Младший байт RCAP2
CB	RCAP2H	00000000	Старший байт RCAP2
CC	TL2	00000000	Младший байт таймера 2
CD	TH2	00000000	Старший байт таймера 2
* D0	PSW	00000000	Регистр слова-состояния
* E0	ACC	00000000	Аккумулятор A
* F0	B	00000000	Регистр B

Блок SFR содержит две группы регистров: регистры ядра МК и регистры периферийных устройств.

Все *регистры ядра МК*, за исключением программного счетчика PC и четырех банков регистров общего назначения, представлены в области SFR. Это регистры A (ACC), B и PSW блока АЛУ, а также регистры-указатели SP, DPTR.

Аккумулятор ACC предназначен для хранения операндов при выполнении арифметических и логических операций. Допускается обозначение аккумулятора как A.

Регистр B используется в качестве регистра-расширителя аккумулятора при выполнении операций умножения и деления. При выполнении других операций регистр B может использоваться как регистр общего назначения.

Информация о состоянии выполняемой программы содержится в регистре PSW (табл. 3).

Таблица 3

Номер бита	**Имя**	**Назначение**
7	C	Перенос
6	AC	Межтетрадный перенос
5	F0	Пользовательский флаг
4	RS1	Номер рабочего банка РОН: 00 – банк 0 01 – банк 1 10 – банк 2 11 – банк 3
3	RS0	
2	OV	Переполнение
1	-	Не используется
0	P	Четный паритет

Флаг C устанавливается, если операция приводит к переносу из старшего бита результата (переполнение разрядов) или заему в старший бит результата (отрицательный результат). При отсутствии переноса и заема флаг C сбрасывается.

Флаг AC устанавливается, если операция приводит к переносу из младшего полубайта в старший (при сложении) или заему в младший полубайт из старшего

(при вычитании). При отсутствии межтетрадного переноса и заема флаг AC сбрасывается.

Флаг OV устанавливается, если операция приводит к переносу в старший бит результата, но не к переносу из старшего бита, или наоборот (при отсутствии переноса в старший бит, но при наличии переноса из старшего бита). В противном случае OV сбрасывается. Таким образом, флаг OV фиксирует переполнение для чисел со знаком. Отрицательные числа представляются в МК дополнительном коде, поэтому в знаковом представлении числа с нулевым старшим битом (от 00h до 7Fh) считаются положительными, а числа с единичным старшим битом (от 80h до FFh) – отрицательными. Флаг OV устанавливается, например, если результат сложения двух положительных чисел превышает 7Fh, или если результат вычитания из отрицательного числа получился меньше 80h.

Флаг P обновляется в каждом машинном цикле: устанавливается, если число единиц в аккумуляторе нечетно, и сбрасывается, если число единиц в аккумуляторе четно. Флаг программно недоступен: при записи в PSW флаг P не изменяется.

Флаг F0 предназначен для свободного использования программистом.

Команды, приводящие к изменению флагов C, AC и OV, приведены в разделе 1.4 (табл. 7).

Указатель стека SP (Stack pointer) может адресовать любую область IRAM. Глубина стека не может превышать 256 байт. При загрузке данных в стек сначала осуществляется инкремент SP, а затем выполняется запись в стек. При чтении данные извлекаются из стека, а затем производится декремент SP.

С помощью 16-разрядного указателя данных DPTR (*Data pointer*) обеспечивается косвенная адресация операндов при обращении к внешней памяти данных и программ.

1.3.3. Подключение внешней памяти программ и данных

В микроконтроллерах семейства MCS-51 имеется возможность подключения внешней памяти программ (EROM) и данных (ERAM) объемом до 64 Кбайт каждая. Механизм обращения МК к внешней памяти предусматривает

подключение микросхем памяти к выводам портов P0 и P2 и P3 с использованием внешнего регистра. Схема одновременного подключения 64 Кбайт внешней памяти программ и данных изображена на рис. 5.

При доступе к внешней памяти программ и данных 16-разрядный адрес формируется на выводах порта P0 (младший байт) и порта P2 (старший байт). Выдаваемый через порт P0 младший байт адреса фиксируется во внешнем регистре R по спаду сигнала ALE (Address Latch Enable), после чего линии порта P0 используются как шина данных.

Доступ к *внешней памяти программ* разрешен либо если присутствует низкий уровень напряжения на входе $\overline{EA}$, либо, если содержимое счетчика команд PC превышает значение максимального адреса внутренней памяти. Чтение из внешней памяти программ происходит по спаду сигнала на выводе $\overline{PSEN}$ (Program Store ENable), выполняющего функцию разрешающего сигнала чтения EROM.

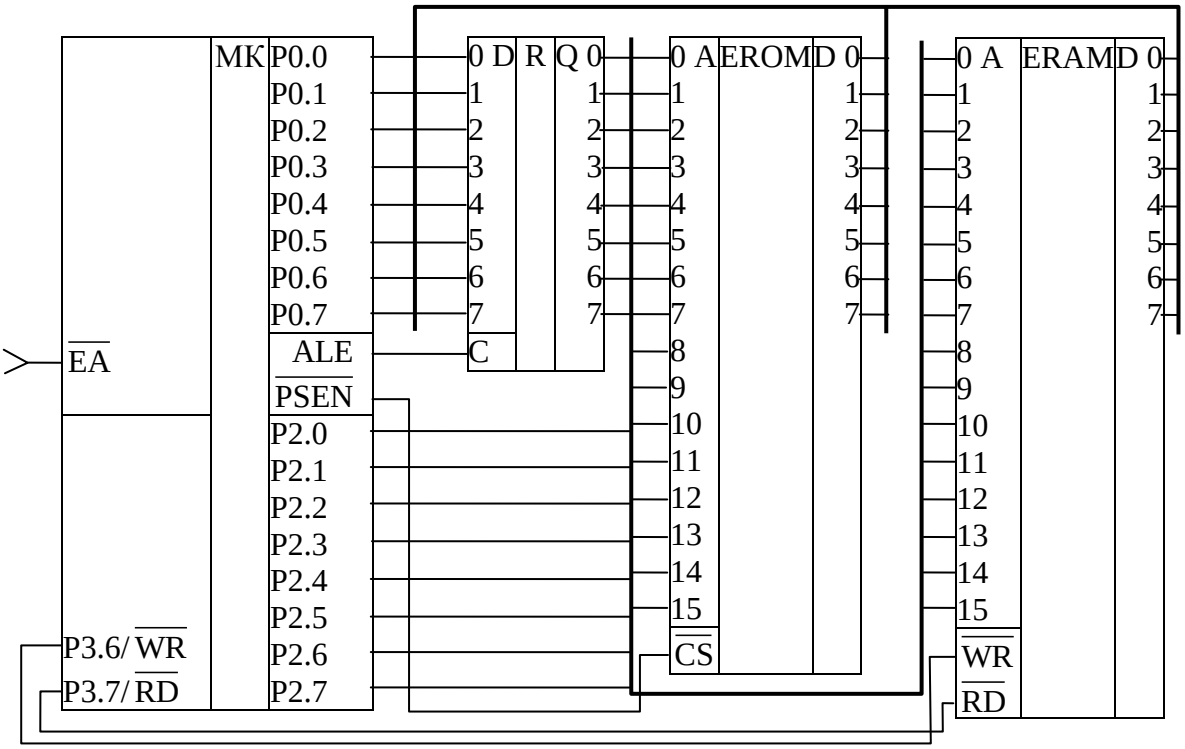

Рис. 5. Подключение внешней памяти программ и данных

Чтение *внешней памяти данных* происходит по спаду сигнала на выводе $\overline{RD}$, а запись – по спаду сигнала на выводе $\overline{WR}$.

1.4. Система команд

Система команд микроконтроллеров семейства MCS-51 состоит из 111 команд, выполняющих 54 операции (табл. 6). Операции производятся над данными четырех типов: битами, тетрадами (4-битными словами), байтами и 16-битными словами, при этом большинство операндов являются байтами.

Машинные коды команд представляются одним, двумя или тремя байтами. Команды выполняются за один или два машинных цикла. Исключение составляют команды умножения и деления, длительность исполнения которых равна 4 машинным циклам.

1.4.1. Способы адресации

Для доступа к операндам используются пять способов адресации (табл. 4).

Указанные способы адресации обеспечивают обращение к операндам-источникам. При обращении к операндам-приемникам непосредственная и базово-индексная адресации не используются.

Таблица 4

Способ адресации	**Область памяти**	**Примеры**
Регистровая	R0..R7 текущего банка, A, B, DPTR, флаг C	R1 DPTR
Прямая	SFR, младшие 128 байт IRAM, прямо адресуемые биты	4Ah P0
Непосредственная	ROM	#5Ah
Косвенная	IRAM, кроме SFR; ERAM	@R0 @DPTR
Базово-индексная	ROM	@DPTR+A @PC+A

Регистровая адресация применяется для адресации регистров A, B, DPTR, флага C, также восьми регистров R0 – R7 текущего (рабочего) банка регистров. Номер регистра содержится в байте кода операции.

Прямая адресация, т.е. адресация с указанием 8-битного адреса операнда во втором или третьем байте команды, используется при обращении к младшим 128 байтам IRAM, регистрам блока SFR и прямо адресуемым битам. К последним

относятся 128 бит из битовой области IRAM (рис. 4), а также биты регистров SFR, отмеченных * в табл. 2.

Операндом команды с *непосредственной адресацией* является константа, расположенная во втором или втором и третьем байтах команды.

При *косвенной адресации* происходит обращение к ячейке памяти, адрес которой содержится в регистре R0, R1 или DPTR. С использованием регистров R0 и R1 возможно организовать доступ к 256 байтам IRAM, а также к 256 байтам ERAM. Доступ в пределах 64 Кбайт может быть организован с помощью регистра DPTR. Для адресации старших 128 байт IRAM допустима только косвенная адресация. Отметим, что косвенная адресация через регистр SP используется в командах, производящих операции со стеком.

Для чтения памяти программ используется *базово-индексная адресация*, обеспечивающая формирование адреса выбираемого байта путем сложения 8-битного содержимого аккумулятора (исполняющего роль индексного регистра) с 16-биным содержимым DPTR или PC (исполняющего роль базового регистра). Такой способ адресации удобно применять для обращения к элементам массива данных, расположенного в памяти программ (пример использования приведен в описании работы с дисплеем).

Для обозначения типов данных и способов адресации в мнемониках команд применяются следующие обозначения (табл. 5).

Таблица 5

Обозначение	**Описание**
Rn	регистры R0 – R7 текущего банка РОН
bit	8-битный адрес бита
/bit	используется инвертированное значение бита, сам бит в IRAM не инвертируется
#data	8-битная константа
#data16	16-битная константа
direct	8-битный адрес
@Ri	обозначение косвенной адресации через регистр R0 или R1
@DPTR	обозначение косвенной адресации через регистр DPTR
addr16	16-битный адрес
addr11	11-битный адрес
rel	8-битное значение смещения со знаком

Примечание. В тексте программ вместо указания значений rel, addr11 и addr16 можно использовать метки (см. пример в разделе 1.5).

Множество команд микроконтроллера может быть разделено на 4 группы:

a. команды передачи данных,
b. команды арифметических операций;
c. команды логических операций;
d. команды управления переходами.

1.4.2. Команды передачи данных

Команды этой группы делятся на три класса: команды общего назначения, команды аккумулятора и команды DPTR.

Команды пересылки общего назначения

MOV выполняет пересылку байта или бита из операнда-источника в операнд-приемник.

PUSH инкрементирует (увеличивает на 1) регистр SP, а затем выполняет пересылку байта-источника по адресу, содержащемуся в SP.

POP выполняет пересылку байта, расположенного по адресу, содержащемуся в SP, в операнд-приемник, а затем декрементирует (уменьшает на 1) регистр SP.

Команды пересылки аккумулятора

XCHG осуществляет обмен байта-источника с аккумулятором A.

XCHD осуществляет обмен младшего полубайта (с 0 по 3 биты) байта-источника с младшим полубайтом аккумулятора A.

MOVX выполняет передачу байта между ячейкой внешней памяти данных (ERAM) и аккумулятором. Адрес ячейки задается в регистре DPTR (16-битный адрес) или в регистре R0 или R1 (8-битный адрес).

MOVC выполняет передачу байта из памяти программ (ROM) в аккумулятор. Адрес ячейки задается в регистре DPTR (16-битный адрес) или в регистре R0 или R1 (8-битный адрес). Содержимое A до передачи используется в качестве индекса в 256-байтном массиве, адрес начала которого содержится в базовом регистре (DPTR или PC).

Команды пересылки DPTR

MOV DPTR, #data16 загружает 16-битную константу в регистр DPTR (представлен в области SFR в виде регистровой пары: DPH и DPL).

1.4.3. Команды арифметических операций

Микроконтроллер выполняет четыре арифметических операции. Операции выполняются над восьмибитными беззнаковыми числами, однако флаг переполнения (см. раздел 1.3.2) позволяет использовать операции сложения и вычитания как для беззнаковых операндов, так и для операндов со знаком. Арифметические операции также могут напрямую производиться с числами в коде BCD.

Сложение

INC (инкремент) прибавляет единицу к операнду-источнику и помещает результат в операнд.

ADD прибавляет аккумулятор к операнду-источнику и возвращает результат в аккумулятор.

ADDC прибавляет аккумулятор к операнду-источнику, затем прибавляет 1, если флаг C был установлен, и возвращает результат в аккумулятор.

DA используется для коррекции результата после операции сложения двух чисел в коде BCD (двоично-десятичный код – каждый разряд десятичного числа представляется четырьмя битами, например 96_{10} = 1001 0110_{BCD}). Результат коррекции записывается в аккумулятор. Флаг C устанавливается, если результат превышает 99_{10}, иначе C сбрасывается.

Вычитание

SUBB вычитает из аккумулятора операнд-источник, затем вычитает 1, если флаг C был установлен, и возвращает результат в аккумулятор. Операция вычитания производится путем сложения уменьшаемого с дополнительным кодом вычитаемого.

DEC (декремент) вычитает единицу из операнда-источника и помещает результат в операнд.

Умножение

MUL выполняет умножение аккумулятора A на регистр B. Сомножители интерпретируются как восьмибитные беззнаковые числа. Результат умножения занимает два байта: младший байт помещается в аккумулятор, старший – в регистр B. Флаг C сбрасывается. Флаг OV устанавливается, если старший байт результата не нулевой, в противном случае OV сбрасывается.

Деление

DIV выполняет деление аккумулятора A на регистр B. Операнды интерпретируются как восьмибитные беззнаковые числа. Целая часть частного помещается в аккумулятор, остаток – в регистр B. Флаг C сбрасывается. Флаг OV устанавливается в случае деления на ноль, в противном случае OV сбрасывается.

1.4.4. Команды логических операций

Команды этой группы выполняют основные логические операции над байтами и битами.

Команды с одним операндом

CLR обнуляет операнд. Операндом может быть аккумулятор или любой прямо адресуемый бит.

SETB записывает 1 в операнд. Операндом может быть любой прямо адресуемый бит.

CPL инвертирует операнд. Операндом может быть аккумулятор или любой прямо адресуемый бит.

RL, RLC, RR, RRC и SWAP – пять сдвиговых операций, которые могут быть применены только к аккумулятору (рис. 6).

Операции RR и RL осуществляют циклический сдвиг аккумулятора на один бит вправо и влево соответственно. При этом вытесняемый бит помещается в освободившийся разряд аккумулятора.

Операции RRC и RLC осуществляют циклический сдвиг с использованием флага C на один бит вправо и влево соответственно. При этом вытесняемый из аккумулятора бит вытесняет флаг C, а вытесняемое значение флага C помещается в освободившийся разряд аккумулятора.

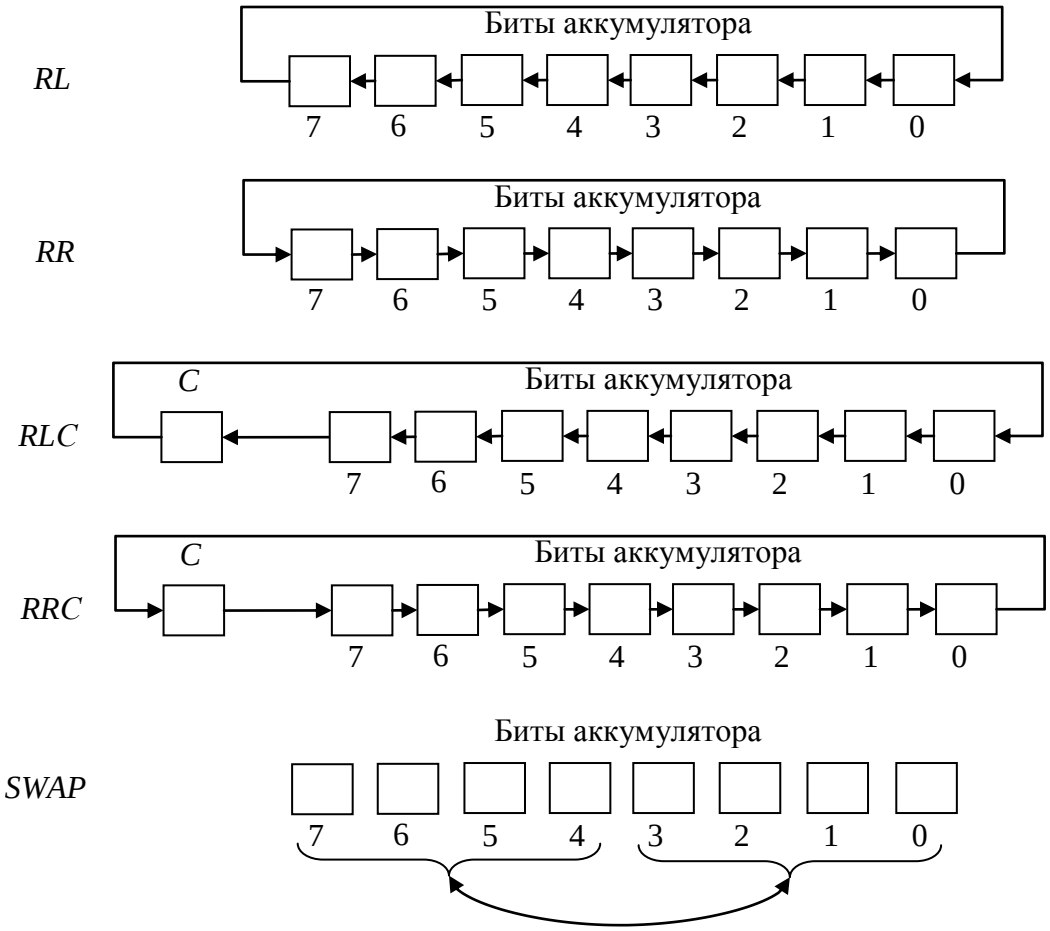

Рис. 6. Операции сдвига

Операция SWAP меняет местами старший (разряды с 4 по 7) и младший (разряды с 0 по 3) полубайты аккумулятора. Результат выполнения этой команды аналогичен результату четырех последовательных операций RL.

Команды с двумя операндами

ANL выполняет побитовое логическое И двух операндов и возвращает результат на место первого операнда. Операндами могут быть байты или биты.

ORL выполняет побитовое логическое ИЛИ двух операндов и возвращает результат на место первого операнда. Операндами могут быть байты или биты.

XRL выполняет побитовое логическое исключающее ИЛИ (сложение по модулю два) двух операндов и возвращает результат на место первого операнда. Операндами могут быть только байты.

1.4.5. Команды управления

К командам этой группы относятся команды безусловных переходов, условных переходов и возврата из прерываний.

Команды безусловных переходов

ACALL – двухбайтная команда вызова подпрограммы, используется, если адрес подпрограммы расположен в пределах текущей 2Кбайтной страницы памяти программ. ACALL помещает в стек двухбайтный адрес следующей за ней в памяти программ команды (т.е. содержимое PC, адрес возврата), а затем заменяет в программном счетчике PC 11 младших бит на 11-битное значение адреса, указанное в операнде. Таким образом, если команда ACALL занимает два последних байта на 2Кбайтной странице, переход будет осуществлен на следующую страницу памяти программ.

LCALL – трехбайтная команда вызова подпрограммы, используется для адресации всего 64Кбайтного адресного пространства памяти программ. LCALL помещает в стек адрес следующей за ней в памяти программ команды (т.е. содержимое PC, адрес возврата), а затем помещает в программный счетчик PC 16-битное значение адреса подпрограммы, указанное в операнде.

RET (возврат из подпрограммы) извлекает из стека два байта (адрес возврата, помещенный в стек предшествующей операцией CALL) и помещает их в PC. При этом значение SP уменьшается на 2.

AJMP – двухбайтная команда безусловного перехода, используется, если адрес перехода расположен в пределах текущей 2Кбайтной страницы памяти программ. AJMP заменяет в программном счетчике PC (содержащем адрес следующей за AJMP команды) 11 младших бит на 11-битное значение адреса, указанное в операнде.

LJMP – трехбайтная команда безусловного перехода, используется для адресации всего 64Кбайтного адресного пространства памяти программ. LJMP помещает в программный счетчик PC 16-битное значение адреса, указанное в операнде.

SJMP – двухбайтная команда короткого безусловного перехода, используется, если адрес перехода расположен на расстоянии от -128 до 127 байт от следующей за SJMP команды. SJMP прибавляет к программному счетчику PC

(содержащему адрес следующей за SJMP команды) значение смещения (интерпретируется как число со знаком), указанное в операнде.

JMP @A+DPTR – однобайтная команда безусловного перехода, используется для адресации всего 64Кбайтного адресного пространства памяти программ. JMP помещает в программный счетчик PC 16-битную сумму аккумулятора (интерпретируется как беззнакове число) и регистра DPTR.

Команды условных переходов

Эта группа команд осуществляет короткий переход только при выполнении определенных условий. Адрес перехода должен быть расположен на расстоянии от -128 до 127 байт от следующей команды. При выполнении соответствующего условия к программному счетчику PC (содержащему адрес следующей команды) прибавляется значение смещения (интерпретируется как число со знаком), указанное в операнде.

JZ осуществляет переход, если аккумулятор равен нулю.

JNZ осуществляет переход, если аккумулятор не равен нулю.

JC осуществляет переход, если флаг C равен единице.

JNC осуществляет переход, если флаг C равен нулю.

JB осуществляет переход, если бит-операнд равен единице.

JNB осуществляет переход, если бит-операнд равен нулю.

JBC осуществляет переход, если бит-операнд равен единице и записывает ноль в этот бит.

CJNE сравнивает первый и второй операнды и выполняет переход, если операнды не равны. Флаг C устанавливается, если первый операнд меньше второго. В противном случае флаг C сбрасывается.

DJNZ уменьшает на 1 (декрементирует) операнд-источник и возвращает результат в операнд. Переход осуществляется, если результат не равен нулю.

Команда возврата из прерывания

RETI извлекает из стека два байта (адрес возврата, помещенный в стек при переходе по вектору прерывания) и помещает их в PC. При этом значение SP уменьшается на 2. Разрешает прерывания с текущим уровнем приоритета.

Подробно работа системы прерываний описана в материалах к лабораторной работе, посвященной прерываниям.

1.4.6. Список команд

Полный перечень команд микроконтроллера приведен в табл. 6. В столбце Б указано количество байт, занимаемых командой в памяти программ. В столбце Ц приведено количество машинных циклов, необходимых для выполнения команды.

Таблица 6

Команда	**Описание**	**Б**	**Ц**
	Команды передачи данных		
MOV A, Rn	Поместить в аккумулятор из регистра	1	1
MOV A, direct	Поместить в аккумулятор из прямо адресуемой ячейки IRAM	2	1
MOV A, @Ri	Поместить в аккумулятор из косвенно адресуемой ячейки IRAM	1	1
MOV A, #data	Поместить в аккумулятор константу	2	1
MOV Rn, A	Поместить в регистр из аккумулятора	1	1
MOV Rn, direct	Поместить в регистр из прямо адресуемой ячейки IRAM	2	2
MOV Rn, #data	Поместить в регистр константу	2	1
MOV direct, A	Поместить в прямо адресуемую ячейку IRAM из аккумулятора	2	1
MOV direct, Rn	Поместить в прямо адресуемую ячейку IRAM из регистра	2	2
MOV direct, direct	Поместить в прямо адресуемую ячейку IRAM из прямо адресуемой ячейки IRAM	3	2
MOV direct, @Ri	Поместить в прямо адресуемую ячейку IRAM из косвенно адресуемой ячейки IRAM	2	2
MOV direct, #data	Поместить в прямо адресуемую ячейку IRAM константу	3	2
MOV @Ri, A	Поместить в косвенно адресуемую ячейку IRAM из аккумулятора	1	1
MOV @Ri, direct	Поместить в косвенно адресуемую ячейку IRAM из прямо адресуемой ячейки IRAM	2	2
MOV @Ri, #data	Поместить в косвенно адресуемую ячейку IRAM константу	2	1
MOV DPTR, #data16	Поместить в DPTR константу	3	2
MOVC A, @A+DPTR	Поместить в аккумулятор из базово-индексно адресуемой ячейки ROM	1	2
MOVC A, @A+PC		1	2
MOVX A, @Ri	Поместить в аккумулятор из косвенно адресуемой	1	2

Команда	**Описание**	**Б**	**Ц**
MOVX A, @DPTR	ячейки ERAM	1	2
MOVX @Ri, A	Поместить в косвенно адресуемую ячейку ERAM	1	2
MOVX @DPTR, A	из аккумулятора	1	2
PUSH direct	Поместить в стек из прямо адресуемой ячейки IRAM	2	2
POP direct	Поместить в прямо адресуемую ячейку IRAM из стека	2	2
XCH A, Rn	Обмен аккумулятора с регистром	1	1
XCH A, direct	Обмен аккумулятора с прямо адресуемой ячейкой IRAM	2	1
XCH A, @Ri	Обмен аккумулятора с косвенно адресуемой ячейкой IRAM	1	1
XCHD A, @Ri	Обмен младшего полубайта аккумулятора с младшим полубайтом косвенно адресуемой ячейки IRAM	1	1
	Команды арифметических операций		
ADD A, Rn	Прибавить к аккумулятору регистр	1	1
ADD A, direct	Прибавить к аккумулятору прямо адресуемую ячейку IRAM	2	1
ADD A, @Ri	Прибавить к аккумулятору косвенно адресуемую ячейку IRAM	1	1
ADD A, #data	Прибавить к аккумулятору константу	2	1
ADDC A, Rn	Прибавить к аккумулятору регистр и фаг C	1	1
ADDC A, direct	Прибавить к аккумулятору прямо адресуемую ячейку IRAM и фаг C	2	1
ADDC A, @Ri	Прибавить к аккумулятору косвенно адресуемую ячейку IRAM и фаг C	1	1
ADDC A, #data	Прибавить к аккумулятору константу и фаг C	2	1
SUBB A, Rn	Вычесть из аккумулятора регистр и фаг C	1	1
SUBB A, direct	Вычесть из аккумулятора прямо адресуемую ячейку IRAM и фаг C	2	1
SUBB A, @Ri	Вычесть из аккумулятора косвенно адресуемую ячейку IRAM и фаг C	1	1
SUBB A, #data	Вычесть из аккумулятора константу и фаг C	2	1
INC A	Инкремент аккумулятора	1	1
INC Rn	Инкремент регистра	1	1
INC direct	Инкремент прямо адресуемой ячейки IRAM	2	1
INC @Ri	Инкремент косвенно адресуемой ячейки IRAM	1	1
DEC A	Декремент аккумулятора	1	1
DEC Rn	Декремент регистра	1	1
DEC direct	Декремент прямо адресуемой ячейки IRAM	2	1
DEC @Ri	Декремент косвенно адресуемой ячейки IRAM	1	1
INC DPTR	Инкремент DPTR	1	2

Команда	**Описание**	**Б**	**Ц**
MUL AB	Умножить A на B	1	4
DIV AB	Разделить A на B	1	4
DA A	Десятичная коррекция аккумулятора	1	1
	Команды логических операций		
ANL A, Rn	Логически умножить аккумулятор на регистр	1	1
ANL A, direct	Логически умножить аккумулятор на прямо адресуемую ячейку IRAM	2	1
ANL A, @Ri	Логически умножить аккумулятор на косвенно адресуемую ячейку IRAM	1	1
ANL A, #data	Логически умножить аккумулятор на константу	2	1
ANL direct, A	Логически умножить прямо адресуемую ячейку IRAM на аккумулятор	2	1
ANL direct, #data	Логически умножить прямо адресуемую ячейку IRAM на константу	3	2
ORL A, Rn	Логически прибавить к аккумулятору регистр	1	1
ORL A, direct	Логически прибавить к аккумулятору прямо адресуемую ячейку IRAM	2	1
ORL A, @Ri	Логически прибавить к аккумулятору косвенно адресуемую ячейку IRAM	1	1
ORL A, #data	Логически прибавить к аккумулятору константу	2	1
ORL direct, A	Логически прибавить к прямо адресуемой ячейке IRAM аккумулятор	2	1
ORL direct, #data	Логически прибавить к прямо адресуемой ячейке IRAM константу	3	2
XRL A, Rn	Логически прибавить по модулю два к аккумулятору регистр	1	1
XRL A, direct	Логически прибавить по модулю два к аккумулятору прямо адресуемую ячейку IRAM	2	1
XRL A, @Ri	Логически прибавить по модулю два к аккумулятору косвенно адресуемую ячейку IRAM	1	1
XRL A, #data	Логически прибавить по модулю два к аккумулятору константу	2	1
XRL direct, A	Логически прибавить по модулю два к прямо адресуемой ячейке IRAM аккумулятор	2	1
XRL direct, #data	Логически прибавить по модулю два к прямо адресуемой ячейке IRAM константу	3	2
CLR A	Очистить аккумулятор	1	1
CPL A	Инвертировать все биты аккумулятора	1	1
RL A	Циклически сдвинуть аккумулятор влево	1	1
RLC A	Циклически сдвинуть аккумулятор влево через C	1	1
RR A	Циклически сдвинуть аккумулятор вправо	1	1
RRC A	Циклически сдвинуть аккумулятор вправо через C	1	1
SWAP A	Обмен полубайтов аккумулятора	1	1

Команда	Описание	Б	Ц
	Команды битового процессора		
CLR C	Сбросить флаг С	1	1
CLR bit	Сбросить прямоадресуемый бит	2	1
SETB C	Установить С	1	1
SETB bit	Установить прямоадресуемый бит	2	1
CPL C	Инвертировать С	1	1
CPL bit	Инвертировать прямоадресуемый бит	2	1
ANL C, bit	Логически умножить С на прямоадресуемый бит	2	2
ANL C, /bit	Логически умножить С на инверсию прямоадресуемого бита	2	2
ORL C, bit	Логически прибавить к С прямо адресуемый бит	2	2
ORL C, /bit	Логически прибавить к С инверсию прямоадресуемого бита	2	2
MOV C, bit	Поместить в С прямоадресуемый бит	2	1
MOV bit, C	Поместить в прямоадресуемый бит С	2	2
	Команды управления переходами		
ACALL addr11	Вызов подпрограммы в пределах текущей 2Кбайтной страницы	2	2
LCALL addr16	Вызов подпрограммы в пределах 64 Кбайт	3	2
RET	Возврат из подпрограммы	1	2
RETI	Возврат из подпрограммы обработки прерывания	1	2
AJMP addr11	Переход в пределах текущей 2Кбайтной страницы	2	2
LJMP addr16	Переход в пределах 64 Кбайт	3	2
SJMP rel	Переход на rel байт	2	2
JMP @A + DPTR	Переход с использованием косвенной адресации	1	2
JZ rel	Переход, если в аккумуляторе ноль	2	2
JNZ rel	Переход, если в аккумуляторе не ноль	2	2
JC rel	Переход, если С установлен	2	2
JNC rel	Переход, если С сброшен	2	2
JB bit, rel	Переход, если бит установлен	3	2
JNB bit, rel	Переход, если бит сброшен	3	2
JBC bit, rel	Переход, если бит установлен, и сброс этого бита	3	2
CJNE A, direct, rel	Сравнить аккумулятор с прямо адресуемой ячейкой IRAM и перейти, если не равно	3	2
CJNE A, #data, rel	Сравнить аккумулятор с константой и перейти, если не равно	3	2
CJNE Rn, #data, rel	Сравнить регистр с константой и перейти, если не равно	3	2
CJNE @Ri, #data, rel	Сравнить косвенно адресуемую ячейку IRAM с константой и перейти, если не равно	3	2
DJNZ Rn, rel	Декремент регистра и переход, если не ноль	2	2
DJNZ direct, rel	Декремент прямо адресуемой ячейки IRAM и переход, если не ноль	3	2

Команда	Описание	Б	Ц
NOP	Нет операции	1	1

Подробное описание всех команд можно найти в руководстве пользователя [3].

В табл. 7 представлены команды, изменяющие флаги в регистре PSW.

Таблица 7

Команда	Флаг			Команда	Флаг		
	C	OV	AC		C	OV	AC
ADD	+	+	+	SETB C	1		
ADDC	+	+	+	CLR C	0		
SUBB	+	+	+	CPL C	+		
MUL	0	+		ANL C, bit	+		
DIV	0	+		ANL C, /bit	+		
DA	+			ORL C, bit	+		
RRC	+			ORL C, /bit	+		
RLC	+			MOV C, bit	+		
CJNE	+						

Примечания.

1. В табл. 7 указаны только флаги C, OV и AC. Флаг P изменяется после каждого машинного цикла, изменяющего аккумулятор.
2. Любая команда, изменяющая прямо адресуемые ячейки памяти IRAM, при применении к PSW изменяет все флаги регистра PSW, кроме флага P.
3. PSW является побитно-адресуемым регистром блока SFR, поэтому все флаги регистра PSW, кроме флага P, могут быть изменены за счет битовых операций.

1.5. Программирование микроконтроллеров семейства MCS-51

Для проектирования и отладки программ 8-разрядных микроконтроллеров семейства MCS-51 различные производители предоставляют соответствующее программное обеспечение. При этом возможно использование различных компиляторов и оформление файла программы для них может отличаться. Однако большинство широко используемых директив (например `#include`, `code`, `org`, `equ`, `end`) является общим для всех компиляторов.

При выполнении лабораторных работ для проектирования и отладки программ предлагается использовать свободно распространяемую среду разработки *MCU 8051 IDE*.

1.5.1. Структура файла программы и директивы компилятора

При использовании *MCU 8051 IDE* каждая строка файла с текстом программы должна быть оформлена в соответствии с одним из трех шаблонов:

```
[label: ][instruction [operand [, operand [, operand ]]][;comment ]
[label: ]directive [argument ][;comment ]
symbol directive argument [;comment ]
```

Квадратными скобками обозначены необязательные части строки.

При компиляции обрабатывается текст программы, расположенный до директивы `end`. Текст программы, расположенный после директивы `end`, при компиляции игнорируется.

Компилятор не чувствителен к регистру символов (большие и маленькие буквы для компилятора одинаковы).

Файл программы рекомендуется оформлять в соответствии со следующим шаблоном.

```
;*****************************************************************
;                                                                *
;    Filename:                                                   *
;    Date:                                                       *
;    File Version:                                               *
;    Author:                                                     *
;    Company:                                                    *
;    Description:                                                *
;                                                                *
;*****************************************************************
; Variables
;*****************************************************************

; TODO PLACE VARIABLE DEFINITIONS HERE

;*****************************************************************
; Reset Vector
;*****************************************************************

org    0h            ; processor reset vector
    ajmp    start    ; go to beginning of program

;*****************************************************************
; Interrupt Service Routines
;*****************************************************************

; TODO INSERT ISR HERE
```

```
;*****************************************************************
; MAIN PROGRAM
;*****************************************************************

org                ; let linker place main program

START:

    ; Insert Your Program Here

    sjmp $                          ; loop forever

    END
```

Согласно приведенному шаблону файл рекомендуется начинать с комментариев, содержащих краткую информацию о имени файла, времени создания, версии, авторах и назначении файла.

В тексте программы кроме команд на языке ассемблера используются так называемые директивы компилятора – инструкции компилятору. Директивы позволяют указать адрес размещения кода в памяти программ, задать символьные имена переменных, подключить к проекту дополнительные файлы, использовать конструкции языков высокого уровня при написании программ.

Для задания символьных имен констант используют директиву `equ`, например:

```
x   equ 18h
```

Тогда команда логического умножения бита C на бит с адресом 23*h* может быть записана следующим образом.

```
anl c,/x
```

Для того чтобы задать адрес размещения кода в памяти программ, используют директивы `org` и `code`. Для того чтобы программа начала выполняться при запуске МК, необходимо разместить код, начиная с адреса 00*h* (адрес вектора сброса (*Reset Vector*)). При этом, так как блок памяти программ с адресами 03h – 93h содержит векторы прерываний, обычно основной код программы размещают с адреса, большего 93*h*, а по адресу вектора сброса располагают команду перехода, например:

```
org    0h              ; processor reset vector
```

```
    ajmp    start       ; go to beginning of program
…
org    100h
start:
```

Основная программа будет размещена по адресу 100*h*.

Между вектором сброса и основной программой при необходимости по адресам векторов прерываний (*Interrupt Vector*) размещают подпрограммы обслуживания соответствующих прерываний, каждая из которых должна завершаться командой *RETI*.

Для размещения байтов данных в памяти программ использую директиву `db`. Например, после компиляции программы, содержащей строки, приведенные ниже, в памяти программ, начиная с адреса 200h будет записано 8 байт: десятичное число 31, 6 байт, содержащих *ASCII*-коды символов слова и десятичное число 20.

```
org 200h
DB 31,'August',20
```

Обязательной директивой любой программы является директива `end`, размещаемая после последней команды программы. Строки, расположенные после директивы `end`, компилятором не обрабатываются.

1.5.2. Форматы данных

При написании программы можно использовать следующие форматы числовых данных: двоичный, восьмеричный, десятичный, шестнадцатеричный. Кроме того компилятор поддерживает символьные данные в формате *ASCII*.

```
db 'string' ; String
```

Ниже приведены примеры записи в аккумулятор одного и того же числа в различных форматах.

```
mov a,#100111b ; двоичное число
mov a,#47q ; восьмиричное число
mov a,#39d ; десятичное число
mov a,#39 ; десятичное число
mov a,#27h ; шестнадцатеричное число
mov a,#'9' ; Character
```

При использовании шестнадцатеричного формата запись таких данных должна начинаться с цифры (например, `mov a,#0bfh`). Запись `mov a,#bfh` вызовет ошибку при компиляции.

1.5.3. Примеры программирования

Приведем примеры программ на языке ассемблера MCS-51.

Пример 1. Написать программу, размещающую массив FFh…F0h во внутренней памяти данных(IRAM), начиная с адреса 50h.

Для удобства формирования массива для доступа к ячейкам IRAM будем использовать косвенную адресацию через регистр R0.

Программа имеет следующий вид.

```
;*****************************************************************
;                                                                *
;    Filename: ex11.asm                                          *
;    Date: 2020/02/07                                            *
;    File Version: 1                                             *
;    Author: Solov'eva T. N.                                     *
;    Company: SUAI                                               *
;    Description: example 1.1                                    *
;                                                                *
;*****************************************************************
; Reset Vector
;*****************************************************************

org    0h              ; processor reset vector
    ajmp    start      ; go to beginning of program

;*****************************************************************
; MAIN PROGRAM
;*****************************************************************

org    100h

start:
    mov  R0,#050h      ; нач. адрес -> R0
    mov  A,#0FFh       ; нач. значение -> A
m1: mov  @R0,A
    inc  R0
    dec  A
    cjne A,#0EFh,m1

    sjmp $             ; loop forever

    END
```

Пример 2. Требуется написать программу вычисления логического выражения $F = \bar{x}y \oplus \bar{r}$, где *x*, *y*, *r* – биты.

Для решения этой задачи нужно использовать команды битового процессора.

Сначала назначим расположение операндов в памяти. Пусть операнды `x`, `y`, `r` и бит промежуточного результата (обозначим его `buf`) находятся в младших разрядах ячейки с адресом 23h (рис. 4 и 7), а бит значения результата (обозначим его `rez`) – в старшем разряде ячейки с адресом 2Dh. При написании программы вместо адресов операндов удобно использовать символьные имена. Для этого применяется директива компилятора `equ`.

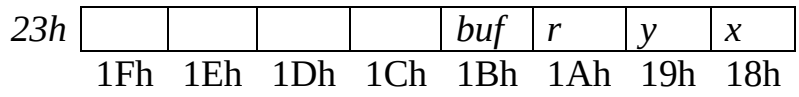

Рис. 7. Расположение операндов программы в памяти

Для указания адреса, начиная с которого в памяти будет располагаться код программы, используют директиву `org`.

Необходимо учесть, что в битовых командах МК нет операции сложения по модулю два, поэтому исходное выражение придется видоизменить: $F = \bar{x}y \oplus \bar{r} = \bar{x}yr \vee \overline{\bar{x}y}\bar{r}$.

Программа имеет следующий вид.

```
;*****************************************************************
;                                                                *
;    Filename: ex12.asm                                          *
;    Date: 2020/02/07                                            *
;    File Version: 0                                             *
;    Author: Solov'eva T. N.                                     *
;    Company: SUAI                                               *
;    Description: example 1.2                                    *
;                                                                *
;*****************************************************************

x   equ 18h
y   equ 19h
r   equ 1ah
buf equ 1bh
rez equ 6fh;

;*****************************************************************
; Reset Vector
;*****************************************************************
```

```
RES_VECT  CODE    0x0000            ; processor reset vector
    SJMP    START                   ; go to beginning of program

;*****************************************************************
; MAIN PROGRAM
;*****************************************************************

MAIN_PROG CODE    0x0100

START:
    mov c,y          ;y -> c
    anl c,/x         ;/x*y -> c
    anl c,r          ;/x*y*r -> c
    mov buf,c        ;/x*y*r -> buf
    mov c,y          ;y -> c
    anl c,/x         ;/x*y -> c
    cpl c            ;/(/x*y) -> c
    anl c,/r         ;/(/x*y)*/r -> c
    orl c,buf        ;F -> c
    mov rez,c        ;F -> rez

    SJMP $                        ; loop forever

    END
```

2. Задание по работе

Необходимо разработать три программы на языке ассемблера MCS-51:

1) программу для вычисления заданного арифметического выражения (для всех операциях полагайте, что операнды и результат – целые однобайтные числа; результат вычислений разместите в ячейке внутренней памяти данных 30h);
2) программу для записи заданного массива чисел *во внешнюю память данных*;
3) программу *на ассемблере битового процессора* для вычисления заданного логического выражения (результат выполнения разместите в любой ячейке памяти данных битового процессора).

Работу программ необходимо проверить с помощью симулятора.

3. Порядок выполнения работы

Выполнение работы состоит из двух частей: разработки программ и проверки корректности их работы с помощью симулятора *MCU 8051 IDE*.

При разработке каждой программы необходимо:

1) ознакомиться с формулировкой задачи;
2) составить схему размещения данных, используемых в программе, в памяти данных;
3) сформировать алгоритм решения задачи;
4) на основании полученного алгоритма составить текст программы на языке ассемблера MCS-51.

Для выполнения работы *на симуляторе MCU 8051 IDE* необходимо выполнить следующие действия.

1. Запустить оболочку симулятора MCU 8051 IDE.
2. Создать новый проект (Project: New). При создании проекта необходимо выбрать любой микроконтроллер и установить ненулевой размер внешней памяти данных (External RAM).
3. В окно редактора вести текст разработанной программы. Начальный адрес программы можно задать с помощью директивы `org`, например, `org 80h` (адрес не должен выходить за пределы области памяти программ Virtual MCU: Show Code memory). Последней командой программы должна быть команда `end`.
4. Сохранить файл кода, добавив его в проект.
5. Выполнить компиляцию программы (Tools: Compile). Проверить наличие ошибок и предупреждений в окне Messages.
6. В случае отсутствия ошибок запустить режим симуляции (Simulator: Start/Shutdown).

7. При выполнении программы 3 в окне побитового просмотра и редактирования памяти данных битового процессора (Virtual MCU: Show bit area) установить желаемые значения исходных данных.

8. Выполнить программу пошагово (Simulator: Step), отслеживая изменения, происходящие в памяти данных и области SFR. Окно внешней памяти данных можно открыть с помощью меню Virtual MCU: Show XDATA memory.

9. Для выхода из режима симуляции нужно воспользоваться меню Simulator: Start/Shutdown.

4. Содержание отчета

Правила оформления отчета приведены в предисловии к практикуму.

Отчет по лабораторной работе должен содержать следующие разделы.

1. Цель работы.
2. Задание по работе.

   Раздел должен включать в себя формулировку задания и данные варианта.
3. Разработка программы 1.

   В данном разделе необходимо привести схему алгоритма (если она использовалась при разработке программы), текст программы с подробными комментариями, а также скриншоты с результатами выполнения программы.
4. Разработка программы 2.

   В данном разделе необходимо привести схему алгоритма (если она использовалась при разработке программы), текст программы с подробными комментариями, а также скриншоты с результатами выполнения программы.
5. Разработка программы 3.

   В данном разделе необходимо привести преобразование логического выражения (если оно производилось), схему алгоритма (если она использовалась при разработке программы), текст программы с подробными комментариями, а также скриншоты с результатами выполнения программы.
6. Вывод.

Пример вывода: «В результате выполнения работы созданы три программы на языке ассемблера MCS-51: программа для вычисления заданного арифметического выражения, программа для записи заданного массива чисел во внешнюю память данных, а также программа на ассемблере битового процессора для вычисления заданного логического выражения. Проверка работоспособности программ произведена в среде *MCU 8051 IDE*. Изучена архитектура и система команд микроконтроллера семейства MCS-51; приобретены навыки программирования микроконтроллера».

5. Контрольные вопросы

1. Какие способы адресации и операции поддерживает битовый процессор MCS-51? Какой флаг играет роль битового аккумулятора?
2. Укажите ячейки памяти, в которых может быть установлен бит с помощью команды SETB.
3. Пусть DPTR = 265Ah, A = 0. Пользуясь рис. 5, опишите состояние выводов МК во время выполнения команд MOVC A, @A+DPTR, MOVX @DPTR, A.
4. Какие команды в приведенной ниже программе относятся к командам управления? Опишите изменения, происходящие в регистрах микроконтроллера, при выполнении этих команд.

```
START:
          mov A,R0
          mov B,#80h
          acall fn1
          add A,#5h
          jmp $

fn1:      mul ab
          jnz m1
          subb A,#60h
m1:       add A,#50h
          ret
```

5. Пусть метка m указывает на ячейку с адресом 080Ah, команда перехода расположена по адресу a) 07A0h; b) 0700h; c) 0800h. Какие из следующих команд перехода можно использовать: AJMP m; LJMP m; SJMP m; JMP m? Почему?

6. Пусть R0 = 0A8h и выполняется одна из команд: a) MOV @R0, A; b) MOV 0A8h, A; c) MOVX @R0, A; в какую ячейку памяти будет записано содержимое аккумулятора в каждом случае?

7. Пусть R0 = 0A5h и выполняется одна из команд: a) MOV A,@R0; b) MOVX A,@R0. К какой области памяти будет происходить обращение в каждом случае?

8. В чем отличие между результатами выполнения команд ANL A, 0Ah и ANL 0Ah, A?

9. Каков будет результат выполнения команд: a) CLR A; b) CLR 0Ah?

10. Каков будет результат выполнения команд: a) CPL A; b) CPL 0Ah?

11. Каков будет результат выполнения команд: a) SETB 7Fh; b) SETB 80h?

12. Пусть A = 0Ah. Рассчитайте в двоичном коде результаты выполнения команд: a) ADD A, #0Ah; b) ORL A, #0Ah; c) XRL A, #0Ah.

13. Опишите флаги регистра PSW. Приведите примеры команд, изменяющих каждый из флагов.

14. Программа осуществляет вычисление логического битового выражения, результат записывается в бит с символическим именем res. Определите местоположение бита результата в памяти, если используется одна из следующих директив компилятора:

a) res: equ 5Fh;
b) res: equ 3Ah;
c) res: equ 62h;
d) res: equ 1Fh;
e) res: equ 6Eh;
f) res: equ 7Fh.

15. Какие способы адресации используются в системе команд микроконтроллеров MCS-51? Напишите программу, использующую все способы.

16. Используя одну команду, запрограммируйте передачу байта из регистра-источника в регистр-приемник.

	Источник		Приемник	
	Регистр	Банк	Регистр	Банк
a	R4	0	R1	2
b	R5	2	R3	1
c	R3	3	R2	0
d	R2	1	R4	3

e	R1	0	R5	1

17. В памяти данных МК имеются три области с одинаковым диапазоном адресов. О каком диапазоне идет речь? Для каждой из областей напишите команды, с помощью которых записывается число A5h в одну из ячеек области.

18. Напишите два способа передачи данных из регистра R1 текущего банка РОН в регистр R2 текущего банка РОН. Сравните объем занимаемой памяти программ и время выполнения предложенных вами способов.

19. Напишите два способа установки флагов F0 и F1 регистра PSW. Сравните объем занимаемой памяти программ и время выполнения предложенных вами способов.

20. Пусть A = 05h. В каком случае после выполнения команды SUBB A,#2h в аккумуляторе окажется число 2? Напишите соответствующую программу.

21. Пусть A = 02h. В каком случае после выполнения команды ADDC A,#2h в аккумуляторе окажется число 5? Напишите соответствующую программу.

22. Для заданного логического выражения напишите программу, записывающую в последовательно расположенные биты значения выражения для всех возможных наборов входных данных.

23. Напишите подпрограмму задержки
    a) на 40 мкс;
    b) на 100 мс.

24. Скорректируйте программу 1 с учетом того, что:
    a) результат умножения может превышать 1 байт;
    b) результат деления может иметь ненулевой остаток.

6. Варианты заданий

X, Y, Z – байты данных, *x, y, r, d – биты* данных.

Номер варианта	Арифметическое выражение	Последовательность элементов массива	Логическое выражение
1	$\left(\frac{X}{Y}-Y\right)X+(2X+Y)$	0h…10h…0h	$\bar{x}y\oplus(r\vee d)$
2	$2X\left(\frac{Z-Y}{X+1}+12h\right)$	F8h…FFh…F0h…F8h	$(x\oplus y)\vee rd$
3	$(Z-Y)+\frac{X}{2}Z$	20h…30h…20h	$(x\oplus y)\vee(r\vee d)$
4	$\left(XZ-\frac{X}{Y}\right)(2Z+Y)$	30h…20h…30h	$(x\vee\bar{y})(r\oplus d)$
5	$\frac{1}{2}\left(\frac{YZ-Y}{X}+\frac{Z}{2}X\right)$	10h…20h…0h…10h	$(\bar{x}\oplus y)(r\vee d)$
6	$\frac{1}{4}(X+Z)\left(Z-\frac{X}{Y}\right)$	10h…0h…10h	$\bar{x}r\oplus y\bar{d}$
7	$0{,}5\left(\frac{X}{Z}-Y\right)(2Y+X)$	0h...10h…0h…10h…0h	$x\bar{r}\oplus\bar{y}d$
8	$X(Y-Z)\left(X+\frac{Y}{Z}\right)$	A7h…B7h…A7h	$x\bar{y}\oplus(r\vee d)$
9	$\frac{1}{2}(X-Z)\left(Y+\frac{X}{Z}\right)$	0h…15h…0h…15h…0h	$(\bar{x}\oplus yr)\vee d$
10	$0,25(Z-2Y)\left(Z+\frac{X}{Y}\right)$	E3h…FFh…F0…E3h	$x\bar{y}\oplus rd$
11	$\left(\frac{X}{Z}-Y\right)X-(2X+Y)$	C4h…D4h…C4h…D5h	$\bar{x}r\oplus yd$
12	$2\left(\frac{XZ-Y}{X+1}+10h\right)$	40h…50h…40h	$x\bar{y}\oplus\bar{r}d$
13	$(Z-Y)X+\frac{X}{2}Z$	50h…60h…50h…60h	$(x\oplus y)rd$
14	$\left(X-\frac{X}{Y}\right)(Z+4Y)$	B9h…A9h…B9h	$x\bar{y}\vee(r\oplus d)$
15	$\frac{1}{4}\left(\frac{Z-Y}{X}+2ZX\right)$	D3h…E3h…D3h…E3h	$\bar{x}d\oplus(y\vee\bar{r})$
16	$X\left(0{,}5(Y+Z)-2\frac{Y}{Z}\right)$	95h…A5h…B5h…95h	$x\bar{d}\oplus(\bar{y}\vee r)$
17	$\frac{1}{2}(X+Y)\left(Z-\frac{X}{Y}\right)$	B3h…BFh…B0h…BFh	$(x\oplus\bar{y}d)\vee r$
18	$\frac{1}{4}(X+2Y)\left(Y-\frac{Y}{X}\right)$	10h…15h…0h…20h	$\left(y\oplus\left(x\vee\bar{d}\right)\right)\vee r$

19	$\frac{XY-Y}{X}(2X+Y)$	10h…0h…20h…10h	$(\bar{x}\oplus r)\vee \bar{y}d$
20	$2X\left((Y-Z)+\frac{Y}{2}Z\right)$	F8h…F0h…FFh…F8h	$\left(x\oplus \bar{d}\right)(y\vee \bar{r})$
21	$\left(\frac{X-Y}{Z+1}+12h\right)Z$	A0h…B0h…90h…B0h	$(\bar{x}\vee y)\oplus r\bar{d}$
22	$Y\left(0{,}5(X+Y)-2\frac{X}{Z}\right)$	C5h…B5h…D5h…C5h	$(x\vee \bar{r})(\bar{y}\oplus d)$
23	$\frac{1}{4}(Y+Z)\left(X-\frac{Y}{Z}\right)$	80h…70h…90h…80h	$(\bar{x}\oplus d)\vee \bar{y}r$
24	$\frac{1}{2}(Y+2X)\left(X-\frac{X}{Y}\right)$	B7h…A7h…C0h	$x\bar{r}\oplus yd$
25	$\left(XY-\frac{X}{Y}\right)(2Y+X)$	A3h…E3h…A3h…E3h	$xd\oplus(y\vee \bar{r})$
26	$2Y\left((X-Z)+\frac{Z}{2}Y\right)$	90h…B0h…C5h…90h	$x\bar{d}\oplus(y\vee r)$
27	$Z\left(0{,}5(Z+X)-2\frac{Z}{X}\right)$	D3h…BFh…D0h…BFh	$(x\oplus \bar{y}d)\vee \bar{r}$
28	$\frac{Y-X}{Z+3}+10h$	0h…15h…10h…20h	$\left(y\oplus\overline{(x\vee \bar{d})}\right)\vee r$
29	$\frac{1}{2}(X+Z)\left(Y-\frac{X}{Z}\right)$	10h…20h…0h…10h	$(\bar{x}\oplus r)\vee \overline{\bar{y}d}$
30	$0{,}5(Z+2X)\left(Z-\frac{X}{Y}\right)$	8h…0h…Fh…8h	$\left(x\oplus \bar{d}\right)\overline{(y\vee \bar{r})}$
31	$\frac{1}{2}\left(\frac{Y-X}{X}Z+\frac{Z}{2}X\right)$	20h…0h…10h	$(\bar{x}\oplus y)\vee(r\vee d)$
32	$\frac{1}{4}\left(Z-\frac{X}{Z}\right)(X+Z)$	10h…20h…10h	$\bar{x}\,\bar{r}\oplus y\bar{d}$
33	$0,25\left(\frac{X}{Z}-Y\right)(2Y+Z)$	0h...20h…0h…10h…0h	$x\bar{r}\vee \bar{y}\oplus d$
34	$(Y+Z)\left(X-\frac{Y}{Z}\right)X$	A7h…D7h…A7h	$x\bar{y}\oplus\left(\bar{r}\vee \bar{d}\right)$
35	$\frac{1}{4}(XZ)-\left(Y+\frac{X}{Z}\right)$	0h…15h…20h…0h	$\overline{(\bar{x}\oplus yr)}\vee d$
36	$0{,}5(Z-2Y)\left(2Z+\frac{X}{Y}\right)$	E3h…FFh…E3h	$x\bar{y}\oplus r\vee d$
37	$\left(\frac{X}{Z}Y\right)X-(2X+Y)$	C0h…D0h…C0h…D0h	$(\bar{x}\vee r)\oplus yd$
38	$2\frac{XZ-Y}{X+1}+16h$	4Ah…50h…4Ah	$\overline{x\bar{y}\oplus \bar{r}d}$
39	$\frac{(Z-1)}{Y}+\frac{X}{2}-Z$	4Fh…60h…4Fh…60h	$(x\oplus y)\vee rd$

40	$\left(X-\frac{X}{Y}\right)(Z+2Y)$	B4h…A4h…B4h	$x\bar{y}\vee\left(r\oplus\bar{d}\right)$
41	$\frac{1}{2}\left(\frac{Z-1}{X}+2ZX\right)$	F3h…E3h…F3h…E3h	$\bar{x}\vee d\oplus\left(y\vee\bar{r}\right)$
42	$X\left((Y+Z)-2\frac{Y}{Z}\right)$	85h…A5h…85h	$\overline{x\bar{d}}\oplus\left(\bar{y}\vee r\right)$
43	$\frac{1}{2}\left(X+Y\right)\left(Z-\frac{X+1}{Y}\right)$	B3h…B0h…BFh	$\left(x\oplus\bar{y}d\right)\vee\bar{r}$
44	$\frac{1}{2}\left(X+4Y\right)\left(Y-\frac{Y}{X}\right)$	10h…0h…20h	$\left(y\oplus\overline{\left(x\vee\bar{d}\right)}\right)\vee r$
45	$\frac{XY-Y}{X}\left(2X+Z\right)$	0h…20h…10h	$\left(\bar{x}\oplus r\right)\vee\overline{\bar{y}d}$
46	$2X-YZ+\frac{Y}{2}Z$	F8h…FFh…F8h	$\overline{\left(x\oplus\bar{d}\right)}\left(y\vee\bar{r}\right)$
47	$\left(\frac{X-Y}{Z+1}+14h\right)Y$	A0h…B0h…90h	$\left(\bar{x}\vee y\right)\oplus\overline{r\bar{d}}$
48	$\left(0,25\left(X+Y\right)-2\frac{X}{Z}\right)Y$	C0h…B0h…D0h…C0h	$\overline{\left(x\vee\bar{r}\right)}\left(\bar{y}\oplus d\right)$
49	$\frac{1}{4}\left(Y+X\right)\left(X-\frac{Y}{Z}\right)$	80h…90h…80h	$\left(\bar{x}\oplus\bar{d}\right)\vee\bar{y}r$
50	$\frac{1}{2}\left(Y+2X\right)\left(X-\frac{X-1}{Y}\right)$	BAh…AAh…CAh	$x\bar{r}\oplus\overline{yd}$

Лабораторная работа 2
ОРГАНИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ МИКРОКОНТРОЛЛЕРА С ПРОСТЕЙШИМИ УСТРОЙСТВАМИ ВЫВОДА

Цель работы: приобретение навыков организации взаимодействия микроконтроллера с простейшими устройствами вывода, знакомство с принципом динамической индикации.

1. Организация взаимодействия микроконтроллера с внешними устройствами

Основное предназначение любого микроконтроллера состоит в управлении внешними устройствами. Внешние устройства подключаются к внешним выводам микроконтроллера. Для обмена информацией между ядром микроконтроллера и внешними устройствами (ВУ) используются простейшие периферийные устройства – порты.

Большинство микроконтроллеров семейства MCS-51 содержат в своем составе четыре двунаправленных восьмиразрядных параллельных портов ввода-вывода P0 – P3, а также двунаправленный последовательный порт.

Рассмотрим устройство параллельных портов ввода-вывода MCS-51.

1.1. Параллельные порты ввода-вывода

Порты P0 – P3 являются двунаправленными 8-разрядными портами. При соответствующем управлении каждый разряд порта может обеспечивать как ввод (прием от ВУ к МК), так и вывод (передачу от МК к ВУ) информации.

Каждый внешний вывод порта связан с триггером. Отдельные триггеры разрядов портов объединены в регистры с именами одноименных портов P0 – P3. Указанные регистры размещаются в блоке SFR, их адреса указаны в табл. 1.

При использовании вывода порта для передачи информации на выводе отображается содержимое триггера (то есть, что запишем в разряд регистра порта, то и будет на выводе). Для организации ввода информации триггер должен быть

установлен в 1 (то есть при настройке вывода порта на ввод, нужно записать в разряд регистра порта единицу)**.**

Таблица 1

Имя порта	**Адрес**
P0 (порт 0)	80h
P1 (порт 1)	90h
P2 (порт 2)	A0h
P3 (порт 3)	B0h

К регистрам P0 – P3 возможно обращение только при использовании прямой адресации, при этом ко всем отдельным разрядам портов допускается побитовое обращение.

Выводы портов микроконтроллеров имеют ограничения по напряжению и току, поэтому для ограничения тока и согласования напряжения при подключении внешних устройств используют дополнительные элементы (резисторы, транзисторы и др.).

Устройство большинства выводов МК таково, что на свободные выводы (не подключенные к источнику высокого или низкого напряжения) подается напряжение питания МК. Исключение составляют выводы порта P0.

Рассмотрим использование портов микроконтроллера для подключения простейших устройств вывода.

1.2. Использование устройств вывода на основе светодиодов

В большинстве встраиваемых систем управления необходимо организовать вывод информации о результатах работы системы. Для этой цели широко применяются небольшие устройства вывода: светодиоды, семисегментые индикаторы и светодиодные матрицы.

1.2.1. Использование семисегментных индикаторов

Семисегментный индикатор представляет собой набор из семи (часто восьми) светодиодов, размещенных определенным образом (рис. 1). Такие индикаторы удобно применять для отображения цифр.

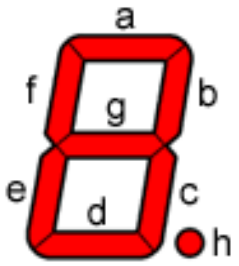

Рис. 1. Расположение светодиодов на семисегментном индикаторе

Светодиоды в индикаторе подключены параллельно друг другу (рис. 2), т. е. объединены аноды (или катоды) светодиодов, и индикатор имеет 8 + 1 управляющий вывод. Различают индикаторы с общим анодом и с общим катодом.

Для включения индикатора с общим катодом необходимо на общий вывод подать низкий уровень напряжения, для индикатора с общим анодом – высокий.

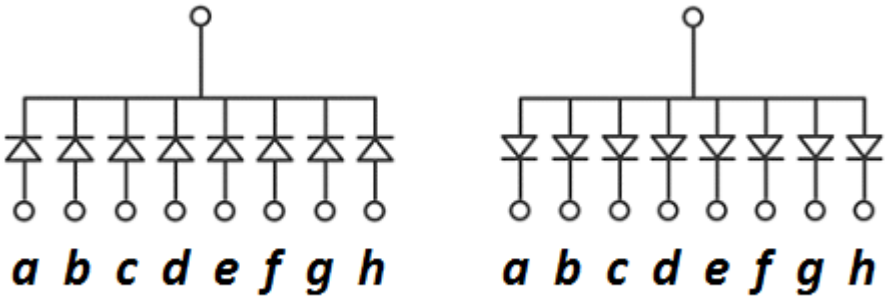

Рис. 2. Индикатор с общим катодом (слева) и с общим анодом (справа)

Соответственно, для засветки сегмента в индикаторах с общим катодом необходимо подать на вывод сегмента высокой уровень напряжения, в индикаторах с общим анодом – низкий.

Например, для отображения на индикаторе с общим катодом цифры 7 необходимо подать 0 на общий вывод индикатора и 1 на сегменты *a*, *b* и *c*.

На рис. 3. изображен пример схемы подключения семисегментного индикатора с общим катодом к микроконтроллеру.

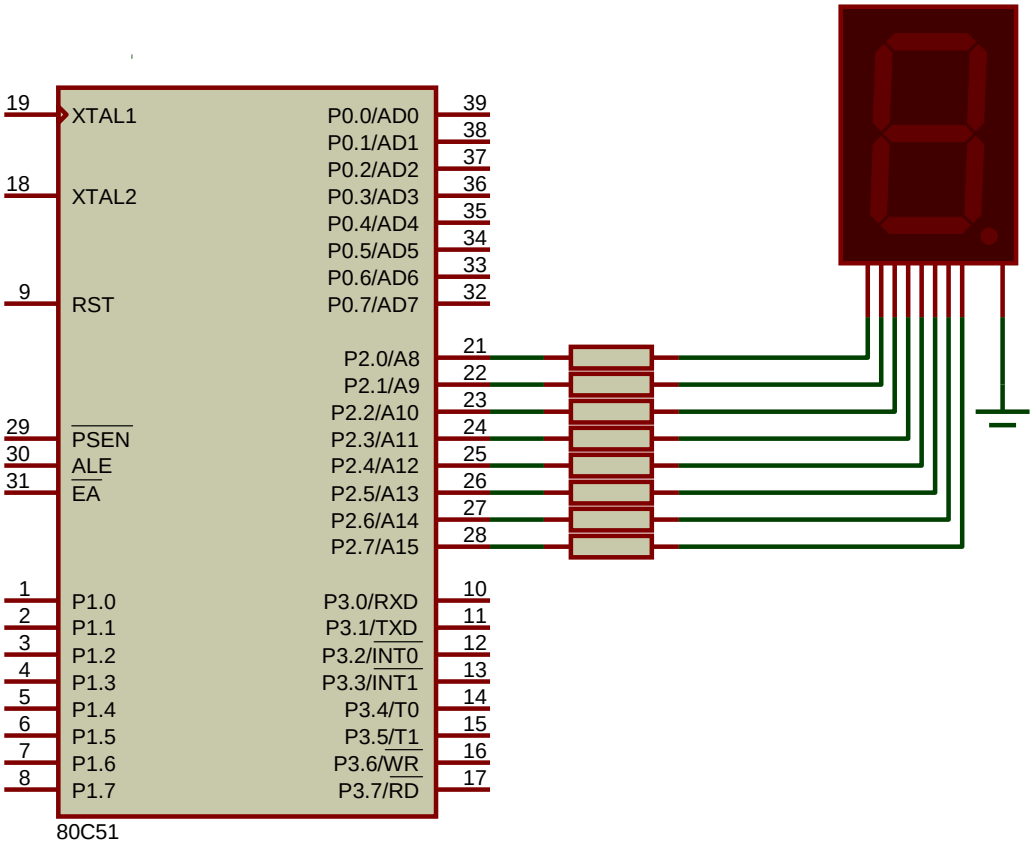

Рис. 3. Подключение индикатора с общим катодом

Так как светодиод обладает практически нулевым сопротивлением, для ограничения тока через каждый светодиод в схеме (рис. 3) установлены резисторы. Величина сопротивления токоограничивающего резистора определяется, исходя из свойств индикатора и микроконтроллера.

1.2.2. Использование панелей семисегментных индикаторов. Динамическая индикация

При необходимости использования нескольких индикаторов с целью экономии числа линий связи выводы сегментов индикаторов объединяются (рис. 4).

В этом случае для отображения информации на индикаторах используют *принцип динамической индикации*, основанный на инерции человеческого зрения. Применение этого принципа заключается в поочередном включении индикаторов, при этом частота включения должна быть более 30 Гц. В этом случае человек увидит все индикаторы включенными одновременно.

Включение и выключение каждого индикатора производится путем подачи соответствующего уровня напряжения на его общий вывод. В связи с этим, в отличие от использования одноразрядного индикатора, при использовании многоразрядных индикаторных панелей общий вывод каждого индикатора соединяется с микроконтроллером (рис. 4).

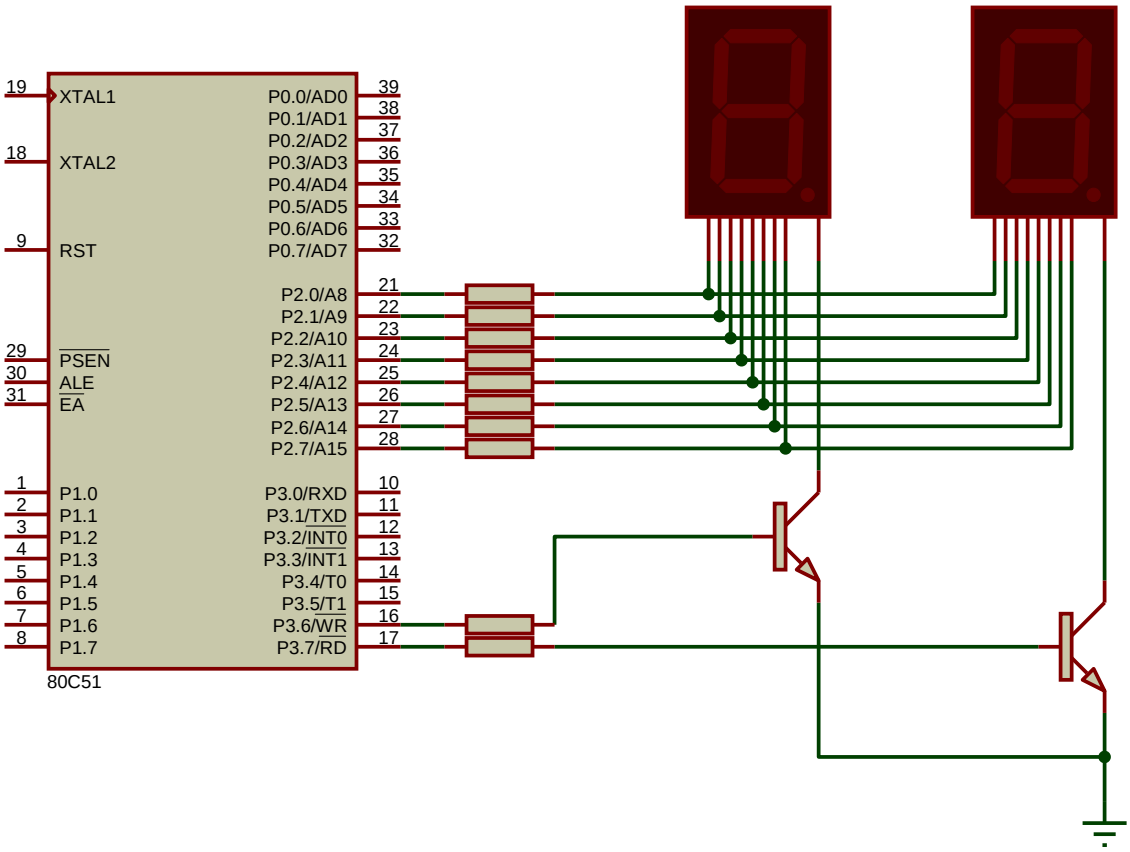

Рис. 4. Подключение многоразрядного индикатора с общим катодом

Если предельное допустимое значение входного тока на выводе микроконтроллера меньше, чем суммарный ток, который может получиться на общем выводе индикатора при одновременной активации всех его светодиодов, следует использовать транзисторный ключ (например, рис. 4). Тип транзистора и величина сопротивлений резисторов зависит от характеристик микроконтроллера и индикаторов.

1.2.3. Использование светодиодных матриц

Семисегментные индикаторы удобны для отображения цифр. Более широкие возможности представляет светодиодная матрица – набор светодиодов, расположенных в виде матрицы (рис. 5). При этом светодиоды, находящие в

одной строке, имеют общий катод, а светодиоды, находящиеся в одном столбце – общий анод. Светодиодные матрицы выпускаются с разным числом строк и столбцов, но наибольшее распространение имеют матрицы 8 × 8.

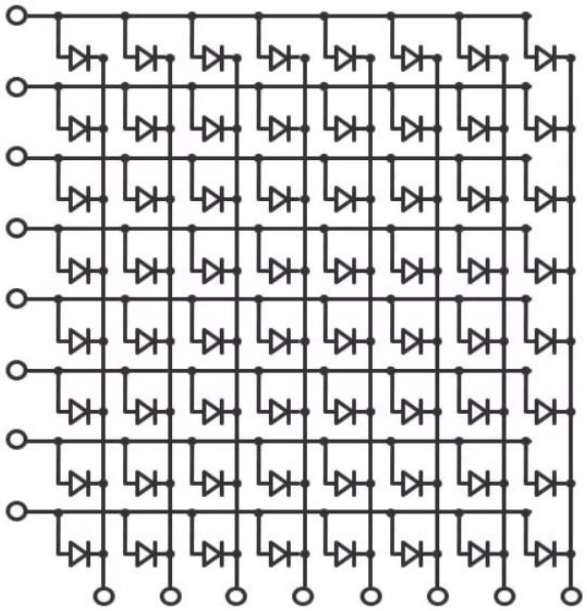

Рис. 5. Светодиодная матрица 8 ×8

Использование светодиодных матриц для отображения информации происходит также за счет динамической индикации: в каждый отельный момент времени активной является только одна строка матрицы (построчная динамическая индикация). Аналогично возможна динамическая индикация по столбцам.

Очевидно, что строки (столбцы), содержащие одинаковую информацию, могут активироваться одновременно.

1.2.4. Пример использования светодиодной матрицы для вывода изображения

Рассмотрим пример вывода на светодиодную матрицу размером 8 × 8 смайлика.

Для управления строками матрицы будем использовать порт P1, а для управления столбцами – порт P3 (рис. 6).

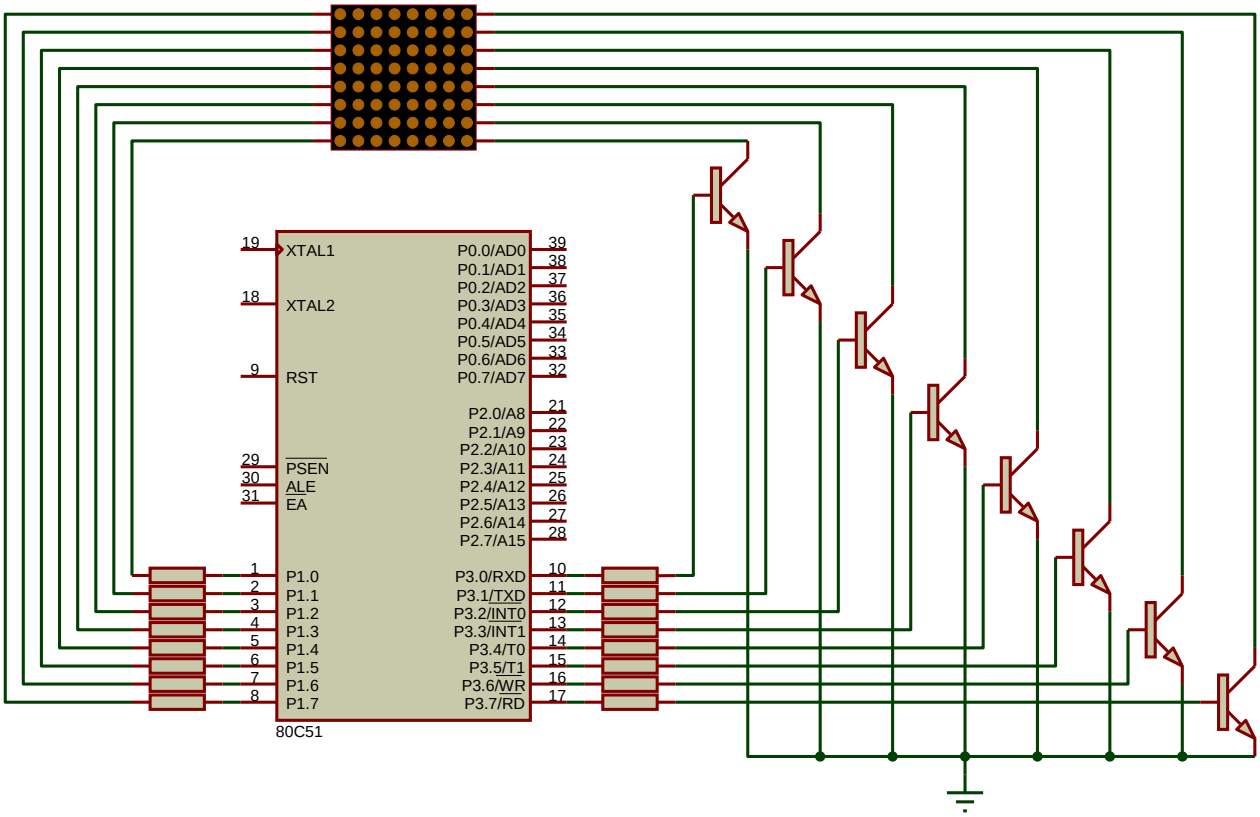

Рис. 6. Подключение светодиодной матрицы к микроконтроллеру

Так как предельное допустимое значение входного тока на выводе микроконтроллера составляет 10 мА и это значение меньше, чем суммарный ток, который может получиться на общем проводе столбца при одновременной активации всех его светодиодов, для каждого столбца в схеме использован транзисторный ключ.

Диоды в столбцах матрицы объединены катодами, следовательно, для активации столбца необходимо подать 1 на соответствующий разряд порта P3, чтобы открыть транзисторный ключ.

Поскольку символ симметричен, будем активировать сразу по 2 столбца: сначала второй и предпоследний, затем – третий и шестой и наконец – два средних.

Текст программы вывода смайлика на матрицу

```
;*****************************************************************
; *
; Filename: LEDMatrix.asm
; Date: 2021/05/12
; File Version: 1
; Author: Solov'eva T.N.
; Company: SUAI
; Description:
; *
;*****************************************************************
; Reset Vector
;*****************************************************************
org    0h              ; processor reset vector
    ajmp    start      ; go to beginning of program
;*****************************************************************
; MAIN PROGRAM
;*****************************************************************
org    100h
start:
     mov P3,#0
loop:
     mov P1,#0           ;очищаем
     mov P3,#01000010b   ;второй и предпоследний столбец
     mov P1,#00000100b
     lcall delay
     mov P1,#0
     mov P3,#00100100b   ;третий и шестой столбец
     mov P1,#00100010b
     lcall delay
     mov P1,#0
     mov P3,#00011000b   ;два средних столбца
     mov P1,#00000010b
     lcall delay
     sjmp loop

delay:                   ;подпрограмма задержки
     nop
     nop
     nop
     nop
     nop
     nop
     ret

finish: sjmp $ ;конец программы

end
```

На рис. 7 представлен результат выполнения программы в симуляторе *MCU 8051 IDE*.

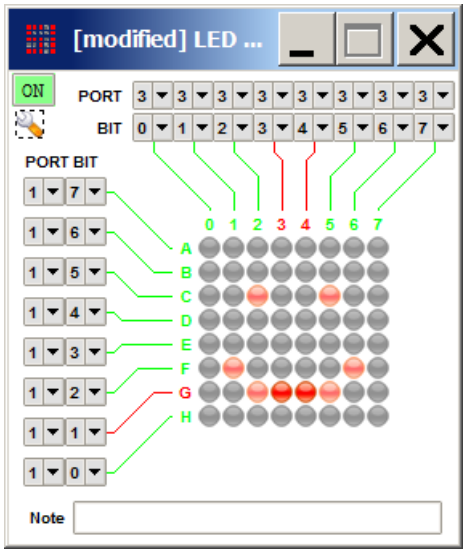

Рис. 7. Результат выполнения программы

На рис. 7 видно, что активными в текущий момент времени являются два средних столбца. Предварительно в симуляторе необходимо настроить матрицу (Light up when: Row 1 & Column 1).

Следует отметить, что в программе реализована задержка на 10 мкс. В реальных системах эта задержка может быть сделана больше (например, 10 мс).

2. Задание по работе

Требуется разработать программу на языке ассемблера MCS-51 для вывода заданных символов на семисегментный индикатор (с общим катодом – для четных вариантов, с общим анодом – для нечетных) и светодиодную матрицу 8 × 8.

Выводы, через которые к МК подключается индикатор и матрица, а также символы, которые необходимо вывести на индикатор и матрицу, указаны в разделе «Варианты заданий».

Работу программы необходимо проверить с помощью симулятора.

3. Порядок выполнения работы

Выполнение работы состоит из двух частей: разработки программы и проверки корректности ее работы с помощью симулятора.

При разработке программы необходимо:

1) составить функциональную схему, отражающую подключение индикатора и светодиодной матрицы к МК с использованием указанных в задании выводов;

2) разработать алгоритм вывода символов на индикатор и светодиодную матрицу;

3) на основании полученного алгоритма составить текст программы на языке ассемблера МК.

Для выполнения работы *на симуляторе MCU 8051 IDE* необходимо выполнить следующие действия.

1. Запустить оболочку симулятора MCU 8051 IDE.
2. Создать новый проект (Project: New).
3. В окно редактора вести текст разработанной программы. Сохранить файл с текстом программы, добавив его в проект. Выполнить компиляцию программы (Tools: Compile).
4. Подключить семисегментный индикатор (Virtual HW: LED Display) и светодиодную матрицу (Virtual HW: LED Matrix) в соответствии разработанной схемой. Настроить тип индикатора (Common electrode) и матрицу (Light up when).
5. Запустить режим симуляции (Simulator: Start/Shutdown) и проверить работоспособность программы.

4. Содержание отчета

Правила оформления отчета приведены в предисловии к практикуму.

Отчет по лабораторной работе должен содержать следующие разделы.

1. Цель работы.
2. Задание по работе.

Раздел должен включать в себя формулировку задания и данные варианта.

3. Разработка программы.

В данном разделе необходимо привести схему алгоритма (если она использовалась при разработке программы), текст программы с подробными комментариями, а также скриншоты с результатами выполнения программы.

4. Вывод.

Пример вывода: «В результате выполнения работы разработана программа на языке ассемблера MCS-51 для вывода символов на семисегментный индикатор и светодиодную матрицу. Проверка работоспособности программы произведена в среде *MCU 8051 IDE*. Приобретены навыки организации взаимодействия микроконтроллера с простейшими устройствами вывода».

5. Контрольные вопросы

1. Почему для семисигнментного индикатора для ограничения тока вместо восьми резисторов не используют один резистор, подключаемый к общему выводу?
2. Выведите первую букву своей фамилии на светодиодную матрицу.
3. С какой целью светодиоды объединяются в матрицу?
4. Выведите номер группы на чтырехразрядный семисегментный индикатор.
5. В чем состоит принцип динамической индикации?
6. С какой целью при подключении светодиодных элементов к микроконтроллеру используют резисторы?

7. Почему при использовании многоразрядных семисегментных индикаторов общий вывод каждого индикатора подключают к микроконтроллеру через транзистор?

6. Варианты заданий

Номер варианта	Семисегментный индикатор		Светодиодная матрица		
	Шина данных	Символ	Строки	Столбцы	Символ
1	P2	1	P3	P0	А
2	P1	7	P2	P3	Б
3	P2	П	P3	P1	В
4	P1	C	P2	P0	Г
5	P0	U	P3	P1	Д
6	P3	0	P2	P1	Е
7	P2	-	P1	P0	Ё
8	P1	3	P0	P3	Ж
9	P0	2	P3	P2	З
10	P3	d	P1	P0	И
11	P3	4	P2	P0	Й
12	P2	9	P1	P3	К
13	P0	5	P2	P1	Л
14	P3	У	P2	P0	М
15	P1	F	P0	P2	Н
16	P1	P	P3	P2	О
17	P1	h	P3	P0	П
18	P2	H	P0	P3	Р
19	P2	A	P1	P0	С
20	P1	E	P2	P3	Т
21	P3	b	P0	P1	У
22	P0	6	P1	P2	Ф
23	P0	8	P1	P3	Х
24	P0	∀	P2	P3	Ц
25	P3	L	P2	P1	Ч
26	P3	Э	P1	P0	Ш
27	P1	Г	P0	P3	Щ
28	P2	11	P3	P1	Ъ
29	P0	=	P3	P2	Ы
30	P1	ŋ	P0	P2	Ь
31	P3	⊇	P1	P0	Э

Номер варианта	Семисегментный индикатор		Светодиодная матрица		
	Шина данных	Символ	Строки	Столбцы	Символ
32	P2	o	P0	P3	Ю
33	P3	⊆	P2	P1	Я
34	P2	⌐	P0	P1	D
35	P3	¯	P2	P1	F
36	P2	Ƌ	P0	P1	G
37	P1	ϐ	P3	P0	I
38	P0	⊦	P1	P3	J
39	P3	Ξ	P2	P0	L
40	P1	ʁ	P0	P2	N
41	P2	⊢	P1	P0	Q
42	P1	⌐	P2	P3	R
43	P2	⌐	P0	P1	S
44	P2	I	P0	P3	U
45	P0	⌋	P2	P1	V
46	P3	⌊	P1	P2	W
47	P1	Ϙ	P3	P0	Y
48	P0	∩	P1	P3	Z
49	P1	∪	P2	P0	Δ
50	P2	Ł	P0	P3	Σ

Лабораторная работа 3
ОРГАНИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ МИКРОКОНТРОЛЛЕРА С ЖИДКОКРИСТАЛЛИЧЕСКИМ ДИСПЛЕЕМ

Цель работы: приобретение навыков организации взаимодействия микроконтроллера с устройствами вывода на примере жидкокристаллического знакосинтезирующего дисплея.

1. Использование жидкокристаллического дисплея

Для вывода текстовой информации в микроконтроллерных системах удобно использовать небольшие знакосинтезирующие жидкокристаллические дисплеи (жидкокристаллические индикаторы (ЖКИ)).

Многие современные ЖКИ, предназначенные для подключения к параллельным портам ввода-вывода МК, построены на основе контроллера HD44780 фирмы Hitachi или его аналогах. Такие дисплеи имеют от 1 до 4 строк и от 8 символов в строке.

1.1. Устройство и система команд дисплея на базе HD44780

ЖКИ на базе HD44780 содержит видеопамять (Display Data RAM, DDRAM), в которой хранятся ASCII-коды отображаемых символов, а также обладает собственной системой команд для управления жидкокристаллической панелью. Помимо стандартных символов имеется возможность создания символов пользователем в памяти CGRAM (Character Generator RAM).

Пример подключения ЖКИ к МК показан на рис. 1.

ЖКИ имеет восьмиразрядную шину команд-данных, которая на рис. 1 подключена к порту P2, и шину управления, в состав которой входят одноразрядные линии: *разрешения* программирования (E), *выбора типа посылки* (RS) и *выбора направления передачи* (RW). Эти линии на рис. 1 подключены к трем старшим разрядам порта P1 (P1.7, P1.5 и P1.6 соответственно).

Прием или выдача информации на входах ЖКИ происходит по спаду сигнала на входе E. При RW = 0 происходит чтение информации на шине DB, при RW = 1

– выдача информации из ЖКИ на шину DB. При RS = 0 информация на шине DB интерпретируется как команда, при RS = 1 – как данные.

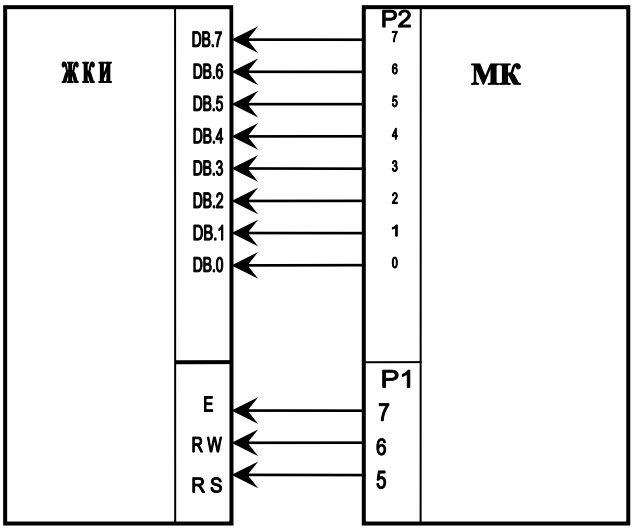

Рис. 1. Пример подключения клавиатуры и ЖКИ к микроконтроллеру

Система команд ЖКИ приведена в табл. 1.

Ниже приведены назначение значения отдельных битов команд.

I/D = 0 декремент AC при записи, I/D = 1 инкремент AC при записи.

S = 1 сдвиг окна дисплея при записи нового символа в DDRAM.

D = 1 включить дисплей.

C = 1 включить курсор.

B = 1 курсор в виде мигающего черного квадрата.

S/C = 0 сдвиг курсора на 1 позицию, S/C = 1 сдвиг окна на 1 позицию.

R/L – направление сдвига курсора и окна (0 влево, 1 вправо).

D/L = 1 шина данных 8 бит, D/L = 0 шина данных 4 бита.

N = 0 одна строка, N = 1 две строки.

F = 0 размер символа 5x8 точек, F = 1 размер символа 5x10 точек.

AG – адрес в памяти CGRAM.

AD – адрес в памяти DDRAM.

BF – флаг занятости.

AC – адрес, хранящийся в указателе адреса ЖКИ.

DT – данные.

Таблица 1

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Значение	Время выполнения, мкс
0	0	0	0	0	0	0	0	0	1	Очистка экрана, AC=0	от 82 до 1640
0	0	0	0	0	0	0	0	1	–	AC=0, сброс сдвигов окна, содержимое DDRAM сохраняется	от 40 до 1600
0	0	0	0	0	0	0	1	I/D	S	Настройка сдвига окна и курсора при записи	40
0	0	0	0	0	0	1	D	C	B	Настройка режима отображения	40
0	0	0	0	0	1	S/C	R/L	–	–	Сдвиг курсора или окна	40
0	0	0	0	1	DL	N	F	–	–	Выбор числа строк, ширины шины данных и размера символа	40
0	0	0	1	AG	AG	AG	AG	AG	AG	Переключить адресацию на CGRAM и задать адрес в CGRAM	40
0	0	1	AD	AD	AD	AD	AD	AD	AD	Переключить адресацию на DDRAM и задать адрес в DDRAM	40
0	1	BF	AC	AC	AC	AC	AC	AC	AC	Прочитать флаг занятости BF и регистр адреса AC	1
1	0	DT	DT	DT	DT	DT	DT	DT	DT	Записать данные в CGRAM или DDRAM	40
1	1	DT	DT	DT	DT	DT	DT	DT	DT	Прочитать данные из CGRAM или DDRAM	40

После перезагрузки ЖКИ управляющие биты имеют следующие значения:

I/D = 1;	D =0;	D/L = 1;
S = 0;	C = 0;	N = 0;
	B = 0;	F = 0.

Перед выполнением команд обращения к памяти ЖКИ (последние 2 команды в табл. 1) необходимо выполнить команду установки адреса (7 или 8 строка в табл. 1). Если подряд выполняются несколько команд обращения к памяти, то достаточно установить адрес один раз – перед первой командой обращения – далее адрес будет автоматически изменяться в соответствии со значением бита I/D.

Во время выполнения дисплеем команды флаг BF устанавливается. по окончании выполнения команды происходит сброс этого флага.

Последний столбец табл. 1 содержит информацию о времени выполнения каждой команды. При выполнении команды все обращения дисплей игнорирует, поэтому в программе работы с дисплеем следует предусмотреть задержки между обращениями к ЖКИ. Большинство команд выполняются за 40 мкс. Исключение составляют команда чтения BF-AC, выполняемая в течение 1 мкс, а также команды очистки дисплея и очистки сдвигов (первые 2 строки в табл. 1), время выполнения которых зависит от каждого конкретного случая. При использовании команд очистки рекомендуется проверять флаг BF для контроля завершения команды.

С целью экономии выводов возможно подключение дисплея по 4-разрядной шине. В этом случае контакты DB0-DB3 не используются, и для передачи байта дисплею необходимо сначала передать старший полубайт, а затем младший. Первой командой, отправляемой дисплею в этом режиме, должна быть команда настройки размера шины данных. Поскольку при запуске дисплея D/L = 1, байт команды будет прочитан сразу, при этом младший полубайт определится состоянием неиспользуемых выводов DB0-DB3, после чего дисплей перейдет в режим 4-разрядной шины.

1.2. Пример программы вывода строк на ЖКИ

Рассмотрим пример вывода на ЖКИ размером 2 × 20 символов, подключенный к МК семейства MCS-51 (рис. 1), двух строк: «it is test example.» и «0123456789ABCDFI@#$».

Графическая схема соответствующего алгоритма приведена на рис. 2, а ниже располагается текст программы для стенда МК.

Выводимая строка символов записана с помощью директивы компилятора db в виде последовательности кодов символов в памяти программ микроконтроллера, начиная с адреса 0FD0h (см. конец программы).

Рис. 2. Графическая схема алгоритма вывода строк на ЖКИ

Текст программы вывода строк на ЖКИ

```
;*****************************************************************
; *
; Filename: ex3.asm
; Date: 2020/05/12
; File Version: 1
; Author: Solov'eva T.N.
; Company: SUAI
; Description: example 3
; *
;*****************************************************************
; Variables
;*****************************************************************
switch    equ  43h  ;переключатель «команда-данные» (RS)
bte  equ  44h       ;выдаваемый на ЖКИ байт
;*****************************************************************
;                              Reset                              Vector
;*****************************************************************
org    0h              ; processor reset vector
    ajmp    start      ; go to beginning of program
;*****************************************************************
; MAIN PROGRAM
;*****************************************************************
org    100h
start:
     ;инициализация ЖКИ
indic_init:    mov  switch, #0;переключатель уст-ть на команду (RS=0)
     mov  bte, #38h ;байт – команда
     lcall     indic_wr  ;вызов подпрограммы передачи в ЖКИ
     mov  bte, #0ch ;активация всех знакомест
     lcall     indic_wr
     mov  bte, #06h ;режим автом. перемещения курсора
     lcall     indic_wr
     mov  bte, #80h ;установка адреса первого символа
     lcall     indic_wr
     ;вывод строк
     mov  switch, #1     ;переключатель – данные (RS=1)
     mov  dptr, #0fd0h   ;адрес, по которому расположены данные
     ;(см. конец программы)
indic_data_wr1:               ;вывод символов первой строки
     clr  a
     movc a, @a+dptr
ind_row1: mov  bte, a    ;передаваемый байт – код символа
     lcall     indic_wr
     inc  dptr
     mov  a, dpl         ;младший байт указателя данных
     cjne a, #0E3h, indic_data_wr1
     ;пока не введены 19 символов 1ой строки
     mov  switch, #0     ;RS=0 – команда
     mov  bte, #0C0h     ;установка адреса первого символа
     lcall     indic_wr  ;второй строки
     mov  switch, #1     ;RS=1 - данные
```

```
indic_data_wr2:                ;вывод символов второй строки
     clr  a
     movc a, @a+dptr
ind_row2: mov  bte, a
     lcall     indic_wr
     inc  dptr
     mov  a, dpl
     cjne a, #0F6h, indic_data_wr2
     ;E3h+13h=F6h – адр. конца второй стр.
     jmp  finish    ;переход на конец программы

     ;подпрограмма передачи в ЖКИ
indic_wr: mov  p2, bte   ;передаваемый байт – в P2
     setb p1.7      ;E:=1
     clr  p1.6      ;RW:=0 (запись)
     mov  a, switch
     mov  c, acc.0  ;нам нужен 0-ой бит аккумулятора
     mov  p1.5, c        ;RS:=switch (команда/данные)
     lcall     indic_delay    ;вызов подпрограммы задержки
     clr  p1.7      ;E:=0
     lcall     indic_delay
     setb p1.7      ;E:=1
     ret
indic_delay:             ;подпрограмма задержки на 40мкс
     push A              ;сохраняем аккумулятор в стеке
     mov A, #0Ah         ; 40 = 2+2+1+A(1+2)+1+2+2
  m: dec A
     jnz m
     nop
     pop A               ;восстанавливаем значение аккумулятора
     ret

;данные располагаем в памяти программ
org 0FD0h
data:     db   'it is test example.'
     db   '0123456789ABCDFI@#$'    ;директива db помещает коды
               ;символов в последовательные ячейки памяти программ

finish: sjmp $ ;конец программы
end
```

Подпрограмма `indic_delay` осуществляет задержку на 40 мкс следующим образом. Команды `lcall`, `push`, `jnz`, `pop` и `ret` выполняются в течение двух машинных циклов, команды `mov`, `dec` и `nop` – в течение одного (см. табл. 6 в работе 1), поэтому время выполнения подпрограммы `indic_delay` без учета `nop` составляет

$$2 + 2 + 1 + A\cdot(1 + 2) + 2 + 2 = 9 + 3A \text{ машинных цикла,}$$

то есть зависит от содержимого A. При тактовой частоте 12 МГц один машинный цикл выполняется за 1 мкс, поэтому для того, чтобы подпрограмма `indic_delay` осуществляла задержку на 40 мкс можно становить значение A = 10 = 0Ah и добавить одну команду `nop` после цикла.

2. Задание по работе

Требуется разработать программу на языке ассемблера MCS-51 для вывода на экране ЖКИ двух заданных строк. Строки необходимо выровнять по центру экрана, при этом содержимое строк, расположенное в памяти программ, не должно содержать пробелы до первого и после последнего символа. Содержание строк указано ниже.

	Строка 1	Строка 2
нечетные варианты	номер группы	И.О. Фамилия
четные варианты	И.О. Фамилия	номер группы

Выводы, через которые к МК подключается ЖКИ, и вид курсора указаны в разделе «Варианты заданий».

Работу программы необходимо проверить с помощью симулятора.

3. Порядок выполнения работы

Выполнение работы состоит из двух частей: разработки программы и проверки корректности ее работы с помощью симулятора.

При разработке программы необходимо:

1) составить функциональную схему, отражающую подключение ЖКИ к МК с использованием указанных в задании выводов;

2) разработать алгоритм вывода строк на ЖКИ;

3) на основании полученного алгоритма составить текст программы на языке ассемблера МК.

Для выполнения работы *на симуляторе MCU 8051 IDE* необходимо выполнить следующие действия.

1. Запустить оболочку симулятора MCU 8051 IDE.
2. Создать новый проект (Project: New).

3. В окно редактора вести текст разработанной программы. Сохранить файл с текстом программы, добавив его в проект. Выполнить компиляцию программы (Tools: Compile).

4. Подключить ЖКИ (Virtual HW: LCD display: 2 × 20) в соответствии с п. 3. Перед запуском программы рекомендуется выполнить перезагрузку ЖКИ (настройки: Reset HD44780). В случае подключения по 4-разряжной шине на неиспользуемые выводы дисплея необходимо подать определенный уровень напряжения (например, подключить к любому неиспользуемому выводу портов P1-P3).

5. Запустить режим симуляции (Simulator: Start/Shutdown) и проверить работоспособность программы.

Для выполнения работы *на стенде МК* необходимо выполнить следующие действия.

1. Если после проверки программы на симуляторе планируется проверка на стенде контроллера, необходимо сначала подключить стенд к компьютеру, выполнить необходимые соединения портов МК с ЖКИ в соответствии с разработанной в отчете схемой и включить стенд.

2. Запустить оболочку Shell51.

3. Ввести программу. *Внимание!* При выполнении работы на стенде следует заменить указанный в задании порт P2 на порт P4 или P5, используя прямую адресацию (адреса портов см. в описании работы 2).

При оформлении текста программы необходимо руководствоваться следующими правилами:

– первая строка программы должна содержать директиву ассемблеру о начальном адресе программы вида: `org 8100h` (пользовательская программа на стенде выполняется как подпрограмма, диапазон доступных пользователю адресов в памяти программ: 8000h – FFFFh; начальные адреса этой области отведены под пользовательские вектора прерываний; основную программу рекомендуется размещать, начиная с адреса 8100h);

– при объявлении переменных после имени переменной ставят двоеточие;
– каждая последующая строка должна начинаться с метки или со знака табуляции;
– последняя строка программы должна содержать команду `ret`.

4. Сохранить текст программы в виде файла с расширением *.asm.

5. Выполнить трансляцию программы, нажав кнопку «Запуск». Появление в специальном окне слова «Внимание» указывает на наличие ошибки в программе. В этом случае нужно открыть вкладку «Листинги», которая содержит результаты трансляции. Пользуясь информацией на вкладке, необходимо исправить ошибки и повторить трансляцию программы. Внимание! При отладке следует игнорировать ошибки компиляции, связанные с адресами портов P4 и P5.

6. Убедиться, что стенд правильно подсоединен к компьютеру. С этой целью открыть вкладку «Окна памяти». Для проверки связи следует нажать копку «Передать из контроллера в ПК». В ходе обмена информацией между компьютером и контроллером поле «Состояние связи» отображает статус линий связи. «Обмен с МК» свидетельствует о наличии очередного сеанса обмена, «Нет связи» – о невозможности взаимодействия с МК, при этом дальнейшие сеансы обмена блокируются. Необходимо выявить и устранить причину, а затем снять блокировку двойным щелчком по сообщению. В большинстве случаев при потере связи требуется завершить работу симулятора, выключить стенд, затем включить его снова и снова запустить симулятор.

7. Занести программу в память программ контроллера, для чего применить кнопку «Загрузка».

8. С помощью кнопки «Запуск» запустить программу.

9. Проверить работу программы.

4. Содержание отчета

Правила оформления отчета приведены в предисловии к практикуму.

Отчет по лабораторной работе должен содержать следующие разделы.

1. Цель работы.
2. Задание по работе.

   Раздел должен включать в себя формулировку задания и данные варианта.
3. Разработка программы.

   В данном разделе необходимо привести схему алгоритма (если она использовалась при разработке программы), текст программы с подробными комментариями, а также скриншоты с результатами выполнения программы.
4. Вывод.

Пример вывода: «В результате выполнения работы разработана программа на языке ассемблера MCS-51 для вывода на экране ЖКИ двух заданных строк. Проверка работоспособности программы произведена в среде *MCU 8051 IDE*. Приобретены навыки организации взаимодействия микроконтроллера с устройствами вывода на примере жидкокристаллического знакосинтезирующего дисплея».

5. Контрольные вопросы

1. Опишите назначение выводов дисплея.
2. Добавьте очистку дисплея при инициализации ЖКИ. Предусмотрите контроль флага BF для контроля окончания выполнения команды.
3. Что необходимо изменить в программе вывода заголовков на ЖКИ, приведенной в разделе 1.2, если текст заголовков размещается в памяти, начиная с адреса:
   a. 0FC0h,
   b. 0EA0h.
4. Как организовать вывод на ЖКИ символа «4» на позиции 12

a) первой строки; b) второй строки?

5. В программе, приведенной в разделе 1.2, switch – переменная типа байт, предназначенная для хранения одного бита. Измените программу так, чтобы switch была переменной типа бит.
6. Фамилию, имя и отчество выведите на ЖКИ полностью, в формате «бегущей строки».
7. ФИО выведите на ЖКИ, используя кириллицу. Символы кириллицы допустимо изобразить в CGRAM с помощью мышки.
8. Пусть клавиша подключена к разряду P2.5 микроконтроллера, к разряду P3.7 подключен светодиод. Напишите программу засветки светодиода при нажатии клавиши.
9. Выведите первую букву своей фамилии на светодиодную матрицу.

6. Варианты заданий

Номер варианта	Шина управления ЖКИ			Шина данных ЖКИ	Вид курсора
	RS	RW	E		
1	P1.0	P1.1	P1.2	P2.0 – P2.7	нет
2	P1.4	P1.5	P1.6	P2.0 – P2.3	подчеркивание
3	P1.1	P1.2	P1.3	P2.4 – P2.7	мигающий
4	P1.5	P1.6	P1.7	P2.0 – P2.7	мигающий
5	P1.0	P1.1	P1.2	P2.0 – P2.3	нет
6	P1.5	P1.6	P1.4	P2.4 – P2.7	подчеркивание
7	P1.0	P1.1	P1.2	P2.0 – P2.7	нет
8	P1.1	P1.0	P1.3	P2.0 – P2.3	подчеркивание
9	P1.5	P1.4	P1.6	P2.4 – P2.7	мигающий
10	P1.0	P1.2	P1.3	P2.0 – P2.7	мигающий
11	P1.7	P1.6	P1.5	P2.0 – P2.3	нет
12	P1.1	P1.2	P1.3	P2.4 – P2.7	подчеркивание
13	P1.1	P1.0	P1.2	P2.0 – P2.7	нет
14	P1.0	P1.2	P1.3	P2.0 – P2.3	подчеркивание
15	P1.0	P1.6	P1.4	P2.4 – P2.7	мигающий
16	P1.6	P1.4	P1.5	P2.0 – P2.7	мигающий
17	P1.4	P1.6	P1.5	P2.0 – P2.3	нет
18	P1.0	P1.2	P1.1	P2.4 – P2.7	подчеркивание
19	P1.0	P1.2	P1.5	P2.0 – P2.7	нет
20	P1.4	P1.2	P1.1	P2.0 – P2.3	подчеркивание
21	P1.6	P1.1	P1.0	P2.4 – P2.7	мигающий
22	P1.7	P1.2	P1.1	P2.0 – P2.7	мигающий
23	P1.5	P1.3	P1.0	P2.0 – P2.3	нет
24	P1.5	P1.7	P1.2	P2.4 – P2.7	подчеркивание
25	P1.3	P1.7	P1.6	P2.0 – P2.7	нет
26	P1.7	P1.4	P1.6	P2.0 – P2.3	подчеркивание
27	P1.4	P1.6	P1.5	P2.4 – P2.7	мигающий
28	P1.1	P1.0	P1.2	P2.0 – P2.7	мигающий
29	P1.4	P1.5	P1.7	P2.0 – P2.3	нет
30	P1.1	P1.2	P1.0	P2.4 – P2.7	подчеркивание
31	P1.0	P1.3	P1.1	P2.0 – P2.7	нет
32	P1.1	P1.2	P1.0	P2.0 – P2.3	подчеркивание
33	P1.5	P1.7	P1.6	P2.4 – P2.7	мигающий
34	P1.5	P1.7	P1.4	P2.0 – P2.7	мигающий
35	P1.2	P1.3	P1.1	P2.0 – P2.3	нет
36	P1.7	P1.5	P1.4	P2.4 – P2.7	подчеркивание

37	P1.3	P1.0	P1.2	P2.0 – P2.7	нет
38	P1.2	P1.3	P1.1	P2.0 – P2.3	подчеркивание
39	P1.3	P1.2	P1.1	P2.4 – P2.7	мигающий
40	P1.0	P1.3	P1.1	P2.0 – P2.7	мигающий
41	P1.7	P1.6	P1.5	P2.0 – P2.3	нет
42	P1.3	P1.5	P1.4	P2.4 – P2.7	подчеркивание
43	P1.3	P1.2	P1.4	P2.0 – P2.7	нет
44	P1.4	P1.2	P1.5	P2.0 – P2.3	подчеркивание
45	P1.3	P1.6	P1.1	P2.4 – P2.7	мигающий
46	P1.7	P1.5	P1.4	P2.0 – P2.7	мигающий
47	P1.7	P1.2	P1.3	P2.0 – P2.3	нет
48	P1.0	P1.6	P1.5	P2.4 – P2.7	подчеркивание
49	P1.5	P1.4	P1.7	P2.0 – P2.7	нет
50	P1.7	P1.0	P1.5	P2.0 – P2.3	подчеркивание

Лабораторная работа 4
ОРГАНИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ МИКРОКОНТРОЛЛЕРА С УСТРОЙСТВАМИ ВВОДА

Цель работы: приобретение навыков организации взаимодействия микроконтроллера с устройствами ввода на примере клавиатуры.

1. Применение устройств ввода в микроконтроллерных системах

Устройства ввода также как и устройства вывода являются внешними устройствами, подключаемыми к микроконтроллеру через его внешние контакты – выводы портов.

Простейшим устройством ввода является кнопка (клавиша). В случае если в микроконтроллерной системе используется достаточно большое число клавиш, с целью сокращения числа сигнальных линий клавиши и объединяются в клавиатуру (матрицу). Если каждую клавишу подключить к разряду порта с помощью отдельной линии, это может привести к использованию большого числа линий и выводов портов для связи клавиш с МК.

1.1. Использование клавиатуры

Пример формирования клавиатуры 4×4 из 16 клавиш показан в правой части рис. 1. Из рис. 1 следует, что вместо 16 сигнальных линий в случае использования отдельной линии для подключения каждой клавиши к МК при объединении клавиш в клавиатуру используется лишь 8 сигнальных линий.

При организации клавиатуры 4×4 клавиши объединяются в группы (строки матрицы) из 4-х клавиш каждая. Одноимённые (на рис. 1 левые) контакты клавиш каждой группы объединяются, образуя вход клавиатуры. Всего таких входов 4. Клавиши разных строк, расположенных друг под другом, образуют столбец матрицы. Их правые контакты объединяются, образуя выход клавиатуры. Всего в данной структуре 4 выхода.

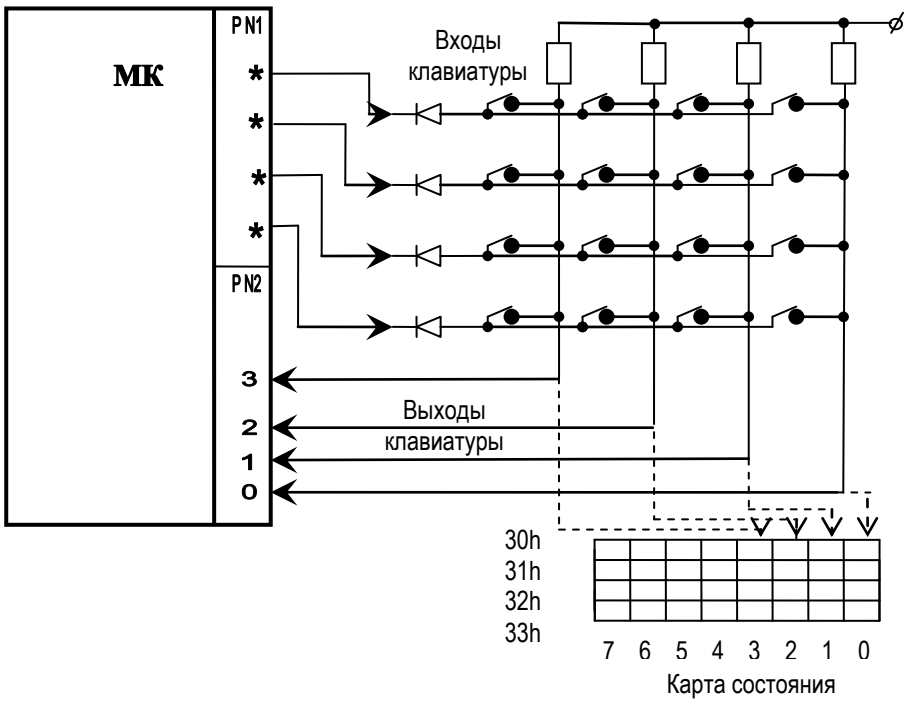

Рис. 1. Пример подключения клавиатуры к микроконтроллеру

При подключении к МК входы клавиатуры соединяются с разрядами порта, которые настраиваются на *вывод*; выходы клавиатуры соединяются с разрядами порта, которые настраиваются на *ввод* (при настройке вывода порта на ввод, нужно записать в соответствующий разряд регистра порта единицу).

1.1.1. Определение номера нажатой клавиши

При объединении клавиш в клавиатуру возникает задача идентификации, то есть определения номера нажатой клавиши (здесь и далее будем считать, что нажата одна клавиша). Как правило, эта задача решается путём так называемого *сканирования* входов клавиатуры с одновременным опросом сигналов на выходах клавиатуры. Сканирование входов означает поочерёдную подачу сигнала, как правило, низкого уровня на входы клавиатуры. При нахождении “0” на одном из входов опрос выходов клавиатуры позволяет определить состояние (замкнут – не замкнут) клавиш строки, связанной с данным входом. Значения сигналов на выходах клавиатуры записываются в некоторую ячейку памяти (см. рис. 1), после чего сигнал “0” поступает на другой вход клавиатуры, а значения сигналов на выходах клавиатуры, отражающие состояние клавиш следующей строки, записываются в следующую ячейку памяти. Таким образом, формируется так называемая *карта состояния* (КС) клавиатуры.

Отметим следующее:

1) количество ячеек памяти в КС совпадает с числом строк клавиатуры;
2) количество значащих разрядов (т.е. разрядов, содержащих информацию о состоянии клавиш) совпадает с числом столбцов клавиатуры;
3) для структуры на рис. 1 возможны пять вариантов значений значащих разрядов содержимого каждой ячейки памяти КС: 1111 (не нажата ни одна клавиши строки) и четыре варианта, соответствующие нажатой одной из четырёх клавиши строки (0111, 1011, 1101, 1110).

Решение задачи определения номера нажатой клавиши включает выполнение следующих шагов.

Шаг 1. Формирование карты состояния клавиатуры с помощью подачи «бегущего нуля» (сканирования, как описано выше).

Шаг 2. Определение номера строки, на которой находится нажатая клавиша.

Для решения этой задачи необходимо организовать поиск ячейки КС, в значащих разрядах которой находится “0”. Искомый номер строки r определяется, как смещение адреса найденной ячейки КС относительно начального адреса КС. Отметим, что $r = 0…3$ для структуры на рис. 1.

Шаг 3. Определение номера столбца s ($s = 0…3$ на рис. 1), в котором находится нажатая клавиша.

С этой целью необходимо осуществлять сдвиги содержимого ячейки КС, найденной на шаге 2, на один разряд до вытеснения из разрядной сетки нулевого значения. Фиксируется количество сдвигов, потребовавшихся для вытеснения “0”, которое и принимается как искомое значение s.

Следует обратить внимание на выбор направления, в котором производится сдвиги. Например, для структуры на рис. 1 сдвиги должны осуществляться вправо.

Шаг 4. Определение номера N нажатой клавиши.

Значение N находится путём вычисления следующего выражения**:**

$$N = rS + s + 1, \qquad (1)$$

где S – количество столбцов клавиатуры, ($N = 1…16$, $S = 4$ на рис. 1).

1.1.2. Пример программы определения номера нажатой клавиши

Рассмотрим пример решения задачи определения номера нажатой клавиши. Пусть 16 клавиш объединены в клавиатуру со структурой 8×2. Схема подключения клавиатуры к МК показана на рис. 2.

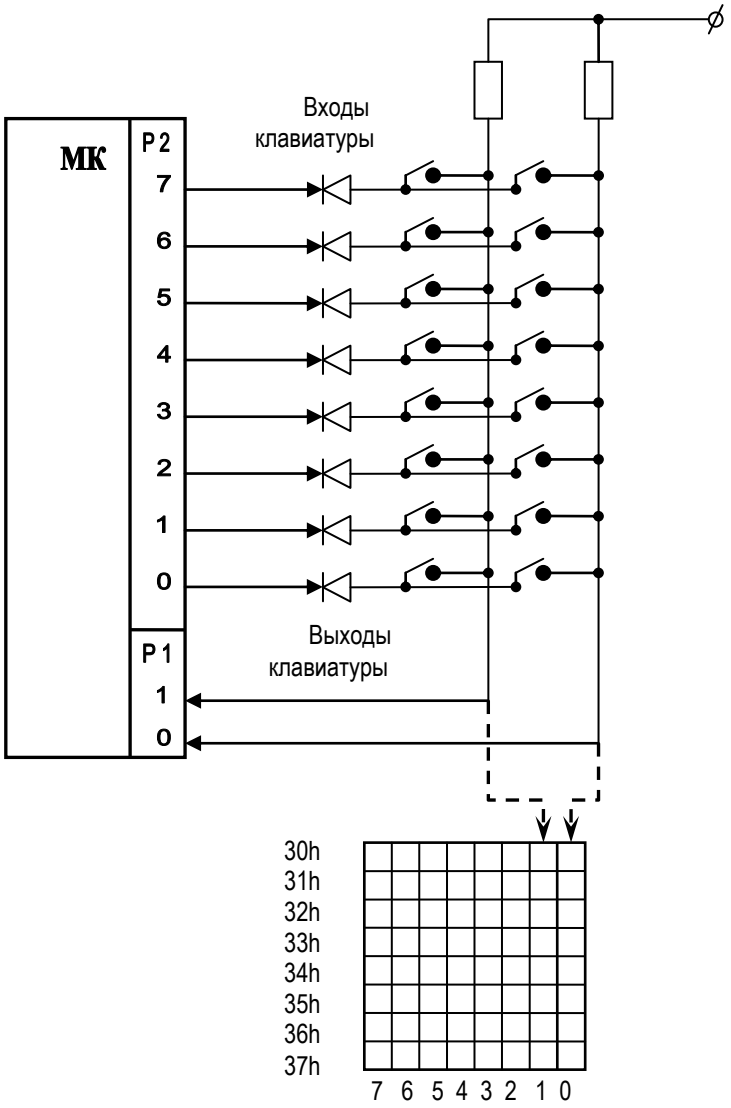

Рис. 2. Схема подключения клавиатуры 8×2 к МК. Карта состояния

Пусть для подсоединения входов клавиатуры к МК используется порт P2, для подключения выходов клавиатуры к МК используется два *младших* разряда порта P1. Основываясь на материале, изложенном выше, составим графическую схему алгоритма (рис. 3) и текст программы микроконтроллера (приведен текст программы для стенда МК).

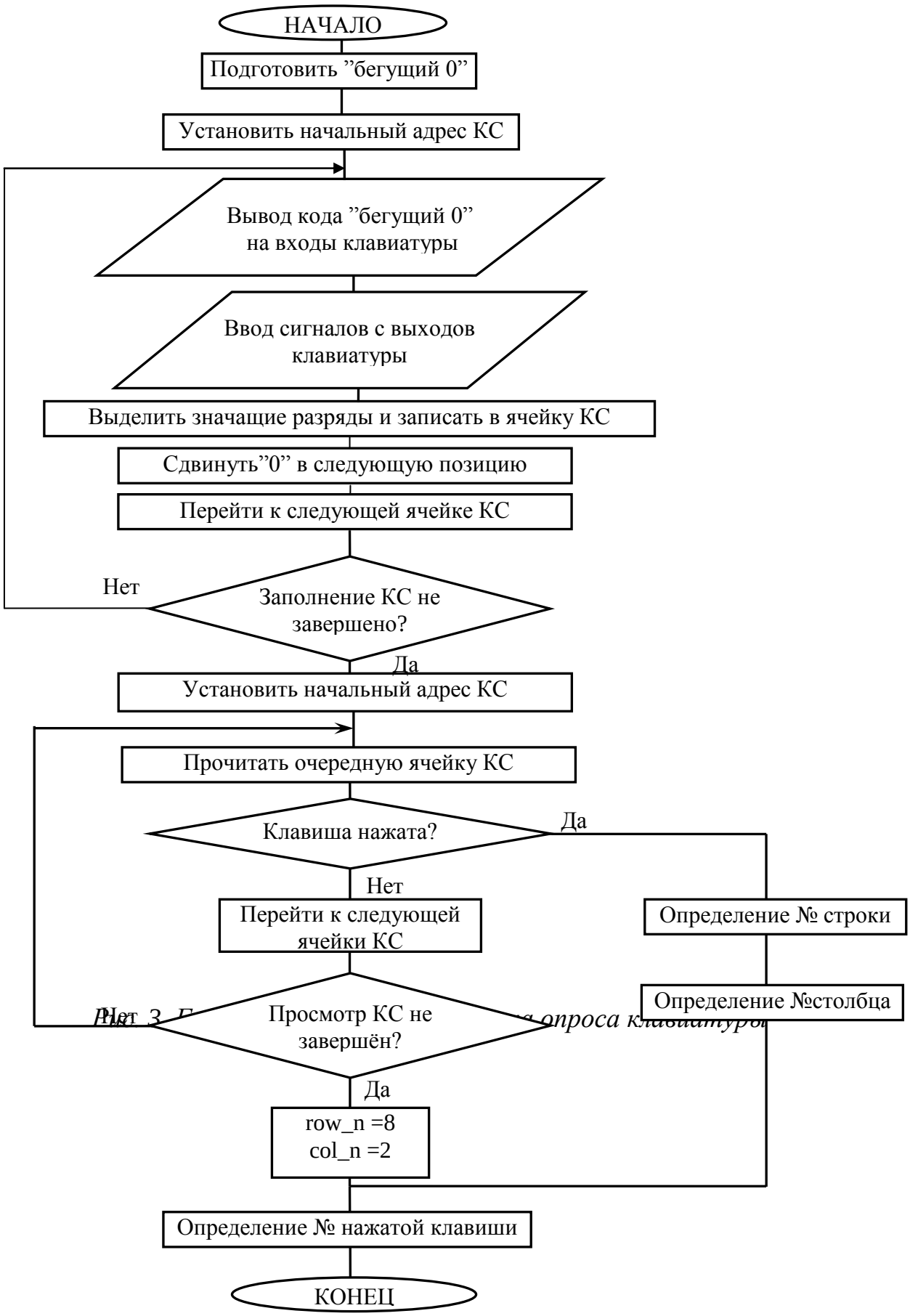

Рис. 3. Г[illegible] опроса клавиатуры

Сначала необходимо организовать вывод на клавиатуру «бегущего нуля».

Так как верхняя строка клавиатуры подключена к старшему разряду порта P2, первое значение, которое нужно выдать в порт P2 для организации «бегущего нуля»: 01111111 (7Fh). Затем необходимо будет производить сдвиг этого числа вправо.

Из рис. 2 видно, что КС будет располагаться в ячейках 30h – 37h, таким образом, начальный адрес КС (map_start) равен 30h.

Текст программы опроса клавиатуры

```
;*****************************************************************
; *
; Filename: ex2.asm
; Date: 2020/05/12
; File Version: 1
; Author: Solov'eva T.N.
; Company: SUAI
; Description: example 2
; *
;*****************************************************************
; Variables
;*****************************************************************
row_n     equ 41h
col_n     equ 42h
N         equ 40h   ;номер нажатой клавиши
map_start equ  30h  ;начало области хранения КС клавиатуры
;*****************************************************************
; Reset Vector
;*****************************************************************
org    0h              ; processor reset vector
    ajmp    start      ; go to beginning of program
;*****************************************************************
; MAIN PROGRAM
;*****************************************************************
org    100h
start:
     ;формирование КС
     ;установка "0" в начальные позиции
     mov  a, #07Fh  ;подготовка "бегущего нуля " (01111111)
     mov  r0, #map_start ;адрес начала карты состояние

opros:    mov  p2, a          ;"бегущего нуля" в порт 2
     setb p1.1      ;настройка разрядов порта P1 на чтение
     setb p1.0
     mov  b, p1          ;чтение
     anl  b, #03h        ;выделение значащих разрядов (у нас 2
     ;младших разряда, поэтому умножаем на
     ;00000011)
     mov  @r0, b         ;записываем стоку карты
```

```
     setb c          ;подготовка нового опроса сдвиг "0"в
     rrc  a          ;следующую позицию
     inc  r0         ;переходим к следующей ячейке КС
     cjne a, #11111111b, opros
     ;b указывает на двоичный код
     ;пока ноль не сдвинется в перенос
     ;дешифрация карты
     mov  r0, #map_start
dc:  mov  a, @r0          ;читаем очередную стоку карты
     cjne a, #03h, dck    ;если в значащих разрядах есть ноль
     ;(нажата клавиша), переходим к dck
     inc  r0         ;если не нажата - просмотр карты далее
     cjne r0, #(map_start+8), dc
     ;пока не закончились строки
     mov  row_n, #8 ;если клавиша не нажата, устанавливаем
     mov  col_n, #2 ;несуществующие значения
     sjmp end1       ;и переходим в конец
dck: mov  a, r0           ;клавиша нажата: в R0 – адрес ячейки
     clr  c
     subb a, #map_start
     ;вычитаем нач. адр. КС, чтобы узнать
     mov  row_n, a  ;номер строки
     mov  a, @r0          ;берем содержимое ячейки КС для
     mov  col_n, #0 ;определения № столбца (сначала № = 0)
dloop1:   rrc  a          ;последовательно сдвигаем вправо, т.к.
     ;значащие разряды - младшие
     jnc  end1       ;пока не ноль вытиснится в перенос
     inc  col_n
     mov  r1, col_n
     cjne r1, #2, dloop1
     ;пока не сдвинем 2 раза
end1:     lcall     get_num        ;вызов подпрограммы опред. номера
     sjmp finish     ;переход на конец программы

get_num:  push a     ;спасаем аккумулятор
     mov  a, row_n
     cjne a, #8, gn_end
     mov  N, #0           ;если row_n = 8, то ничего не нажато
     pop  a
     ret
gn_end:   clr  c
     rlc  a          ;умножаем row_n на два (т.к. 2 столбца)
     add  a, col_n
     inc  a
     mov  N, a
     pop  a
     ret

finish: sjmp $ ;конец программы
end
```

2. Задание по работе

Требуется разработать программу на языке ассемблера MCS-51 для определения положения нажатой клавиши (строка, столбец и номер клавиши) для клавиатуры заданного размера. Порты, через которые к МК подключается клавиатура, указаны в разделе «Варианты заданий».

Работу программы необходимо проверить с помощью симулятора.

3. Порядок выполнения работы

Выполнение работы состоит из двух частей: разработки программы и проверки корректности ее работы с помощью симулятора.

При разработке программы необходимо:

1) составить функциональную схему, отражающую подключение клавиатуры к МК с использованием указанных в задании разрядов;

2) сформировать алгоритм решения задачи;

3) на основании полученного алгоритма составить текст программ на языке ассемблера MCS-51.

Для выполнения работы *на симуляторе MCU 8051 IDE* необходимо выполнить следующие действия.

1. Запустить оболочку симулятора MCU 8051 IDE.
2. Создать новый проект (Project: New).
3. В окно редактора вести текст разработанной программы. Сохранить файл с текстом программы, добавив его в проект. Выполнить компиляцию программы (Tools: Compile).
4. Подключить клавиатуру (Virtual HW: Matrix Keypad) в соответствии с вариантом.
5. Запустить режим симуляции (Simulator: Start/Shutdown). Нажимая различные клавиши клавиатуры, проверить работоспособность программы.

Для выполнения работы *на стенде МК* необходимо выполнить следующие действия.

1. Если после проверки программы на симуляторе планируется проверка на стенде контроллера, необходимо сначала подключить стенд к компьютеру, выполнить необходимые соединения портов МК с клавиатурой в соответствии с разработанной в отчете схемой и включить стенд.

2. Запустить среду Shell51.

3. Открыть вкладку «Текст программы» и ввести в окно редактора текст разработанной программы. *Внимание!* При выполнении работы на стенде следует заменить указанные в задании порты P0, P3 на P4 и P5 соответственно, используя прямую адресацию (адреса портов см. в работе 2).

При оформлении текста программы необходимо руководствоваться следующими правилами:

–первая строка программы должна содержать директиву ассемблеру о начальном адресе программы вида: `org 8100h` (пользовательская программа на стенде выполняется как подпрограмма, диапазон доступных пользователю адресов в памяти программ: 8000h – FFFFh; начальные адреса этой области отведены под пользовательские вектора прерываний; основную программу рекомендуется размещать, начиная с адреса 8100h);

–при объявлении переменных после имени переменной ставят двоеточие;

–каждая последующая строка должна начинаться с метки или со знака табуляции;

–последняя строка программы должна содержать команду `ret`.

4. Сохранить текст программы в виде файла с расширением *.asm.

5. Выполнить трансляцию программы, нажав кнопку «Запуск». Появление в специальном окне слова «Внимание» указывает на наличие ошибки в программе. В этом случае нужно открыть вкладку «Листинги», которая содержит результаты трансляции. Пользуясь информацией на вкладке, необходимо исправить ошибки и повторить трансляцию программы. *Внимание*! При отладке следует игнорировать ошибки компиляции, связанные с адресами портов P4 и P5.

6. Выполнить моделирование с помощью симулятора системы Shell51. С этой целью открыть вкладку «Симулятор» и осуществить загрузку кода программы, нажав кнопку «Загрузка». Затем определить способ запуска: по шагам, до точки останова или целиком всю программу. Наконец, запустить программу, нажав кнопку «Пуск».

7. Проанализировать результаты работы программы по содержимому памяти данных.

8. Убедиться, что стенд правильно подсоединен к компьютеру. С этой целью открыть вкладку «Окна памяти». Для проверки связи следует нажать кнопку «Передать из контролера в ПК». В ходе обмена информацией между компьютером и контроллером поле «Состояние связи» отображает статус линий связи. «Обмен с МК» свидетельствует о наличии очередного сеанса обмена, «Нет связи» – о невозможности взаимодействия с МК, при этом дальнейшие сеансы обмена блокируются. Необходимо выявить и устранить причину, а затем снять блокировку двойным щелчком по сообщению. В большинстве случаев при потере связи требуется завершить работу симулятора, выключить стенд, затем включить его снова и снова запустить симулятор.

9. Занести программу в память программ контроллера, для чего применить кнопку «Загрузка».

10.С помощью кнопки «Запуск» запустить программу.

11.Проверить работу программы при нажатии различных клавиш (результат работы должен отображаться во внутренней памяти данных МК). Для проверки содержимого внутренней памяти данных следует использовать кнопку «Передать из контроллера в ПК».

4. Содержание отчета

Правила оформления отчета приведены в предисловии к практикуму.

Отчет по лабораторной работе должен содержать следующие разделы.

1. Цель работы.
2. Задание по работе.

   Раздел должен включать в себя формулировку задания и данные варианта.
3. Разработка программы.

   В данном разделе необходимо привести схему алгоритма (если она использовалась при разработке программы), текст программы с подробными комментариями, а также скриншоты с результатами выполнения программы.
4. Вывод.

   Пример вывода: «В результате выполнения работы создана программа на языке ассемблера MCS-51 для определения положения нажатой клавиши клавиатуры 4×4. Проверка работоспособности программы произведена в среде *MCU 8051 IDE*. Приобретены навыки организации взаимодействия микроконтроллера с устройствами ввода на примере клавиатуры».

5. Контрольные вопросы

1. С какой целью клавиши объединяются в клавиатуру при подключении их к МК? Подтвердите примером.
2. Каково максимальное количество кнопок, которое нецелесообразно объединять в клавиатуру?
3. С какой целью в подпрограмме определения номера в разделе 1.2.2 используются команды `pop` и `push`?
4. Пусть клавиша одним выводом подключена к разряду 5 порта P1, а второй вывод клавиши заземлен. Как организовать программно ожидание нажатия клавиши?

5. По приведенной в разделе 1.2.2 программе опроса клавиатуры напишите номера клавиш, изображенных на рис. 2.
6. Какие изменения необходимо внести в программу опроса клавиатуры в разделе 1.2.2, если выходы клавиатуры подключить к старшим разрядам порта P1 (см. рис. 2).
7. В программе опроса клавиатуры (раздел 1.2.2) бегущий «0» организован с помощью операции сдвига вправо. Допустим, что используется операция сдвига влево. Определить: a) что нужно изменить в программе; b) номера клавиш, изображенных на рис. 2.
8. Можно ли, используя принцип сканирования клавиатуры, определить положение нескольких одновременно нажатых клавиш?
9. Пусть клавиша одним контактом подключена к P2.2 микроконтроллера, а второй контакт клавиши заземлен. К разряду P3.7 микроконтроллера подключен светодиод. Напишите программу засветки светодиода при нажатии клавиши.
10. Напишите программу, зажигающую зеленый светодиод после набора верного кода на клавиатуре и красный – после набора неверного кода.
11. Напишите программу, определяющую положение нажатой клавиши на клавиатуре путем «быстрого» сканирования: сначала одновременно подается 0 на все строки, и читаются столбцы; затем одновременно подается 0 на все столбцы, и читаются строки.

6. Варианты заданий

Номер варианта	Размер клавиатуры	Порт входа	Порт выхода
1	4×4	P0	P0
2	3×4	P0	P3
3	3×3	P3	P3
4	5×5	P3	P0
5	5×4	P0	P3
6	4×6	P3	P0
7	8×8	P0	P3
8	8×4	P3	P0
9	3×5	P0	P0
10	2×4	P3	P3
11	4×7	P0	P3
12	3×6	P3	P0
13	2×6	P0	P0
14	5×6	P3	P0
15	3×8	P3	P0
16	6×6	P0	P3
17	4×5	P3	P0
18	4×3	P0	P0
19	3×7	P0	P3
20	2×7	P3	P0
21	4×8	P0	P3
22	5×2	P0	P3
23	5×7	P3	P0
24	5×8	P3	P0
25	6×3	P0	P3
26	5×3	P0	P0
27	6×4	P3	P0
28	6×5	P0	P3
29	7×7	P0	P3
30	7×3	P0	P3
31	2×3	P3	P3
32	4×2	P0	P3
33	3×2	P3	P0
34	2×5	P0	P0
35	7×2	P3	P0
36	7×4	P3	P0
37	6×2	P0	P3

Номер варианта	Размер клавиатуры	Порт входа	Порт выхода
38	8×2	P3	P0
39	4×4	P3	P0
40	8×5	P0	P3
41	2×8	P3	P0
42	8×6	P0	P3
43	8×7	P0	P3
44	7×5	P3	P0
45	7×6	P3	P0
46	6×7	P0	P3
47	7×8	P3	P0
48	7×3	P3	P0
49	8×3	P0	P3
50	3×3	P0	P3

Лабораторная работа 5
РАЗРАБОТКА МИКРОКОНТРОЛЛЕРНОЙ СИСТЕМЫ С ИСПОЛЬЗОВАНИЕМ ВНЕШНИХ ПРЕРЫВАНИЙ

Цель работы: изучение принципов работы системы прерываний микроконтроллера; приобретение навыков разработки микроконтроллерных систем, использующих внешние прерывания.

1. Система прерываний микроконтроллеров семейства MCS-51.

Под *прерыванием* подразумевается ситуация, когда внешнее (ВУ) или периферийное устройство (ПУ) формирует специальный сигнал (*запрос прерывания*), поступающий на МК. МК, независимо от того, разрешены ли прерывания, фиксирует запрос установкой флага в специальном регистре (например, TCON, SCON и др.). В каждом машинном цикле МК опрашивает флаги прерываний и в случае, если прерывание не запрещено, приостанавливает выполнение основной программы и передает управление подпрограмме обслуживания прерывания (иногда ее называют *обработчиком прерывания*) от данного источника. Начальный адрес обработчика прерывания называется *вектором прерывания.* После завершения работы обработчика прерывания возобновляется выполнение основной программы.

Система прерываний большинства МК семейства MCS-51 имеет восемь источников прерываний. Прерывания могут вызывать периферийные устройства, размещенные на кристалле МК (таймеры и последовательный порт), а также внешние устройства, подключаемые к выводам INT0 и INT1 МК.

Точки входа в обработчик прерывания (адреса векторов прерываний) для каждого источника прерываний аппаратно зафиксированы (табл. 1). По указанным адресам должны размещаться первые команды обработчиков прерываний.

Таблица 1

	Наименование прерывания	**Источник (флаг)**	**Адрес вектора**	**Бит разрешения**	**Бит приоритета**
1	Внешнее прерывание 0	IE0	0003h	EX0	PX0
2	Прерывание от таймера 0	TF0	000Bh	ET0	PT0
3	Внешнее прерывание 1	IE1	0013h	EX1	PX1
4	Прерывание от таймера 1	TF1	001Bh	ET1	PT1
5	Прерывание от последовательного порта	RI,TI	0023h	ES	PS
6	Прерывание от таймера 2	TF2,EXF2	002Bh	ET2,EXEN2	PT2

Примечание. В *лабораторном стенде* адреса векторов прерываний начинаются не с 0000h, а с адреса 8000h. Это объясняется тем, что на стенде в диапазоне 8000h...FFFFh реализована область пользователя. Младшая половина диапазона адресов для пользователя недоступна и содержит служебные программы.

При написании подпрограммы обработки прерывания (ППОП) следует руководствоваться следующими правилами:

1) ППОП нужно размещать, начиная с адреса соответствующего вектора (например, для того, чтобы при нажатии на кнопку, подключённую к INT0, были выполнены команды, нужно разместить эти команды по адресу 0003h);
2) расстояние между векторами составляет 8 байт, поэтому если ППОП не превышает этот объем, по адресу вектора нужно разметить команду безусловного перехода;
3) если в программе предусмотрены прерывания, основная программа должна размещаться в памяти программ после векторов прерываний;
4) ППОП должна оканчиваться командой RETI.

Прерывания могут быть полностью запрещены сбросом бита EA. Каждое прерывание в отдельности запрещается сбросом соответствующего ему бита разрешения (табл. 1). Расположение битов разрешений зависит от модели МК. Например, для AT89C55WD биты разрешения расположены в регистре IE, за исключением бита EXEN2, расположенного в регистре T2CON.

Регистр разрешения прерываний IE (адрес - 0A8h)

EA	-	ET2	ES	ET1	EX1	ET0	EX0
7	6	5	4	3	2	1	0

Таким образом, в случае, если в некотором машинном цикле (МЦ) МК обнаружит, что некоторый флаг прерывания установлен, МК завершит этот МЦ и перейдет к вызову ППОП по адресу соответствующего вектора, если выполнены все следующие условия:

1) EA = 1;
2) бит разрешения этого прерывания установлен;
3) текущий МЦ является завершающим МЦ при выполнении команды (актуально для команд из 2 и 4 МЦ, т.к. иначе текущая команда не будет выполнена);
4) не выполняется команда RETI или запись в регистры IE или IP;
5) не выполняется обслуживание прерывания от источника с большим или равным приоритетом.

Для источников прерываний, имеющих один флаг, в большинстве МК семейства MCS-51 происходит автоматический сброс флага при переходе по вектору прерывания.

1.1. Приоритеты прерываний

В случае если при обслуживании прерывания МК получает запрос на прерывание, или в случае, если несколько источников запрашивают прерывания одновременно, учитываются приоритеты прерываний.

Если одновременно получены запросы на прерывания от двух источников с разным уровнем приоритета, то обслуживается сначала более высокоприоритетный. Если одновременно получены запросы на прерывания от двух источников с равным уровнем приоритета, то обслуживается сначала первое обнаруженное при опросе флагов. Опрос флагов осуществляется в порядке расположения прерываний в табл. 1 (сверху вниз).

В некоторых моделях микроконтроллеров семейства MCS-51, например, в AT89C55WD предусмотрена возможность программной установки двух уровней приоритетов (0 и 1) прерываний для источников с помощью регистра IP.

Регистр приоритетов прерываний IP (адрес – 0B8h)

-	-	PT2	PS	PT1	PX1	PT0	PX0
7	6	5	4	3	2	1	0

Если такая возможность в микроконтроллере не предусмотрена, все прерывания имеют равные приоритеты.

1.2. Внешние прерывания

Большинство микроконтроллеров семейства MCS-51 имеет два источника внешних прерываний (табл. 3).

Таблица 3

Разряд порта	**Прерывание**	**Флаг**	**Бит разрешения**	**Бит настройки**
P3.2	INT0	IE0	EX0	IT0
P3.3	INT1	IE1	EX1	IT1

Внешние прерывания от выводов INT0 и INT1 могут функционировать в двух режимах (настраиваются битами IT0 и IT1 в регистре TCON соответственно). Если ITx = 0, флаг прерывания устанавливается при обнаружении низкого уровня (0) на выводе INTx в текущем машинном цикле. Если ITx = 1, флаг прерывания устанавливается при обнаружении изменении уровня на выводе INTx из 1 в 0 (задний фронт).

Регистр управления TCON

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

Биты настройки внешних прерываний, а также их флаги расположены в регистре TCON.

1.3. Пример использования внешних прерываний

Рассмотрим простой пример микроконтроллерной системы, зажигающей светодиод при нажатии на кнопку. Пусть кнопка подключена к выводу P3.2 (INT0), светодиод – к выводу P1.0 (рис. 1).

По низкому уровню сигнала на входе INT0 (запуск события) возникает прерывание, обработчик которого зажигает светодиод.

Определим структуру программы, которая должна быть разработана для решения поставленной задачи. Программа должна включать основную программу и подпрограмму обслуживания прерывания.

На *основную программу* возложим выполнение следующих действий:

1) разрешение прерывания INT0;

2) ожидание выхода из программы и выключение светодиода; с этой целью еще одна кнопка соединяется с входом P1.2 и осуществляется программный опрос состояния этого входа; при нажатии кнопки происходит выход из программы.

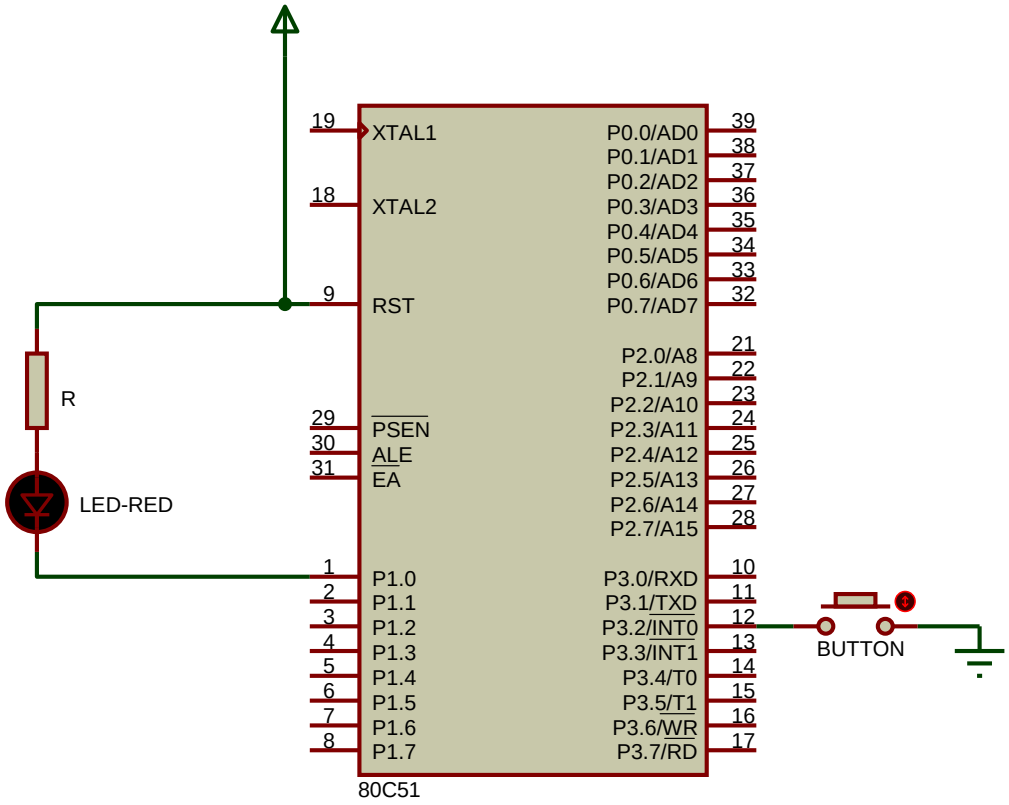

Рис. 1. Схема микроконтроллерной системы

В *подпрограмме обслуживания прерывания* определим следующие действия:

1) запрет прерывания;
2) включение светодиода.

Текст программы

```
;*******************************************************************
; *
; Filename: ex5.asm
; Date: 2020/05/12
; File Version: 1
; Author: Solov'eva T.N.
; Company: SUAI
; Description: example 5
; *
;*******************************************************************
; Reset Vector
;*******************************************************************
org    0h              ; processor reset vector
    ajmp    start      ; go to beginning of program
;*******************************************************************
; Interrupt Vector
;*******************************************************************
org    0003h             ; processor interrupt vector
    ajmp int_0           ; go to int0 interrupt service routine
;*******************************************************************
; MAIN PROGRAM
;*******************************************************************
org    100h
start:
     setb EX0       ;Разрешение прерывания от int0
     setb EA
loop:     setb P1.2 ;Ожидание выхода
     jnb P1.2, finish
     sjmp loop
finish:
     sjmp $         ;конец программы

;Обработчик прерывания INT0
int_0:
     clr EX0   ;запрет INT0
     clr P1.0 ;зажигаем светодиод
     reti

end
```

2. Задание по работе

Требуется разработать микроконтроллерную систему, включающую в себя микроконтроллер семейства MCS-51, ЖКИ, кнопку, а также дополнительные внешние устройства, если они предусмотрены вариантом задания.

При включении системы на дисплей выводится постоянно бегущая строка, содержащая фамилию, имя отчество и номер группы автора программы. Курсор отключен. По нажатию кнопки происходит событие в соответствии с вариантом.

В разделе «Варианты заданий» указано внешнее прерывание, с которым следует связать кнопку. Выводы, через которые к МК требуется подключить ЖКИ, определяются вариантом подключения, указанным в работе, посвященной ЖКИ. Если вариант задания предполагает использование дополнительных внешних устройств, их следует подключить к любым свободным выводам портов МК. Работу системы необходимо проверить с помощью симулятора.

3. Порядок выполнения работы

Выполнение работы состоит из двух частей: разработки программы и проверки корректности ее работы с помощью симулятора.

При разработке программы необходимо:

1) составить функциональную схему, отражающую подключение внешних устройств к МК с использованием указанных в задании выводов;

2) сформировать алгоритм решения задачи;

3) на основании полученного алгоритма составить текст программы на языке ассемблера.

Для выполнения работы *на симуляторе MCU 8051 IDE* необходимо выполнить следующие действия.

1. Запустить оболочку симулятора MCU 8051 IDE.
2. Создать новый проект (Project: New).

3. В окно редактора вести текст разработанной программы. Сохранить файл с текстом программы, добавив его в проект. Выполнить компиляцию программы (Tools: Compile).

4. Подключить внешние устройства (Virtual HW) в соответствии с разработанной схемой.

5. Запустить режим симуляции (Simulator: Start/Shutdown) и проверить работоспособность программы.

4. Содержание отчета

Правила оформления отчета приведены в предисловии к практикуму.

Отчет по лабораторной работе должен содержать следующие разделы.

1. Цель работы.
2. Задание по работе.

Раздел должен включать в себя формулировку задания и данные варианта.

3. Разработка схемы микроконтроллерной системы.

В данном разделе следует привести функциональную схему микроконтроллерной системы, отображающую подключение внешних устройств к микроконтроллеру с указанием выводов. В качестве схемы может быть использован скриншот из симулятора.

4. Разработка программы.

В данном разделе необходимо привести схему алгоритма (если она использовалась при разработке программы), текст программы с подробными комментариями, а также скриншоты с результатами выполнения программы. Для демонстрации результатов следует нажать на кнопку, пока вывод строки на ЖКИ не окончен, и привести два скриншота: до и после нажатия.

5. Вывод.

5. Контрольные вопросы

1. Какие условия необходимы для того, чтобы при обнаружении флага прерывания, оно было обслужено в текущем машинном цикле?
2. Опишите действия, выполняемые МК при обслуживании прерываний.
3. В чем заключаются отличия подпрограммы обработки прерываний от обычной подпрограммы?
4. Во время выполнения команды anl c, 5a, расположенной в памяти программ по адресу 0085h, был обнаружен флаг прерывания от таймера 1. Каким будет значение содержимого счетчика команд и какие данные будут записаны в стек по окончании выполнения команды, если прерывание будет обслужено.
5. Во время выполнения команды anl c, 5a был обнаружен флаг прерывания от таймера 0. Все прерывания разрешены. В каком случае прерывание не будет обслужено по окончании выполнения команды?
6. Если все прерывания разрешены и одновременно поступил запрос от всех внешних источников прерывания, в каком порядке прерывания будут обслужены, если
   a) приоритеты не были установлены;
   b) внешнее прерывание 1 имеет приоритет 1, остальные – приоритет 0;
   c) внешние прерывания 1 и 0 имеют приоритет 1, остальные – приоритет 0.
7. В чем отличие между командами RET и RETI?
8. Напишите программу с использованием прерываний, выводящую на семисегментный индикатор символ 1, если нажата кнопка 1, символ 2, если нажата кнопка 2, символ 3, если нажаты обе кнопки и символ 0, если ни одна кнопка не нажата.

6. Варианты заданий

Номер варианта	Кнопка	Событие
1	INT0	На второй строке выводится номер варианта
2	INT0	ФИО заменяется на ИОФ
3	INT1	В начале первой строки появляется символ *
4	INT1	На семисегментном индикаторе выводится номер

Номер варианта	Кнопка	Событие
		варианта
5	INT0	Символы с первой строки перемещаются на вторую
6	INT0	На семисегментный индикатор выводится число нажатий на кнопку (максимальное число нажатий – 5)
7	INT1	На второй строке выводится номер группы
8	INT1	Стирается первый выведенный символ
9	INT0	На второй строке выводится число нажатий на кнопку
10	INT0	На панели из 8 светодиодов отображается двоичный код числа нажатий на кнопку
11	INT1	На панели из светодиодов отображается двоичный код числа символов, выведенных на первой строке
12	INT1	Изменяется направление движения бегущей строки
13	INT0	Включается курсор в виде мигающего черного квадрата
14	INT0	Включается курсор в виде подчеркивания
15	INT1	ФИО заменяется на Фамилия И.О.
16	INT1	Весь текст на первой строке заменяется на *
17	INT0	На панели из светодиодов отображается двоичный код номера варианта
18	INT0	На семисегментный индикатор выводится последняя цифра номера варианта
19	INT1	К первой строке дописывается символ *
20	INT1	Стирается имя и отчество
21	INT0	Последний выведенный на панели символ удаляется
22	INT0	Загорается/гаснет светодиод
23	INT1	Выключается/включается панель
24	INT1	Изменяется видимость курсора
25	INT0	Стирается имя, отчество и номер группы
26	INT0	Символы с первой строки копируются на вторую
27	INT1	На панели из 8 светодиодов загорается еще один светодиод (максимальное число нажатий – 8)
28	INT1	На панели из 8 светодиодов загорается светодиод, соответствующий числу нажатий (максимальное число нажатий – 8)
29	INT0	Все буквы фамилии, кроме первой, заменяются на *
30	INT0	На второй строке панели выводится символ * под последним символом, выведенным на первой строке
31	INT0	Стирается отчество
32	INT1	На второй строке выводится номер варианта
33	INT1	Первый символ бегущей строки заменяется на *
34	INT0	Курсор в виде мигающего черного квадрата смещается на одну позицию вправо
35	INT0	В ФИО все буквы, кроме первых, заменяются на *

Номер варианта	Кнопка	Событие
36	INT1	На двухразрядном семисегментном индикаторе выводится номер варианта
37	INT1	Курсор в виде подчеркивания смещается на одну позицию влево
38	INT0	Стирается фамилия
39	INT0	На панели из 8 светодиодов гаснет один светодиод
40	INT1	На панели из 16 горящих светодиодов каждый раз гаснет новый светодиод
41	INT1	На семисегментный индикатор выводится число нажатий на кнопку (максимальное число нажатий – 9)
42	INT0	Курсор в виде подчеркивания смещается на одну позицию вправо
43	INT0	Стирается ФИО
44	INT1	ФИО переносится на вторую строку
45	INT1	На панели светодиодов выводится двоичный код номера варианта
46	INT0	Курсор в виде мигающего черного квадрата смещается на одну позицию влево
47	INT0	Стирается номер группы
48	INT1	ФИО копируется на вторую строку
49	INT1	На четырехразрядном семисегментном индикаторе выводится номер группы
50	INT0	Вся вторая строка заполняется символами *

Лабораторная работа 6
РАЗРАБОТКА МИКРОКОНТРОЛЛЕРНОЙ СИСТЕМЫ С ИСПОЛЬЗОВАНИЕМ ТАЙМЕРОВ

Цель работы: изучение принципов работы таймеров и системы прерываний микроконтроллера; приобретение навыков разработки микроконтроллерных систем, использующих таймеры.

1. Таймеры-счетчики микроконтроллеров семейства MCS-51

Большинство микроконтроллеров семейства MCS-51 имеет на своем борту три программируемых 16-битных счетчика-таймера: таймеры-счетчики 0 и 1, обладающие базовым набором функций, а также многофункциональный таймер-счетчик 2 (присутствует не во всех моделях семейства).

Каждый из таймеров-счетчиков представляет собой 16-разрядный синхронный суммирующий последовательный счетчик и имеет свои особенности.

1.1. Устройство таймеров-счетчиков

16-битное содержимое каждого таймера-счетчика представляется в памяти данных МК в области SFR в виде двух регистров (TH0 и TL0 для таймера 0; TH1 и TL1 – для таймера 1; TH2 и TL2 для таймера 2).

Таймеры-счетчики, как следует из их названия, могут работать как счетчики или как таймеры.

При работе в качестве таймера содержимое счетчика инкрементируется в каждом машинном цикле, т.е. через каждые 12 тактовых импульсов. В этом случае источником счетных импульсов является генератор тактовых импульсов микроконтроллера.

При работе в качестве счетчика содержимое счетчика инкрементируется под воздействием заднего фронта (перехода из 1 в 0) внешнего сигнала, подаваемого на соответствующий вывод МК (T0, T1 или T2). Опрос значения внешнего входного сигнала выполняется в каждом машинном цикле. Содержимое счетчика будет увеличено на 1 в том случае, если в предыдущем цикле был считан входной

сигнал высокого уровня (1), а в следующем – сигнал низкого уровня (0). Новое (инкрементированное) значение счетчика будет сформировано в цикле, следующем за тем, в котором был обнаружен переход сигнала из 1 в 0. В этом случае источником счетных импульсов является внешнее устройство.

Переполнение таймера-счетчика вызывает установку флага переполнения TF0(1, 2) и формирует, таким образом, запрос на прерывание.

Для запуска и остановки таймеров служат биты TR0(1, 2).

Управление таймерами-счетчиками 0 и 1 осуществляется с помощью регистра TCON (табл. 1).

Регистр управления TCON

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

Таблица 1

Биты TCON	**Назначение**
TF1, TF0	Флаг переполнения таймера 1 (0). Устанавливается аппаратно при переполнении таймера/счетчика. Сбрасывается при обслуживании прерывания аппаратно.
TR1, TR0	TR1(0) = 1 – таймер 1 (0) запущен; TR1(0) = 0 – таймер 1 (0) остановлен.

Для управления режимами работы таймеров-счетчиков 0 и 1 используется регистра режимов TMOD (табл. 2).

Регистр режимов TMOD

T/C1				T/C0			
Gate	CT	M1	M0	Gate	CT	M1	M0
7	6	5	4	3	2	1	0

Таблица 2

Биты TMOD	**Назначение**
Gate	Управление блокировкой. Если Gate = 1, то T/C0(1) работает до тех пор, пока на входе INT0(1) высокий уровень и бит управления TR0(1) = 1. Если Gate = 0, то T/C0 (1) работает, если бит управления TR0(1) = 1.
CT	Бит выбора режима таймера или счетчика событий. Если C/T = 0, то работает таймер от генератора тактовых импульсов. Если C/T = 1, то работает счетчик от внешних сигналов на входе T0(1).

M1, M0	M1 M0 задает режим работы: 00: Режим 0 01: Режим 1 10: Режим 2 11: Режим 3

Для обоих таймеров режимы работы 0, 1 и 2 одинаковы. Режимы 3 – различны. Рассмотрим кратко работу таймеров-счетчиков во всех четырех режимах.

Управление таймером-счетчиком 2 осуществляется с помощью регистра T2CON (табл. 3).

Регистр управления T2CON

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	CT2	CPRL2
7	6	5	4	3	2	1	0

Таблица 3

Биты T2CON	**Назначение**
TF2	Флаг переполнения таймера 2. Устанавливается аппаратно при переполнении таймера/счетчика (только при RCLK = 0 и TCLK = 0). Не сбрасывается аппаратно при обслуживании прерывания.
EXF2	Флаг внешнего прерывания по спаду сигнала на выводе T2EX. Не сбрасывается аппаратно при обслуживании прерывания.
RCLK	Установка режима генератора скорости приема для последовательного порта.
TCLK	Установка режима генератора скорости передачи для последовательного порта.
EXEN2	Бит разрешения прерывания по спаду сигнала на выводе T2EX.
TR2	TR2 = 1 – таймер 2 запущен; TR2 = 0 – таймер 2 остановлен.
CT2	Бит выбора режима таймера или счетчика событий. Если CT = 0, то работает таймер от генератора тактовых импульсов. Если CT = 1, то работает счетчик от внешних сигналов на входе T2.
CPRL2	Установка режима захвата (CPRL2 = 1) или автоматической перезагрузки (CPRL2 = 0).

Рассмотрим режимы работы таймеров-счетчиков более подробно.

1.2. Режимы работы таймеров-счетчиков 0 и 1

Режим 0. Перевод любого таймера-счетчика в режим 0 превращает его из 16-разрядного в 13-разрядный. Этот режим носит название «8-битный таймер-счетчик с предделителем на 32», так как 8 старших разрядов 13-разрядного содержимого счетчика отображаются в регистре THx (x = 0 или 1), а 5 младших

разрядов – в 5 младших разрядах регистра TLx. 3 старших бита TLx в этом режиме не используются.

Режим 1. Режим 16-разрядного таймера-счетчика.

Режим 2. Режим 8-разрядного таймера-счетчика с перезагрузкой. В режиме 2 таймер-счетчик имеет разрядность 8. Содержимое 8-битного счетчика отражается в регистре TLx. Переполнение счетчика приводит не только к установке флага, но и автоматически перезагружает в TLx содержимое THx. Содержимое THx в режиме 2 не изменяется.

Режим 3. В режиме 3 таймеры 0 и 1 работают по-разному. Таймер-счетчик 1 сохраняет неизменным свое текущее содержимое. Таймер 0 преобразуется в два независимых 8-разрядных счетчика. Содержимое первого счетчика отображается в регистре TL0. Работа этого счетчика аналогична работе таймера 0 в режиме 1, за исключением того, что счетчик 8-разрядный. Для работы с этим счетчиком используются биты таймера 0 (C/T, Gate, TR0, TF0) и входной сигнал INT0. Содержимое второго счетчика отображается в регистре TH0. Второй счетчик может выполнять только функцию таймера (т.е. подсчет машинных циклов МК). Для работы с этим счетчиком используются биты таймера 1 (только TR1 и TF1). Для включения второго счетчика используется бит TR1, а переполнение этого счетчика приводит к установке флага TF1.

1.3. Режимы работы таймера-счетчика 2

Таймер-счетчик 2 может работать в одном из трех режимов. Выбор режима осуществляется с помощью битов регистра T2CON (табл. 4).

Таблица 4

RCLK+TCLK	**CPRL**	**Реим**
0	0	16-битный таймер-счетчик с автоматической перезагрузкой
0	1	16-битный режим захвата
1	-	Генератор скорости последовательного порта

Режим захвата. При EXEN2 = 1 спад сигнала на выводе T2EX инициирует запись содержимого таймера (т.е. текущего значения регистров TH2 и TL2) в 16-разрядный регистр RCAP2 (т.е. в регистры RCAP2H и RCAP2L соответственно).

Режим автоматической перезагрузки. При EXEN2 = 0 переполнение таймера инициирует загрузку в него (т.е. в регистры TH2 и TL2) содержимого 16-разрядного регистра RCAP2 (т.е. регистров RCAP2H и RCAP2L соответственно).

При EXEN2 = 1 загрузка содержимого RCAP2 в таймер происходит не только при переполнении таймера, но и при спаде сигнала на выводе T2EX.

Режим генератора скорости последовательного порта. Будет рассмотрен в описании следующей работы.

1.4. Пример применения таймеров-счетчиков 0 и 1

Рассмотрим применение таймеров для создания микроконтроллерной системы «Секундомер» (рис. 1). Система состоит из микроконтроллера, кнопки «Пуск», кнопки «Стоп» и дисплея, на котором будет выводиться измеренное время.

Для измерения времени воспользуемся таймером 0. Таймер имеет возможность блокировки счета при подаче логического нуля на вход INT0, поэтому подключим кнопку «Стоп» к выводу INT0.

Кнопку «Пуск» подключим к выводу INT1.

Для удобства отображения информации на ЖКИ создадим переменные hmks, ms, dms и hms для хранения десятичных разрядов времени в миллисекундах (десятых долей, единиц, десятков и сотен миллисекунд соответственно). Для реализации этой идеи таймер 0 необходимо настроить так, чтобы он переполнялся каждые 100 мкс. С этой целью будем использовать режим 2 таймера 0 (режим с перезагрузкой), и значение TH0 = 156 для перезагрузки.

Максимальное значение, измеряемое системой «Секундомер» будет составлять 1 с (поэтому правильнее было бы назвать нашу систему – «Миллисекундомер»).

Программа для системы «Секундомер» будет включать в себя основную программу и обработчики прерываний от трех источников: кнопки «Пуск» (INT1), кнопки «Стоп» (INT0) и таймера 0.

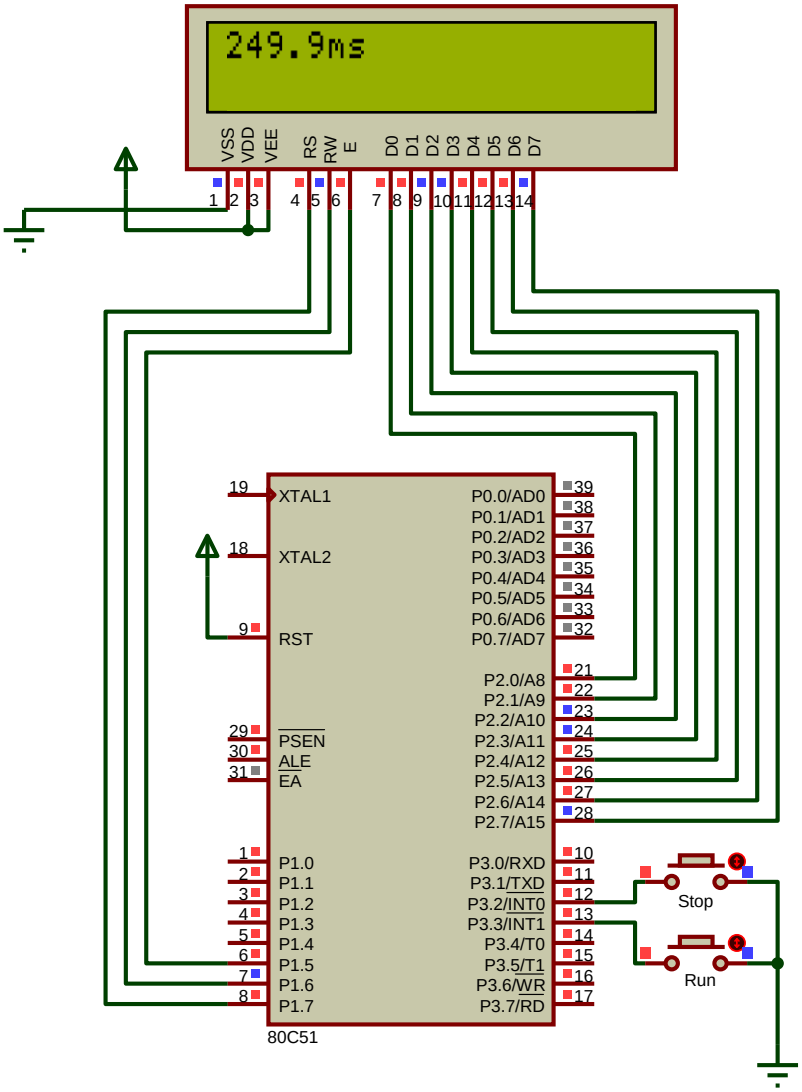

Рис. 1. Схема микроконтроллерной системы «Секундомер»

На *основную программу* возложим выполнение следующих действий:

1) настройка ЖКИ;

2) настройка режима 2 таймера 0 с разрешением GATE;

3) разрешение прерывания INT1.

В *подпрограмме обслуживания прерывания от кнопки «Пуск» (INT1)* необходимо выполнить следующие действия:

1) установка начальных значений содержимого таймера и переменных hmks, ms, dms и hms;

2) включение таймера;

3) запрет прерываний INT1;

4) разрешение прерываний INT0.

В *подпрограмме обслуживания прерывания от таймера 0* будем производить подсчет сотен микросекунд, измененяя значения переменных hmks, ms, dms и hms.

В *подпрограмме обслуживания прерывания от кнопки «Стоп» (INT0)* определим следующие действия:

1) вывод информации об измеренном времени на ЖКИ;
2) запрет прерываний INT0;
3) разрешение прерываний INT1.

Текст программы для системы «Секундомер»

```
;*****************************************************************
; *
; Filename: ex6.asm
; Date: 2021/01/22
; File Version: 1
; Author: Solov'eva T.N.
; Company: SUAI
; Description: example 6
; *
;*****************************************************************
; Variables
;*****************************************************************
switch    equ  43h  ;переключатель «команда-данные» (RS)
bte  equ  44h  ;выдаваемый на ЖКИ байт
hmks equ  45h  ;сотни мкс
ms   equ  46h  ;единицы мс
dms  equ  47h  ;десятки мс
hms  equ  48h  ;сотни мс
;*****************************************************************
; Reset Vector
;*****************************************************************
org    0h              ; processor reset vector
    ajmp    start      ; go to beginning of program
;*****************************************************************
; Interrupt Vectors
;*****************************************************************
org    0003h             ; processor interrupt vector
    ajmp int_0           ; go to int0 interrupt service routine
org    0013h             ; processor interrupt vector
    ajmp int_1           ; go to int1 interrupt service routine
org    000bh             ; processor interrupt vector
    ajmp tim_0           ; go to timer 0 interrupt service routine
;*****************************************************************
; MAIN PROGRAM
;*****************************************************************
org    100h
start:
```

```
     ;инициализация ЖКИ
     mov  switch, #0;переключатель уст-ть на команду (RS=0)
     mov  bte, #38h ;настройка строк
     lcall     indic_wr  ;вызов подпрограммы передачи в ЖКИ
     mov  bte, #0ch ;активация всех знакомест
     lcall     indic_wr
     mov  bte, #06h ;режим автом. перемещения курсора
     lcall     indic_wr

     ;инициализация таймера
     mov  TMOD,#00001010b ;таймер 0 в режиме 2 с разрешением int0
     setb EX1       ;разрешение int1
     setb EA

finish: sjmp $ ;конец программы

indic_wr: mov  p2, bte   ;передаваемый байт – в P2
     setb p1.5      ;E:=1
     clr  p1.6      ;RW:=0 (запись)
     mov  a, switch
     mov  c, acc.0  ;нам нужен 0-ой бит аккумулятора
     mov  p1.7, c        ;RS:=switch (команда/данные)
     lcall     indic_delay
     clr  p1.5      ;E:=0
     lcall     indic_delay
     setb p1.5      ;E:=1
     ret

indic_delay:   nop
     nop
     nop
     nop
     nop
     nop
     nop
     nop
     ret

     ;ППОП int_1(нажатие кнопки = запуск таймера)
int_1:
     mov  TH0,#156  ;считаем сотни мкс
     mov  TL0,#156
     mov  hmks,#0
     mov  ms,#0
     mov  dms,#0
     mov  hms,#0
     setb ET0       ;разрешение прерываний от таймера
     setb EX0       ;разрешение прерываний от int0
     mov  TCON,#00010101b     ;включаем таймер прерывания по фронту
     clr  EX1       ;запрет прерываний от int1
     reti
```

```
     ;ППОП таймера 0 (переполнение таймера, прошло 100 мкс)
     ;максимальное время 1с
tim_0:
     inc  hmks      ;десятые доли мс
     mov  a,hmks
     cjne      a,#10,exit     ;не прошла 1 мс
     mov  hmks,#0
     inc  ms        ;единицы мс
     mov  a,ms
     cjne      a,#10,exit     ;не прошло 10 мс
     mov  ms,#0
     inc  dms       ;десятки мс
     mov  a,dms
     cjne      a,#10,exit     ;не прошло 100 мс
     mov  dms,#0
     inc  hms       ;сотни мс
     mov  a,hms
     cjne      a,#10,exit     ;не прошла 1 с
     mov  hms,#0
exit:     reti

     ;ППОП int_0(нажатие кнопки = конец счета)
int_0:
     mov  switch, #0;переключатель уст-ть на команду (RS=0)
     mov  bte, #80h ;установка адреса первого символа
     lcall     indic_wr
     mov  switch, #1     ;переключатель уст-ть на данные (RS=1)
     mov  a,hms
     add  a,#30h
     mov  bte,a
     lcall     indic_wr
     mov  a,dms
     add  a,#30h
     mov  bte,a
     lcall     indic_wr
     mov  a,ms
     add  a,#30h
     mov  bte,a
     lcall     indic_wr
     mov  bte,#'.'
     lcall     indic_wr
     mov  a,hmks
     add  a,#30h
     mov  bte,a
     lcall     indic_wr
     mov  bte,#'m'
     lcall     indic_wr
     mov  bte,#'s'
     lcall     indic_wr
     clr  EX0  ;для нового старта
     setb EX1
     reti
end
```

2. Задание по работе

Требуется разработать микроконтроллерную систему «Миллисекундомер», включающую в себя микроконтроллер семейства MCS-51, ЖКИ и одну или две кнопки (в зависимости от варианта).

При включении системы на первой строке ЖКИ выводится ФИО автора работы. При наступлении события «Старт» запускается миллисекундомер, при этом на второй строке появляется курсор в виде мигающего черного прямоугольника. При наступлении события «Стоп» останавливается миллисекундомер, при этом на второй строке гаснет курсор и выводится измеренное время с точностью до микросекунд.

В разделе «Варианты заданий» указан таймер, который необходимо использовать, максимальное время, на которое должен быть рассчитан миллисекундомер, и внешние прерывания, с которыми нужно связать события «Старт» и «Стоп». Выводы, через которые к МК требуется подключить ЖКИ, определяются вариантом подключения, указанным в работе 3. Работу системы необходимо проверить с помощью симулятора.

3. Порядок выполнения работы

Выполнение работы состоит из двух частей: разработки программы и проверки корректности ее работы с помощью симулятора.

При разработке программы необходимо:

1) составить функциональную схему, отражающую подключение внешних устройств к МК;

2) сформировать алгоритм решения задачи;

3) на основании полученного алгоритма составить текст программы на языке ассемблера.

Для выполнения работы *на симуляторе MCU 8051 IDE* необходимо выполнить следующие действия.

1. Запустить оболочку симулятора MCU 8051 IDE.
2. Создать новый проект (Project: New).
3. В окно редактора вести текст разработанной программы. Сохранить файл с текстом программы, добавив его в проект. Выполнить компиляцию программы (Tools: Compile).
4. Подключить внешние устройства (Virtual HW) в соответствии с разработанной схемой.
5. Запустить режим симуляции (Simulator: Start/Shutdown) и проверить работоспособность программы.

4. Содержание отчета

Правила оформления отчета приведены в предисловии к практикуму.

Отчет по лабораторной работе должен содержать следующие разделы.

1. Цель работы.
2. Задание по работе.

Раздел должен включать в себя формулировку задания и данные варианта.

3. Разработка схемы микроконтроллерной системы.

В данном разделе следует привести функциональную схему микроконтроллерной системы, отображающую подключение внешних устройств к микроконтроллеру с указанием выводов. В качестве схемы может быть использован скриншот из симулятора.

4. Разработка программы.

В данном разделе необходимо привести схему алгоритма (если она использовалась при разработке программы), текст программы с подробными комментариями, а также скриншоты с результатами выполнения программы.

5. Вывод.

5. Контрольные вопросы

1. Расскажите о четырех режимах работы таймеров-счетчиков.

2. Какие регистры используются для управления таймерами-счетчиками 0 и 1?

3. Почему в режиме 3 таймер 1 не изменяет своего содержимого?

4. В чем отличие между командами RET и RETI?

5. Организуйте работу системы с использованием режима 1 таймера.

6. Разработайте микроконтроллерную систему, зажигающую светодиод через 10 нажатий на кнопку. При разработке используйте возможности таймеров микроконтроллера.

7. Разработайте микроконтроллерную систему, выводящую на ЖКИ символ «*» через каждые 5 нажатий на кнопку. При разработке используйте возможности таймеров микроконтроллера.

8. Добавьте в разработанную вами микроконтроллерную систему семисегментный индикатор, на котором после нажатия кнопки «Пуск» отображаются миллисекунды периода мигания.

9. Добавьте в разработанную вами микроконтроллерную систему светодиоды по числу миллисекунд в периоде мигания. После нажатия кнопки «Пуск» с истечением каждой миллисекунды в периоде должен зажигаться новый светодиод.

10. Напишите подпрограмму задержки

    a. на 40 мкс;

    b. на 100 мс.

6. Варианты заданий

Номер варианта	Таймер	T_{max}, мс	Старт	Стоп
1	0	20	INT0	INT0
2	1	8	INT0	INT1
3	2	15	INT1	INT0
4	0	20	INT1	INT1
5	1	10	INT0	INT0
6	2	5	INT0	INT1
7	0	28	INT1	INT0
8	1	4	INT1	INT1
9	2	6	INT0	INT0
10	0	18	INT0	INT1
11	1	19	INT1	INT0
12	2	30	INT1	INT1
13	0	12	INT0	INT0
14	1	7	INT0	INT1
15	2	6	INT1	INT0
16	0	12	INT1	INT1
17	1	7	INT0	INT0
18	2	8	INT0	INT1
19	0	10	INT1	INT0
20	1	9	INT1	INT1
21	2	29	INT0	INT0
22	0	19	INT0	INT1
23	1	11	INT1	INT0
24	2	20	INT1	INT1
25	0	16	INT0	INT0
26	1	21	INT0	INT1
27	2	26	INT1	INT0
28	0	3	INT1	INT1
29	1	17	INT0	INT0
30	2	2	INT0	INT1
31	0	21	INT1	INT0
32	1	14	INT1	INT1
33	2	22	INT0	INT0
34	0	11	INT0	INT1
35	1	25	INT1	INT0
36	2	15	INT1	INT1
37	0	10	INT0	INT0
38	1	22	INT0	INT1
39	2	17	INT1	INT0

Номер варианта	Таймер	T_{max}, мс	Старт	Стоп
40	0	23	INT1	INT1
41	1	27	INT0	INT0
42	2	24	INT0	INT1
43	0	15	INT1	INT0
44	1	17	INT1	INT1
45	2	24	INT0	INT0
46	0	12	INT0	INT1
47	1	14	INT1	INT0
48	2	23	INT1	INT1
49	0	13	INT0	INT0
50	1	16	INT0	INT1

Лабораторная работа 7
РАЗРАБОТКА МИКРОКОНТРОЛЛЕРНОЙ СИСТЕМЫ С ИСПОЛЬЗОВАНИЕМ ПОСЛЕДОВАТЕЛЬНЫХ ИНТЕРФЕЙСОВ

Цель работы: изучение принципов последовательной передачи данных; приобретение навыков разработки микроконтроллерных систем, использующих последовательные интерфейсы.

1. Последовательный порт микроконтроллеров семейства MCS-51

Последовательный порт микроконтроллеров MCS-51 позволяет производить одновременный прием и передачу данных. Прием и передача данных в большинстве режимов работы последовательного порта осуществляется через выводы микроконтроллера RxD (P3.0) и TxD (P3.1) соответственно. Прием и передача производится, начиная с младшего бита (LSB first).

Доступ к сдвиговым регистрам приема и передачи последовательного пота осуществляется с помощью специального регистра SBUF. Запись данных в SBUF загружает данные в регистр передачи, а при чтении данных из SBUF чтение происходит из регистра приема.

Управление последовательным портом производится с помощью регистра SCON (табл. 1).

Регистр управления SCON

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

Таблица 1

Биты	**Назначение**				
SM0, SM1	SM0	SM1	Режим	Формат	Частота бит
	0	0	0	Sync. shift register	$F_{osc}/12$
	0	1	1	8-bit UART	настраиваемая
	1	0	2	9-bit UART	$F_{osc}/64$ или $F_{osc}/32$
	1	1	3		настраиваемая
SM2	Разрешает мультипроцессорный обмен в режимах 2 и 3.				
REN	Бит разрешения приема.				
TB8	9ый бит передачи в режимах 2 и 3				
RB8	9ый бит приема в режимах 2 и 3				
TI	флаг прерывания по окончанию передачи пакета				
RI	флаг прерывания по окончанию приема пакета				

Как видно из табл. 1, последовательный порт может работать в одном из четырех режимов.

1.1. Режимы работы последовательного порта

Режим 0. Synchronous shift register. Данные передаются и принимаются через вывод RxD. Вывод TxD используется для выдачи синхросигнала – частоты сдвига. Передача и прием осуществляется пакетами по 8 бит (LSB first) с частотой сдвига, равной 1/12 частоты микроконтроллера.

Режим 1. 8-bit UART. Передача (через TxD) и прем (через RxD) осуществляется пакетами по 10 бит: старт-бит (0), 8 бит данных (LSB first) и стоп-бит (l). При приеме стоп-бит загружается в бит RB8 регистра SCON. Частота бит передачи настраиваемая.

Режим 2. 9-bit UART. Передача (через TxD) и прем (через RxD) осуществляется пакетами по 11 бит: старт-бит (0), 8 бит данных (LSB first), 9ый бит данных и стоп-бит (l). При передаче 9ый бит загружается из бита TB8 регистра SCON. При приеме 9ый бит загружается в бит RB8 регистра SCON. Частота бит настраиваемая и составляет 1/32 или 1/64 частоты микроконтроллера.

Режим 3. 9-bit UART. Передача (через TxD) и прем (через RxD) осуществляется пакетами по 11 бит: старт-бит (0), 8 бит данных (LSB first), 9ый бит данных и стоп-бит (l). Режим 3 отличается от режима 2 лишь более широкой возможностью настройки частоты бит.

Во всех четырех режимах по окончании передачи пакета устанавливается флаг TI, по окончании приема – флаг RI. Запрос прерывания, формируемый установкой этих флагов, вызывает переход по одному и тому же вектору прерывания, в связи с этим аппаратного сброса флагов не происходит, флаги необходимо сбрасывать программно.

Во всех четырех режимах передача запускается при выполнении любой команды, использующей SBUF в качестве операнда-приемника.

Прием в режиме 0 инициализируется сбросом флага RI и установкой бита REN. Прием в других режимах инициализируется установкой REN и поступлением старт-бита на вход RxD.

1.2. Скорость работы последовательного порта

Скорость (частота бит, baud rate) последовательного порта в большинстве режимов настраиваемая. Способ настройки зависит от режима.

Скорость в *режиме 0* фиксированная и составляет

$$Baud\ Rate_{Mode0} = \frac{F_{osc}}{12},$$

где F_{ocs} – частота микроконтроллера.

Скорость в *режиме 2* зависит от значения бита SMOD, расположенного в регистре PCON. Если SMOD = 0, скорость составляет 1/64 частоты микроконтроллера, если SMOD = 1, скорость составляет 1/32 частоты микроконтроллера:

$$Baud\ Rate_{Mode2} = 2^{SMOD}\frac{F_{osc}}{64}.$$

Скорость в *режимах 1 и 3* определяется одинаково и зависит от частоты переполнения таймера 1, таймера 2 или обоих таймеров одновременно. В последнем случае один из таймеров определяет скорость приема, а другой – скорость передачи. Очевидно, использование таймера 2 для задания скорости невозможно, если он отсутствует в микроконтроллере.

При использовании таймера 1 скорость определяется следующим образом:

$$Baud\ Rate_{Mode1,3} = \frac{2^{SMOD}}{32}\mathrm{T1}_{overflow},$$

где $T1_{overflow}$ – частота переполнения таймера 1.

В случае если таймер 1 работает в режиме 2 (режим с автоматической перезагрузкой), скорость последовательного порта определяется как

$$Baud\ Rate_{Mode1,3} = \frac{2^{SMOD}}{32}\cdot\frac{F_{osc}}{12\left(256-\mathrm{TH1}\right)}.$$

Таймер 2 может быть использован для определения скорости передачи и/или приема данных последовательным портом при установке в регистре T2CON битов

TCLK и RCLK соответственно. Установка хотя бы одного из этих битов переводит таймер 2 в режим генератора скорости последовательного порта.

$$Baud\ Rate_{Mode1,3} = \frac{\mathrm{T2}_{overflow}}{16}.$$

Работа таймера 2 в этом режиме отличается от работы в режиме с автоматической перезагрузкой тем, что инкремент значения таймера осуществляется с частотой Fosc/2 вместо обычной Fosc/12. В связи с этим скорость последовательного порта будет равна

$$Baud\ Rate_{Mode1,3} = \frac{F_{osc}}{32\left(65536 - \mathrm{RCAP2}\right)}. \quad (1)$$

Отметим еще два отличия режима генератора скорости последовательного порта от режима с автоматической перезагрузкой для Таймера 2. В режиме генератора скорости при переполнении таймера не устанавливается флаг TF2 и, соответственно, не формируется запроса на прерывание. Кроме того, отсутствует возможность перезагрузки по спаду сигнала на входе T2EX.

1.3. Пример применения последовательного порта

Рассмотрим применение последовательного порта для создания микроконтроллерной системы «Терминал» (рис. 1). Система состоит из микроконтроллера и виртуального терминала, на котором будет выводиться строка «HELLO!». Обмен данными с терминалом будем осуществлять в формате 8-bit UART со скоростью 960 бит/с.

Для осуществления обмена с заданной скоростью будем использовать таймер 2. При этом необходимо рассчитать значения RCAP2 и частоты микроконтроллера.

Сначала проведем расчеты RCAP2 для стандартной частоты 12 МГц. Из (1) следует:

$$\mathrm{RCAP2} = 65536 - \frac{F_{osc}}{32\,Baud\ Rate} = 65536 - \frac{12 \cdot 10^6}{32 \cdot 960} = 65536 - 390,625.$$

Значение RCAP2 не является целым числом, в связи с этим необходимо изменить значение частоты микроконтроллера.

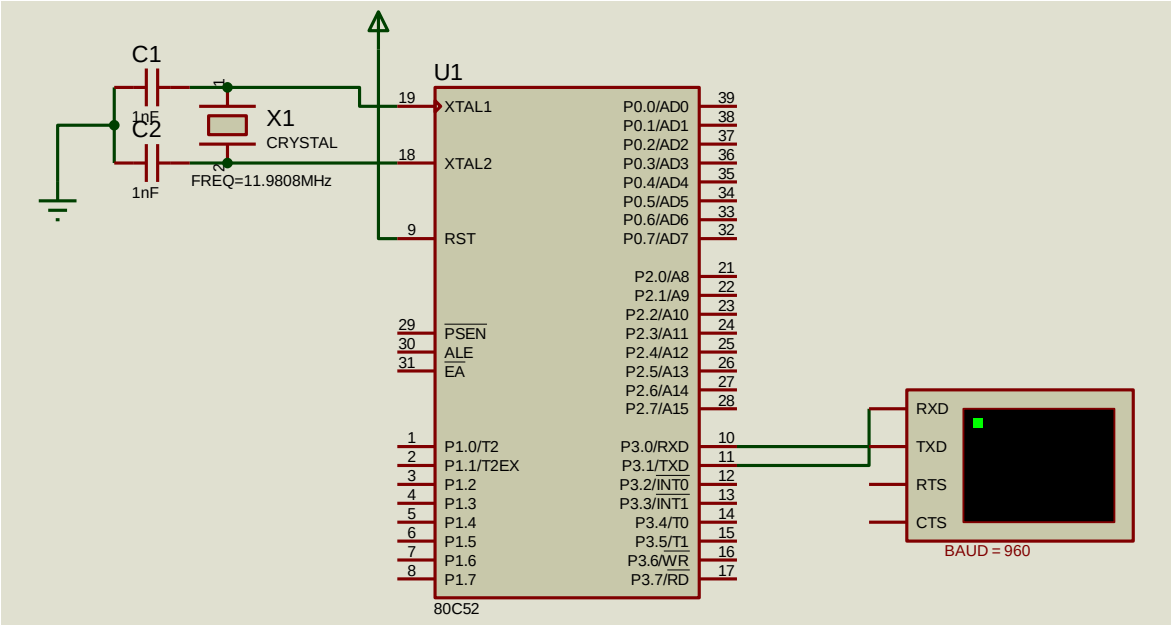

Рис. 1. Схема микроконтроллерной системы «Терминал»

Рассчитаем значение частоты для RCAP2 = 65536 – 390 = 65146 = FE7Ah. Из (1) получим:

$$F_{osc} = 32\left(65536 - \text{RCAP2}\right) Baud\ Rate_{Mode1,3} = 32 \cdot 390 \cdot 960 = 11980800.$$

Для установки частоты микроконтроллера необходимо подключить к нему внешний кварцевый резонатор (рис. 1).

Прием и передача данных может быть организована как с использованием прерываний, так и без них. Рассмотрим первый вариант. В этом случае программа для системы «Терминал» будет включать в себя основную программу и обработчик прерываний от последовательного порта.

На *основную программу* возложим выполнение следующих действий:

1) настройка последовательного пота;
2) разрешение прерывания от последовательного порта;
3) настройка и запуск таймера 2;
4) запуск передачи первого символа.

В *подпрограмме обслуживания прерывания от последовательного пота* необходимо выполнить следующие действия:

1) сброс флага TI;
2) проверка окончания строки;
3) выдача очередного символа.

Текст программы для системы «Терминал»

```
;===================================================================
; Main.asm file generated by New Project wizard
; Created:   Ср янв 24 2024
; Processor: 80C52
; Compiler:  ASEM-51 (Proteus)
;===================================================================
$NOMOD51
$INCLUDE (80C52.MCU)
;===================================================================
; RESET and INTERRUPT VECTORS
;===================================================================
      ; Reset Vector
      org   0000h
      jmp   Start

      ; Interrupt Vector
      org 23h
     CLR TI; сброс флага прерывания
     INC DPTR
     CLR A
     MOVC A,@A+DPTR; чтение из памяти очередного символа
     CJNE A,#0h, hold; если не конец строки
finish:   CLR ES ; запрет прерываний от послед. порта
     CLR EA
     CLR TR2; выключаем таймер
     RETI
hold:     MOV SBUF,A; запуск передачи
     RETI
;===================================================================
; CODE SEGMENT
;===================================================================
      org   0100h
Start:
      MOV SCON,#01000000b; Послед. порт в режиме 1
      MOV RCAP2H,#0FEh; Скорость приема 960
      MOV RCAP2L,#7Ah
      SETB TCLK; Таймер 2 в режиме генератора скорости передачи
      SETB TR2; Запуск таймера
      SETB ES; разрешение прерываний от послед. порта
      SETB EA
      MOV DPTR,#200h
      CLR A
      MOVC A,@A+DPTR; чтение из памяти первого символа
      MOV SBUF,A; запуск передачи
Loop:
      jmp Loop

     org 200h; строка символов
db 'HELLO!',0h;
;===================================================================
      END
```

2. Задание по работе

Требуется разработать микроконтроллерную систему «Калькулятор», включающую в себя микроконтроллер семейства MCS-51 и виртуальный терминал (рис. 1).

При включении системы на экране терминала выводится ФИО автора работы, после чего система переходит в состояние ожидания ввода данных (например, 15*3=). После ввода пользователем символа «=» система выводит результат или сообщение об ошибке, если таковая обнаружена.

Обмен данными с виртуальным терминалом осуществляется в формате 8-bit UART. Скорость обмена данными, а также операция, выполняемая калькулятором, указаны в разделе «Варианты заданий». Операнды являются целыми однобайтными числами без знака.

Разработка и моделирование системы производится в САПР Proteus.

3. Порядок выполнения работы

Выполнение работы состоит из двух частей: разработки программы и моделирования работы системы в САПР *Proteus*.

При разработке программы необходимо:

1) рассчитать необходимые значения частоты микроконтроллера, регистров последовательного порта и таймера для осуществления обмена данными с заданной скоростью;
2) сформулировать алгоритм решения задачи;
3) на основании полученного алгоритма составить текст программы на языке ассемблера MCS-51.

При моделировании работы программы в системе *Proteus* необходимо выполнить следующие действия.

1. Запустить *Proteus*.

2. Запустить мастер создания проектов (*File: New Project*). В открывшемся окне мастера ввести имя проекта и путь к нему.

3. На следующем шаге создания проекта следует выбрать пункт *Create a schematic from the selected template*. Далее рекомендуется выбрать пункт *Do not create a PCB layout*.

4. На следующем шаге следует выбрать пункт *Create Firmware Project* и в выпадающем меню выбрать любой микроконтроллер семейства 8051 (рекомендуемый микроконтроллер: *80C52*), компилятор *ASEM-51* (*Proteus*). Рекомендуется оставить галочку *Create Quick Start Files*, в этом случае по завершении работы мастера вместе с окном схемы проекта (*Schematic Capture*) откроется и окно, содержащее шаблон файла программы для микроконтроллера (*Source Code*).

5. В окне схемы (*Schematic Capture*) необходимо подключить к микроконтроллеру виртуальный терминал. Для этого следует щелкнуть правой кнопкой мыши на свободном участке поля и воспользоваться меню *Place: Virtual Instrument: VIRTUAL TERMINAL*. Терминал необходимо расположить на схеме и соединить с микроконтроллером (рис. 1).

6. Далее следует выполнить настройку микроконтроллера и терминала. Для этого в контекстном меню микроконтроллера нужно выбрать пункт *Edit Properties* и установить частоту (*Clock Frequency*). В контекстном меню терминала нужно выбрать пункт *Edit Properties* и установить скорость обмена данными (*Baud Rate*) в соответствии с вариантом.

7. Во вкладке *Source Code* необходимо набрать текст программы. Вкладку *Source Code* можно открыть, щелкнув правой кнопкой мыши по микроконтроллеру на схеме и выбрав пункт *Source Code*.

8. Запуск моделирования осуществляется нажатием синей стрелки в левом нижнем углу окна. При этом происходит сохранение и компиляция проекта. При наличии ошибок компиляции сообщение о них отображается в нижней части окна.

9. После успешного запуска проекта отроется окно виртуального терминала, воспользовавшись которым необходимо проверить корректность работы системы. После закрытия окна терминала, его снова можно открыть, воспользовавшись меню *Debug: Virtual Terminal*. Вводимые пользователем символы по умолчанию не отображаются в окне терминала. Включить отображение символов можно, воспользовавшись контекстным меню окна терминала (*Echo Typed Characters*).

10. При необходимости для отладки программы можно воспользоваться режимом отладки (*Debug: Start VSM Debugging*). Для наблюдения за состоянием внутренней памяти микроконтроллера необходимо открыть соответствующие окна (*Debug: 8051 CPU: Registers, Internal (IDATA) Memory*).

4. Содержание отчета

Правила оформления отчета приведены в предисловии к практикуму.

Отчет по лабораторной работе должен содержать следующие разделы.

1. Цель работы.
2. Задание по работе.

Раздел должен включать в себя формулировку задания и данные варианта.

3. Разработка схемы микроконтроллерной системы.

В данном разделе следует привести функциональную схему микроконтроллерной системы, отображающую подключение внешних устройств к микроконтроллеру с указанием выводов. В качестве схемы может быть использован скриншот Proteus.

4. Разработка программы.

В данном разделе необходимо привести схему алгоритма (если она использовалась при разработке программы), текст программы с подробными комментариями, а также скриншоты с результатами выполнения программы.

5. Вывод.

5. Контрольные вопросы

1. Расскажите о четырех режимах работы последовательного порта.
2. Какие регистры используются для управления последовательным портом?
3. Какие таймеры могут использоваться для задания скорости работы последовательного порта?
4. Организуйте работу системы без использования прерываний.
5. Разработайте микроконтроллерную систему, организующую вывод ФИО на дисплей *MILFORD-2X16-BKP* (при подаче питания на дисплей запускается инициализация, длящаяся около 500 мс).
6. Добавьте в разработанную микроконтроллерную систему зеленый и красный светодиоды, сигнализирующие о нормальной работе системы и наличии ошибки соответственно.

6. Варианты заданий

Номер варианта	Скорость UART, бит/с	Таймер	Операция
1	110	1	+
2	300	2	-
3	1200	1	*
4	2400	2	/
5	4800	1	|
6	9600	2	&
7	19200	1	^
8	38400	2	+
9	57600	1	-
10	62500	2	*
11	137,5	1	/
12	110	2	|
13	300	1	&
14	1200	2	^
15	2400	1	+
16	4800	2	-
17	9600	1	*
18	19200	2	/
19	38400	1	|
20	57600	2	&
21	62500	1	^
22	137,5	2	+
23	110	1	-
24	300	2	*
25	1200	1	/
26	2400	2	|
27	4800	1	&
28	9600	2	^
29	19200	1	+
30	38400	2	-
31	57600	1	*
32	62500	2	/
33	137,5	1	|
34	110	2	&
35	300	1	^
36	1200	2	+
37	2400	1	-
38	4800	2	*
39	9600	1	/

Номер варианта	Скорость UART, бит/с	Таймер	Операция
40	19200	2	|
41	38400	1	&
42	57600	2	^
43	62500	1	+
44	137,5	2	-
45	110	1	*
46	300	2	/
47	1200	1	|
48	2400	2	&
49	4800	1	^
50	9600	2	+

Обозначения:
+ – сложение,
- – вычитание,
* – умножение,
/ – деление,
| – поразрядное логическое ИЛИ,
& – поразрядное логическое И,
^ – сложение по модулю два.

Список литературы

1. *Магда, Ю. С.* Современные микроконтроллеры. Архитектура, программирование, разработка устройств / Ю. С. Магда. М. : ДМК Пресс, 2017. 224 с.
2. *Магда, Ю. С.* Микроконтроллеры серии 8051 : практический подход / Ю. С. Магда. М. : ДМК Пресс, 2008. 228 с.
3. *MCS® 51 Microcontroller Family User's Manual*, http://web.mit.edu/6.115/www/document/8051.pdf