

МАТЕРИАЛ 2.

ОСНОВЫ ПРОГРАММИРОВАНИЯ
ДЛЯ ВСТРАИВАЕМЫХ МИКРОПРОЦЕССОРНЫХ СИСТЕМ.

Цель – изучить основные элементы программирования МК-систем, ведение функций, интерфейс с ассемблером для семейства MCS-51 и производных.

1. АРХИТЕКТУРА МИКРОКОНТРОЛЛЕРА INTEL 8051 (СЕМЕЙСТВО MCS-51)

Это самостоятельный класс интегральных схем – **микроконтроллеры**, которые предназначены для встраивания в приборы различного назначения. Их отличает наличие встроенной памяти, развитые средства взаимодействия с внешними устройствами. Через четыре программируемых параллельных порта ввода/вывода и один последовательный порт микроконтроллер взаимодействует с внешними устройствами. Основу структурной схемы (рис. 1а) образует внутренняя двунаправленная 8-битная шина, которая связывает между собой основные узлы и устройства микроконтроллера: резидентную память программ (RPM), резидентную память данных (RDM), арифметико-логическое устройство (ALU), блок регистров специальных функций, устройство управления (CU) и порты ввода/вывода (P0-P3).

Структурная схема микроконтроллера Intel 8051

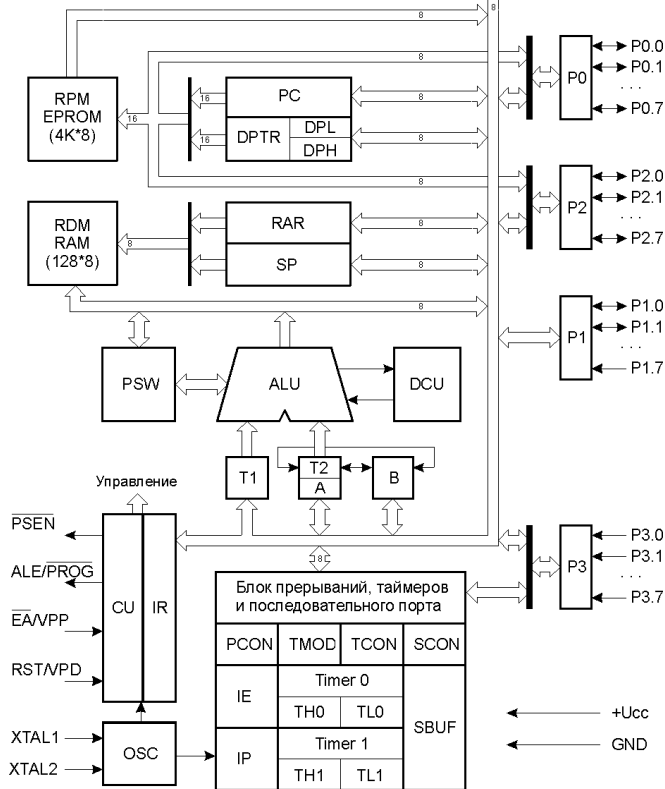


Рис. 1а.

В архитектуре 80C51XA, дополняя вычислительное ядро дополнительными блоками и устройствами, получают различные представители этого семейства, такие как XA-S3, XA-G3, XA-G4, XA-H3 и XA-C3. Семейство Philips XA пока не завершено. Для приведенных представителей XA шина адреса неполно разрядная. Все они пока работают в пределах 1 Мбайт, хотя ядро спроектировано на будущее для работы с 16 Мбайт памяти.

Хотя семейство XA – 16-разрядная архитектура, регистры специальных функций (SFR) в ней, по традиции, 8-разрядные. Однако следует отметить, что число SFR-регистров существенно возросло.

Система команд существенно расширена. Дополнены режимы адресации. Есть и байтовые команды (старые), и команды работы со словами. Умножение и деление затрагивают 32-разрядные данные. Реализована поддержка многозадачности. Развита система прерываний. Три таймера, плюс сторожевой таймер. Два последовательных интерфейса.

Структурная схема достаточно развитого представителя семейства XA приведена на рис. 1в. Дополнительно присутствует канал CAN.

Развитием 8-разрядной архитектуры 80C51 является высокопроизводительная 16-разрядная архитектура 80C51XA, в основе которой заложено вычислительное ядро, приведенное ниже на рис. 1б.

Архитектура ядра семейства Philips XA

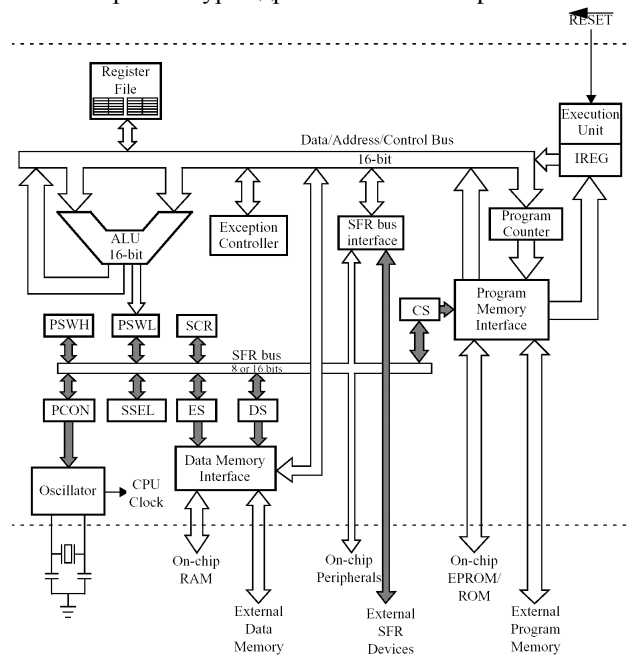


Рис. 1б.

Структурная схема микроконтроллера XA-C3

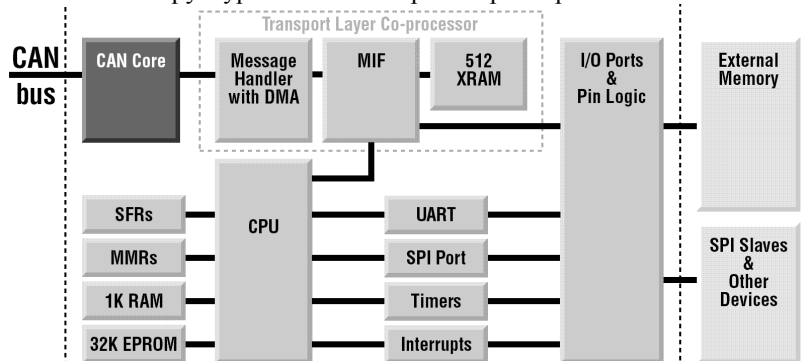


Рис. 1в.

1.1. АРИФМЕТИКО-ЛОГИЧЕСКОЕ УСТРОЙСТВО

8-битное арифметико-логическое устройство (ALU) может выполнять арифметические операции сложения, вычитания, умножения и деления; логические операции И, ИЛИ, исключающее ИЛИ, а также операции циклического сдвига, сброса, инвертирования и т.п. К входам подключены программно-недоступные регистры T1 и T2, предназначенные для временного хранения операндов, схема десятичной коррекции (DCU) и схема формирования признаков результата операции (PSW).

Простейшая операция сложения используется в ALU для инкрементирования содержимого регистров, продвижения регистра-указателя данных (RAR) и автоматического вычисления следующего адреса резидентной памяти программ. Простейшая операция вычитания используется в ALU для декрементирования регистров и сравнения переменных.

Простейшие операции автоматически образуют “тандемы” для выполнения таких операций, как, например, инкрементирование 16-битных регистровых пар. В ALU реализуется механизм каскадного выполнения простейших операций для реализации сложных команд. Так, например, при выполнении одной из команд условной передачи управления по результату сравнения в ALU трижды инкрементируется счётчик команд (PC), дважды производится чтение из RDM, выполняется арифметическое сравнение двух переменных, формируется 16-битный адрес перехода и принимается решение о том, делать или не делать переход по программе. Все перечисленные операции выполняются всего лишь за 2 мкс.

Важной особенностью ALU является его способность оперировать не только байтами, но и битами. Отдельные программно-доступные биты могут быть установлены, сброшены, инвертированы, переданы, проверены и использованы в логических операциях. Эта способность достаточно важна, поскольку для управления объектами часто применяются алгоритмы, содержащие операции над входными и выходными булевыми переменными, реализация которых средствами обычных микропроцессоров сопряжена с определенными трудностями.

Таким образом, ALU может оперировать четырьмя типами информационных объектов: **булевыми** (1 бит), **цифровыми** (4 бита), **байтными** (8 бит) и **адресными** (16 бит). В ALU выполняется 51 различная операция пересылки или преобразования этих данных. Так как используется **11 режимов адресации** (7 для данных и 4 для адресов), то путем комбинирования операции и режима адресации базовое число команд 111 расширяется до 255 из 256 возможных при однобайтном коде операции.

1.2. РЕЗИДЕНТНАЯ ПАМЯТЬ ПРОГРАММ И ДАННЫХ

Резидентные (размещённые на кристалле) память программ (RPM) и память данных (RDM) физически и логически разделены, имеют различные механизмы адресации, работают под управлением различных сигналов и выполняют разные функции.

Память программ RPM имеет ёмкость 4 Кбайта и предназначена для хранения команд, констант, управляющих слов инициализации, таблиц перекодировки входных и выходных переменных и т.п. Память имеет 16-битную шину адреса, через которую обеспечивается доступ из программного счётчика PC или из регистра-указателя данных (DPTR). DPTR выполняет функции базового регистра при косвенных переходах по программе или используется в операциях с таблицами.

Память данных RDM предназначена для хранения переменных в процессе выполнения прикладной программы, адресуется одним байтом и имеет ёмкость 128 байт. Кроме того, к её адресному пространству примыкают адреса регистров специальных функций, которые перечислены в табл. 1.

Память программ, так же как и память данных, может быть расширена до 64 Кбайт путем подключения внешних микросхем.

1.3. АККУМУЛЯТОР, РЕГИСТРЫ ОБЩЕГО НАЗНАЧЕНИЯ И ФЛАГИ

Аккумулятор (A) является источником операнда и местом фиксации результата при выполнении арифметических, логических операций и ряда операций передачи данных. Кроме того, только с использованием аккумулятора могут быть выполнены операции сдвига, проверка на нуль, формирование флага паритета и т.п. В распоряжении пользователя имеются 8 регистров общего назначения R0–R7 одного из четырёх возможных банков. При выполнении многих команд в ALU формируется ряд признаков операции (флагов), которые фиксируются в регистре PSW.

Таблица 1

Блок регистров специальных функций		
Символ	Наименование	Адрес
* A	Аккумулятор	0E0H
* B	Регистр-расширитель аккумулятора	0F0H
* PSW	Слово состояния программы	0D0H
SP	Регистр-указатель стека	81H
DPTR	Регистр-указатель данных (DPH) (DPL)	83H
		82H
* P0	Порт 0	80H
* P1	Порт 1	90H
* P2	Порт 2	0A0H
* P3	Порт 3	0B0H
* IP	Регистр приоритетов прерываний	0B8H
* IE	Регистр маски прерываний	0A8H
TMOD	Регистр режима таймера/счётчика	89H
* TCON	Регистр управления/статуса таймера	88H
TH0	Таймер 0 (старший байт)	8CH
TL0	Таймер 0 (младший байт)	8AH
TH1	Таймер 1 (старший байт)	8DH
TL1	Таймер 1 (младший байт)	8BH
* SCON	Регистр управления приёмопередатчиком	98H
SBUF	Буфер приёмопередатчика	99H
PCON	Регистр управления мощностью	87H

Примечание. Регистры, имена которых отмечены знаком (*), допускают адресацию отдельных битов.

В табл. 2 приводится перечень флагов PSW, даются их символические имена и описываются условия их формирования.

Таблица 2

Формат слова состояния программы PSW		
Символ	Разряд	Имя и назначение
C	PSW.7	Флаг переноса. Устанавливается и сбрасывается аппаратно или программно при выполнении арифметических и логических операций
AC	PSW.6	Флаг вспомогательного переноса. Устанавливается и сбрасывается только аппаратно при выполнении команд сложения и вычитания и сигнализирует о переносе или займе в бите 3
F0	PSW.5	Флаг 0. Может быть установлен, сброшен или проверен программой как флаг, специфицируемый пользователем
RS1	PSW.4	Выбор банка регистров. Устанавливается и сбрасывается программно для выбора рабочего банка регистров (табл. 3)
RS0	PSW.3	
OV	PSW.2	Флаг переполнения. Устанавливается и сбрасывается аппаратно при выполнении арифметических операций
-	PSW.1	Не используется
P	PSW.0	Флаг паритета. Устанавливается и сбрасывается аппаратно в каждом цикле и фиксирует нечётное/чётное число единичных битов в аккумуляторе, т.е. выполняет контроль по четности

Таблица 3

Выбор рабочего банка регистров			
RS1	RS0	Банк	Границы адресов
0	0	0	00H – 07H
0	1	1	08H – 0FH
1	0	2	10H – 17H
1	1	3	18H – 1FH

Наиболее “активным” флагом PSW является флаг переноса, который принимает участие и модифицируется в процессе выполнения множества операций, включая сложение, вычитание и сдвиги. Кроме того, флаг переноса (C) выполняет функции “булева аккумулятора” в командах, манипулирующих с битами. Флаг переполнения (OV) фиксирует арифметическое переполнение при операциях над целыми числами со знаком и делает возможным использование арифметики в дополнительных кодах. ALU не управляет флагами селекции банка регистров (RS0, RS1), их значение полностью определяется прикладной программой и используется для выбора одного из четырёх регистровых банков.

В микропроцессорах, архитектура которых опирается на аккумулятор, большинство команд работают с ним, используя неявную адресацию. В Intel 8051 дело обстоит иначе. Хотя процессор имеет в своей основе аккумулятор, он может выполнять множество команд и без его участия. Например, данные могут быть переданы из любой ячейки RDM в любой регистр, любой регистр может быть загружен непосредственным операндом и т.д. Многие логические операции могут быть выполнены без участия аккумулятора. Кроме того, переменные могут быть инкрементированы, декрементированы и проверены без использования аккумулятора. Флаги и управляющие биты могут быть проверены и изменены аналогично.

1.4. РЕГИСТРЫ-УКАЗАТЕЛИ

8-битный указатель стека (SP) может адресовать любую область RDM. Его содержимое инкрементируется прежде, чем данные будут запомнены в стеке в ходе выполнения команд PUSH и CALL. Содержимое SP декрементируется после выполнения команд POP и RET. Подобный способ адресации элементов стека называют прединкрементным/постдекрементным. В процессе инициализации микроконтроллера после сигнала RST в SP автоматически загружается код 07H. Это значит, что если прикладная программа не переопределяет стек, то первый элемент данных в стеке будет располагаться в ячейке RDM с адресом 08H.

Двухбайтный регистр-указатель данных DPTR обычно используется для фиксации 16-битного адреса в операциях с обращением к внешней памяти. Командами микроконтроллера регистр-указатель данных может быть использован или как 16-битный регистр, или как два независимых 8-битных регистра (DPH и DPL).

1.5. РЕГИСТРЫ СПЕЦИАЛЬНЫХ ФУНКЦИЙ

Регистры с символическими именами IP, IE, TMOD, TCON, SCON и PCON используются для фиксации и программного изменения управляющих бит и бит состояния схемы прерывания, таймера/счётчика, приёмопередатчика последовательного порта и для управления энергопотреблением. Их организация будет описана ниже при рассмотрении особенностей работы микроконтроллера в различных режимах.

1.6. УСТРОЙСТВО УПРАВЛЕНИЯ И СИНХРОНИЗАЦИИ

Кварцевый резонатор, подключаемый к внешним выводам микроконтроллера, управляет работой внутреннего генератора, который в свою очередь формирует сигналы синхронизации. Устройство управления (CU) на основе сигналов синхронизации формирует машинный цикл фиксированной длительности, равной 12 периодам резонатора. Большинство команд микроконтроллера выполняется за один машинный цикл. Некоторые команды, оперирующие с 2-байтными словами или связанными с обращением к внешней памяти, выполняются за два машинных цикла. Только команды деления и умножения требуют четырех машинных циклов. На основе этих особенностей работы устройства управления производится расчёт времени исполнения прикладных программ.

На схеме микроконтроллера к устройству управления примыкает регистр команд (IR). В его функцию входит хранение кода выполняемой команды.

Входные и выходные сигналы устройства управления и синхронизации:

- PSEN – разрешение программной памяти,
- ALE – выходной сигнал разрешения фиксации адреса,
- PROG – сигнал программирования,
- EA – блокировка работы с внутренней памятью,
- VPP – напряжение программирования,
- RST – сигнал общего сброса,
- VPD – вывод резервного питания памяти от внешнего источника,
- XTAL – входы подключения кварцевого резонатора.

1.7. ПАРАЛЛЕЛЬНЫЕ ПОРТЫ ВВОДА/ВЫВОДА ИНФОРМАЦИИ

Все четыре порта (P0-P3) предназначены для ввода или вывода информации побайтно. Каждый порт содержит управляемые регистр-защёлку, входной буфер и выходной драйвер.

Выходные драйверы портов 0 и 2, а также входной буфер порта 0 используются при обращении к внешней памяти. При этом через порт 0 в режиме временного мультиплексирования сначала выводится младший байт адреса, а затем выдается или принимается байт данных. Через порт 2 выводится старший байт адреса в тех случаях, когда разрядность адреса равна 16 бит.

Все выводы порта 3 могут быть использованы для реализации альтернативных функций, перечисленных в табл. 4. Эти функции могут быть задействованы путем записи 1 в соответствующие биты регистра-защёлки (P3.0-P3.7) порта 3.

Таблица 4

Альтернативные функции порта P3		
Символ	Разряд	Имя и назначение
RD	P3.7	Чтение. Активный сигнал низкого уровня формируется аппаратно при обращении к внешней памяти данных
WR	P3.6	Запись. Активный сигнал низкого уровня формируется аппаратно при обращении к внешней памяти данных
T1	P3.5	Вход таймера/счётчика 1 или тест-вход
T0	P3.4	Вход таймера/счётчика 0 или тест-вход
INT1	P3.3	Вход запроса прерывания 1. Воспринимается сигнал низкого уровня или срез
INT0	P3.2	Вход запроса прерывания 0. Воспринимается сигнал низкого уровня или срез
TXD	P3.1	Выход передатчика последовательного порта в режиме UART. Выход синхронизации в режиме регистра сдвига
RXD	P3.0	Вход приёмника последовательного порта в режиме UART. Ввод/вывод данных в режиме регистра сдвига

Порт 0 является двунаправленным, а порты 1-3 - квазидвунаправленными. Каждая линия портов может быть использована независимо для ввода или вывода.

По сигналу RST в регистры-защёлки всех портов автоматически записываются единицы, настраивающие их тем самым на режим ввода.

Все порты могут быть использованы для организации ввода/вывода информации по двунаправленным линиям передачи. Однако порты 0 и 2 не могут быть использованы для этой цели в случае, если система имеет внешнюю память, связь с которой организуется через общую разделяемую шину адреса/данных, работающую в режиме временного мультиплексирования.

Обращение к портам ввода/вывода возможно с использованием команд, оперирующих с байтом, отдельным битом, произвольной комбинацией битов. При этом в тех случаях, когда порт является одновременно операндом и местом назначения результата, устройство управления автоматически реализует специальный режим, который называется “чтение-модификация-запись”. Этот режим обращения предполагает ввод сигналов не с внешних выводов порта, а из его регистра-защёлки, что позволяет исключить неправильное считывание ранее выведенной информации. Этот механизм обращения к портам реализован в командах:

- ANL – логическое И, например, ANL P1,A;
- ORL – логическое ИЛИ, например, ORL P2,A;
- XRL – исключающее ИЛИ, например, XRL P3,A;
- JBC – переход, если в адресуемом бите единица, и последующий сброс бита, например, JBC P1.1, LABEL;
- CPL – инверсия бита, например, CPL P3.3;
- INC – инкремент порта, например, INC P2;
- DEC – декремент порта, например, DEC P2;
- DJNZ – декремент порта и переход, если его содержимое не равно нулю, например, DJNZ rr, LABEL;
- MOV PX.Y,C – передача бита переноса в бит Y порта X;
- SET PX.Y – установка бита Y порта X;
- CLR PX.Y – сброс бита Y порта X.

1.8. ТАЙМЕР/СЧЁТЧИК

В составе микроконтроллера имеются регистровые пары с символическими именами TH0, TL0 и TH1, TL1, на основе которых функционируют два независимых программно-управляемых 16-битных таймера/счётчика событий

(T/C0 и T/C1). При работе в качестве таймера содержимое T/C инкрементируется в каждом машинном цикле, то есть через каждые 12 периодов резонатора. При работе в качестве счётчика содержимое T/C инкрементируется под воздействием перехода из 1 в 0 внешнего входного сигнала, подаваемого на соответствующий (T0, T1) вход микроконтроллера. Опрос сигналов выполняется в каждом машинном цикле. Так как на распознавание перехода требуется два машинных цикла, то максимальная частота подсчёта входных сигналов равна 1/24 частоты резонатора. На длительность периода входных сигналов ограничений сверху нет. Для гарантированного прочтения входного считываемого сигнала он должен удерживать значение 1 как минимум в течение одного машинного цикла.

Для управления режимами работы и для организации взаимодействия таймеров с системой прерывания используются два регистра специальных функций TMOD и TCON, описание которых приводится в табл. 5-7. Для обоих T/C режимы работы 0, 1 и 2 одинаковы. Режимы 3 для T/C0 и T/C1 различны.

Таблица 5

Регистр режима работы таймера/счётчика		
Символ	Разряд	Имя и назначение
GATE	TMOD.7 для T/C1 TMOD.3 для T/C0	Управление блокировкой. Если бит установлен, то таймер/счётчик “х” разрешен до тех пор, пока на входе “INT х” высокий уровень и бит управления “TRx” установлен. Если бит сброшен, то T/C разрешается, как только бит управления “TRx” устанавливается
C/T	TMOD.6 для T/C1 TMOD.2 для T/C0	Бит выбора режима таймера или счётчика событий. Если бит сброшен, то работает таймер от внутреннего источника сигналов синхронизации. Если бит установлен, то работает счётчик от внешних сигналов на входе “Tx”
M1	TMOD.5 для T/C1 TMOD.1 для T/C0	Режим работы (см. табл. 6)
M0	TMOD.4 для T/C1 TMOD.0 для T/C0	

Таблица 6

Режимы работы таймера/счётчика		
M1	M0	Режим работы
0	0	“TLx” работает как 5-битный предделитель
0	1	16-битный таймер/счётчик. “THx” и “TLx” включены последовательно
1	0	8-битный автоперезагружаемый таймер/счётчик. “THx” хранит значение, которое должно быть перезагружено в “TLx” каждый раз по переполнению
1	1	Таймер/счётчик 1 останавливается. Таймер/счётчик 0: TL0 работает как 8-битный таймер/счётчик, и его режим определяется управляющими битами таймера 0. TH0 работает только как 8-битный таймер, и его режим определяется управляющими битами таймера 1

Режим 0. Перевод любого T/C в этот режим делает его 8-разрядным таймером, на вход которого подключен 5-битный предделитель частоты на 32. В этом режиме таймерный регистр имеет разрядность 13 бит. При переходе из состояния “все единицы” в состояние “все нули” устанавливается флаг прерывания от таймера TF1. Входной синхросигнал таймера 1 разрешен (поступает на вход T/C), когда управляющий бит TR1 установлен в 1 и либо управляющий бит GATE (блокировка) равен 0, либо на внешний вход запроса прерывания INT1 поступает уровень 1.

Установка бита GATE в 1 позволяет использовать таймер для измерения длительности импульсного сигнала, подаваемого на вход запроса прерывания.

Таблица 7

Регистр управления/статуса таймера		
Символ	Разряд	Имя и назначение
TF1	TCON.7	Флаг переполнения таймера 1. Устанавливается аппаратно при переполнении таймера/счётчика. Сбрасывается при обслуживании прерывания аппаратно
TR1	TCON.6	Бит управления таймера 1. Устанавливается/сбрасывается программой для пуска/останова
TF0	TCON.5	Флаг переполнения таймера 0. Устанавливается аппаратно. Сбрасывается при обслуживании прерывания
TR0	TCON.4	Бит управления таймера 0. Устанавливается/сбрасывается программой для пуска/останова таймера/счётчика
IE1	TCON.3	Флаг фронта прерывания 1. Устанавливается аппаратно, когда детектируется срез внешнего сигнала INT1. Сбрасывается при обслуживании прерывания
IT1	TCON.2	Бит управления типом прерывания 1. Устанавливается/сбрасывается программно для спецификации запроса INT1 (срез/низкий уровень)
IE0	TCON.1	Флаг фронта прерывания 0. Устанавливается по срезу сигнала INT0. Сбрасывается при обслуживании прерывания
IT0	TCON.0	Бит управления типом прерывания 0. Устанавливается/сбрасывается программно для спецификации запроса INT0 (срез/низкий уровень)

Режим 1. Работа любого T/C в этом режиме такая же, как и в режиме 0, за исключением того, что таймерный регистр имеет разрядность 16 бит.

Режим 2. В этом режиме работа организована таким образом, что переполнение (переход из состояния “все

единицы” в состояние “все нули”) 8-битного счётчика TL1 приводит не только к установке флага TF1, но и автоматически перезагружает в TL1 содержимое старшего байта (TH1) таймерного регистра, которое предварительно было задано программным путем. Перезагрузка оставляет содержимое TH1 неизменным. В режиме 2 T/C0 и T/C1 работают совершенно одинаково.

Режим 3. В этом режиме T/C0 и T/C1 работают по-разному. T/C1 сохраняет неизменным своё текущее содержимое. Иными словами, эффект такой же, как и при сбросе управляющего бита TR1 в нуль. В этом режиме TL0 и TH0 функционируют как два независимых 8-битных счётчика. Работу TL0 определяют управляющие биты T/C0 (C/T, GATE, TR0), входной сигнал INT0 и флаг переполнения TF0. Работу TH0, который может выполнять только функции таймера (подсчёт машинных циклов микроконтроллера), определяет управляющий бит TR1. При этом TH0 использует флаг переполнения TF1.

Режим 3 используется в тех случаях, когда требуется наличие дополнительного 8-битного таймера или счётчика событий. Можно считать, что в режиме 3 микроконтроллер имеет в своем составе три таймера/счётчика. В том случае, если T/C0 используется в режиме 3, T/C1 может быть или включен, или выключен, или переведен в свой собственный режим 3, или может быть использован последовательным портом в качестве генератора частоты передачи, или, наконец, может быть использован в любом применении, не требующем прерывания.

Совместимость с семейством XA.

Все таймеры/счётчики в режиме таймера тактируются одной частотой **TCLK**, значение которой можно устанавливать в программе пользователя из трёх возможных значений: **Osc/4**, **Osc/16**, **Osc/64**. Усовершенствован режим 0, который исключает 13-разрядный счётчик, а взамен делает автозагружаемый 16-разрядный счётчик. Для автозагрузки введены 4 дополнительных SFR-регистра – RTLx и RTHx, в которых хранится значение автозагрузки.

Значение автозагрузки для 2-го режима хранится также в RTLx, при этом величина в THx не меняется.

1.9. ПОСЛЕДОВАТЕЛЬНЫЙ ПОРТ

Через универсальный асинхронный приёмопередатчик UART (Universal Asynchronous Receiver-Transmitter) происходит передача информации, представленной последовательным кодом (младшими битами вперед), в полном дуплексном режиме обмена. В состав UART, называемого часто последовательным портом, входят принимающий и передающий сдвигающие регистры, а также специальный буферный регистр (SBUF) приёмопередатчика.

1.9.1. Регистр SBUF

Представляет собой два независимых регистра: буфер приёмника и буфер передатчика. Загрузка байта в SBUF немедленно вызывает начало процесса передачи через последовательный порт. Когда байт считывается из SBUF, это значит, что его источником является приёмник последовательного порта. Запись байта в буфер приводит к автоматической переписи байта в сдвигающий регистр передатчика и инициирует начало передачи байта. Наличие буферного регистра приёмника позволяет совмещать операцию чтения ранее принятого байта с приёмом очередного байта. Если к моменту окончания приёма байта предыдущий байт не был считан, то он будет потерян.

Последовательный порт может работать в четырех различных режимах (смотри табл. 8 и 9).

Режим 0. Информация передаётся и принимается через вход приёмника RXD. Принимаются и передаются 8 бит данных. Через внешний выход передатчика TXD выдаются импульсы сдвига, которые сопровождают каждый бит. Частота передачи равна 1/12 частоты резонатора.

Режим 1. Через TXD передаются или из RXD принимаются 10 бит: старт-бит (0), 8 бит данных и стоп-бит (1). Скорость приёма/передачи – величина переменная и задаётся таймером.

Режим 2. Через TXD передаются или из RXD принимаются 11 бит: старт-бит, 8 бит данных, программируемый девятый бит и стоп-бит. При передаче девятый бит может использоваться для повышения достоверности передачи путём контроля по чётности и в него можно поместить значение признака паритета из PSW. Частота приёма/передачи выбирается программно и может быть равна 1/32 или 1/64 частоты резонатора в зависимости от SMOD.

Режим 3. Совпадает с режимом 2, но частота приёма/передачи является величиной переменной и задаётся таймером.

Таблица 8

Регистр управления/статуса UART

Символ	Разряд	Имя и назначение
SM0	SCON.7	Биты управления режимом работы UART. Устанавливаются/сбрасываются программно (табл. 9)
SM1	SCON.6	Бит управления режимом UART. Устанавливается программно для запрета приёма сообщения, в котором девятый бит равен 0
SM2	SCON.5	Бит разрешения приёма. Устанавливается/сбрасывается программно для разрешения/запрета приёма последовательных данных
REN	SCON.4	Передача бита 8. Устанавливается/сбрасывается программно для задания девятого передаваемого бита в режиме UART -9 бит
TB8	SCON.3	Приём бита 8. Устанавливается/сбрасывается аппаратно для фиксации девятого принимаемого бита в режиме UART -9 бит
RB8	SCON.2	Флаг прерывания передатчика. Устанавливается аппаратно при окончании передачи байта. Сбрасывается программно после обслуживания прерывания
TI	SCON.1	Флаг прерывания приёмника. Устанавливается аппаратно при приёме байта. Сбрасывается программно после обслуживания прерывания
RI	SCON.0	

1.9.2. Регистр SCON

Регистр предназначен для управления режимом работы UART. Регистр содержит управляющие биты и девятый бит принимаемых или передаваемых данных RB8 и TB8, а также биты прерывания приёмопередатчика RI и TI. Функциональное назначение битов указано в табл. 8, а режимы работы в табл. 9.

Таблица 9

Прикладная программа путём загрузки в два старших разряда SCON определяет режим работы UART. Во всех режимах передача инициируется любой командой, где SBUF указан как получатель байта. Приём в UART в режиме 0 происходит при условии RI=0 и REN=1. В режимах 1-3 приём начинается с приходом старт-бита, если REN=1.

		Режим работы UART
SM0	SM1	Режим работы UART
0	0	Сдвигающий регистр расширения ввода/вывода
0	1	UART - 8 бит. Изменяемая скорость передачи
1	0	UART - 9 бит. Фиксированная скорость передачи
1	1	UART - 9 бит. Изменяемая скорость передачи

В TB8 программно устанавливается значение девятого бита данных, который будет передан в режиме 2 или 3. В RB8 фиксируется в режимах 2 и 3 девятый принимаемый бит данных. В режиме 1, если SM2=0, в бит RB8 заносится стоп-бит. В режиме 0 RB8 не используется.

Флаг прерывания передатчика TI устанавливается аппаратно в конце периода передачи восьмого бита данных в режиме 0 и в начале периода передачи стоп-бита в режимах 1-3. Подпрограмма обслуживания этого прерывания должна сбрасывать бит TI.

Флаг прерывания приёмника RI устанавливается аппаратно в конце периода приёма восьмого бита данных в режиме 0 и в середине периода приёма стоп-бита в режимах 1-3. Подпрограмма обслуживания прерывания должна сбрасывать бит RI.

1.9.3. Работа UART в мультиконтроллерных системах

В системах децентрализованного управления, которые используются для управления и регулирования в топологически распределенных объектах, возникает задача обмена информацией между множеством микроконтроллеров, объединенных в локальную вычислительно-управляющую сеть. Как правило, локальные сети на основе Intel 8051 имеют магистральную архитектуру с разделяемым моноканалом (коаксиальный кабель, витая пара, оптическое волокно), по которому осуществляется обмен информацией между контроллерами.

Бит SM2 в SCON позволяет простыми средствами реализовать межконтроллерный обмен. Механизм обмена построен на том, что в режимах 2 и 3 программируемый девятый бит данных при приёме фиксируется в бите RB8. UART может быть запрограммирован таким образом, что при получении стоп-бита прерывание от приёмника будет возможно только при условии RB8=1. Ведущий контроллер всем ведомым передаёт широковещательное сообщение с байтом-идентификатором абонента, которое отличается от байтов данных только тем, что в его девятом бите содержится 1. Ведомые по этому признаку вызывают подпрограммы сравнения байта-идентификатора с кодом собственного сетевого адреса. Адресуемый контроллер сбрасывает свой SM2 и готовится к приёму блока данных. Остальные ведомые микроконтроллеры оставляют неизменными свои SM2=1 и передают управление основной программе. При SM2=1 информационные байты в сети прерывания не вызывают.

В режиме 1 автономного микроконтроллера SM2 используется для контроля истинности стоп-бита. В режиме 0 SM2 не используется и должен быть сброшен.

1.9.4. Скорость приёма/передачи

Скорость зависит от режима работы UART. В режиме 0 частота зависит только от частоты кварцевого резонатора: $f_0 = f_{\text{рез}}/12$. За один машинный цикл передаётся один бит. В режимах 1-3 скорость зависит от значения управляющего бита SMOD в регистре специальных функций PCON (табл. 10). В режиме 2 частота передачи определяется выражением $f_2 = (2^{\text{SMOD}}/64)f_{\text{рез}}$.

Таблица 10

Регистр управления мощностью PCON		
Символ	Разряд	Наименование и функция
SMOD	PCON.7	Удвоенная скорость передачи. Если бит установлен в 1, то скорость передачи вдвое больше, чем при SMOD=0
-	PCON.6-4	Не используются
GF1	PCON.3	Флаги, специфицируемые пользователем (флаги общего назначения)
GF0	PCON.2	
PD	PCON.1	
IDL	PCON.0	Бит пониженной мощности. При установке в 1 микроконтроллер переходит в режим пониженного энергопотребления
		Бит холостого хода. Если бит установлен в 1, то микроконтроллер переходит в режим холостого хода

Примечание. При одновременной записи 1 в PD и IDL бит PD имеет преимущество. Сброс PCON выполняется путём загрузки в него кода 0XXX0000.

В режимах 1 и 3 в формировании частоты передачи кроме управляющего бита SMOD принимает участие таймер 1. При этом частота передачи зависит от частоты переполнения (OVT1) и определяется следующим образом: $f_{1,3} = (2^{\text{SMOD}}/32)f_{\text{OVT1}}$. Прерывание от таймера 1 в этом случае должно быть заблокировано. Сам T/C1 может работать и как таймер, и как счётчик событий в любом из трёх режимов. Однако наиболее удобно использовать режим таймера с автоперезагрузкой (старшая тетрада TMOD=0010B). При этом частота передачи определяется выражением

$f_{1,3} = (2^{SMOD}/32)(f_{\text{рез}}/12)(256 - (TH1))$. В табл. 11 приводится описание способов настройки Т/С1 для получения типовых частот передачи данных через UART.

Таблица 11

Настройка таймера 1 для управления частотой работы UART

Частота приёма/ передачи (BAUD RATE)		Частота резонатора, МГц	SMOD	Таймер/счётчик 1		
				С/Т	Режим (MODE)	Перезагружаемое число
Режим 0, макс.:	1 МГц	12	X	X	X	X
Режим 2, макс.:	375 кГц	12	1	X	X	X
Режимы 1,3:	62,5 кГц	12	1	0	2	0FFH
	19,2 кГц	11,059	1	0	2	0FDH
	9,6 кГц	11,059	0	0	2	0FDH
	4,8 кГц	11,059	0	0	2	0FAH
	2,4 кГц	11,059	0	0	2	0F4H
	1,2 кГц	11,059	0	0	2	0E8H
	137,5 Гц	11,059	0	0	2	1DH
	110 Гц	6	0	0	2	72H
	110 Гц	12	0	0	1	0FEEBH

1.10. СИСТЕМА ПЕРЕРЫВАНИЙ

Внешние прерывания INT0 и INT1 (рис. 2) могут быть вызваны уровнем или переходом сигнала из 1 в 0 на входах микроконтроллера в зависимости от значений управляющих битов IT0 и IT1 в регистре TCON. От внешних прерываний устанавливаются флаги IE0 и IE1 в регистре TCON, которые инициируют вызов соответствующей подпрограммы обслуживания прерывания. Сброс этих флагов выполняется аппаратно только в том случае, если прерывание было вызвано по переходу (срезу) сигнала. Если же прерывание вызвано уровнем входного сигнала, то сбросом флага IE управляет соответствующая подпрограмма обслуживания прерывания путем воздействия на источник прерывания с целью снятия им запроса.

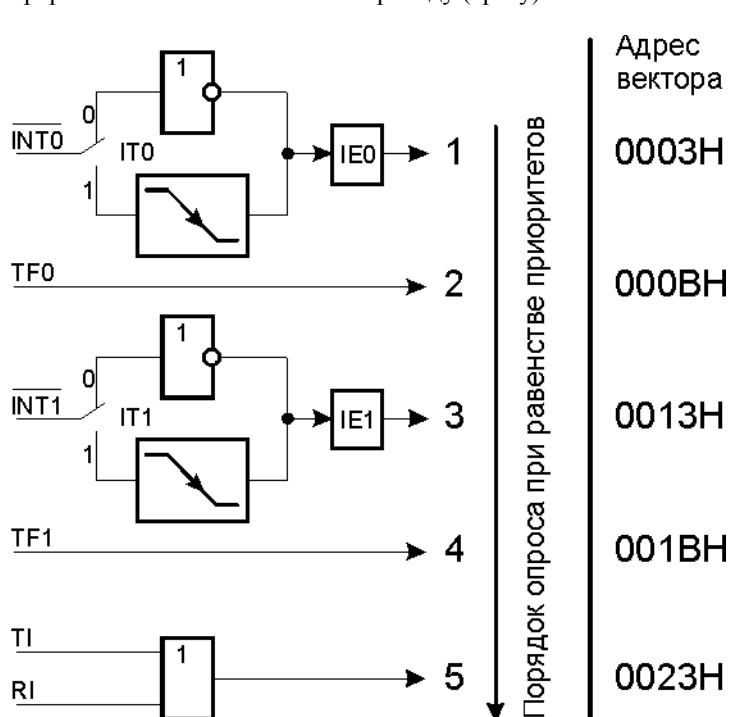


Рис. 2. Схема прерываний

Флаги запросов прерывания от таймеров TF0 и TF1 сбрасываются автоматически при передаче управления подпрограмме обслуживания. Флаги запросов прерывания RI и TI устанавливаются UART аппаратно, но сбрасываться должны программой. Прерывания могут быть вызваны или отменены программой, так как все перечисленные флаги программно доступны.

В блоке регистров специальных функций есть два регистра, предназначенных для управления режимом прерываний и уровнями приоритета. Форматы этих регистров, имеющих символические имена IE и IP описаны в табл. 12 и 13 соответственно.

Возможность программной установки/сброса любого управляющего бита в этих двух регистрах делает систему прерываний исключительно гибкой с 2 уровнями приоритетов.

Таблица 12

Регистр масок прерывания IE

Символ	Разряд	Имя и назначение
EA	IE.7	Снятие блокировки прерываний. Сбрасывается программно для запрета всех прерываний независимо от состояний IE4-IE0
-	IE.6, 5	Не используются
ES	IE.4	Бит разрешения прерывания от UART. Установка/сброс программой для разрешения/запрета прерываний от флагов TI, RI
ET1	IE.3	Бит разрешения прерывания от таймера 1. Установка/сброс программой для разрешения/запрета прерываний от таймера 1
EX1	IE.2	Бит разрешения внешнего прерывания 1. Установка/сброс программой для разрешения/запрета прерываний
ET0	IE.1	Разрешение прерывания от таймера 0. Работает аналогично IE.3

EX0	IE.0	Разрешения внешнего прерывания 0. Работает аналогично IE.2
-----	------	--

Таблица 13

Регистр приоритетов прерывания IP		
Символ	Разряд	Имя и назначение
-	IP.7-5	Не используются
PS	IP.4	Бит приоритета UART. Установка/сброс программой для назначения прерыванию от UART высшего/низшего приоритета
PT1	IP.3	Бит приоритета таймера 1. Установка/сброс программой для назначения прерыванию от таймера 1 высшего/низшего приоритета
PX1	IP.2	Бит приоритета внешнего прерывания 1. Установка/сброс программой для назначения прерыванию INT1 высшего/низшего приоритета
PT0	IP.1	Бит приоритета таймера 0. Работает аналогично IP.3
PX0	IP.0	Приоритет внешнего прерывания 0. Работает аналогично IP.2

Флаги прерываний опрашиваются в каждом машинном цикле. Ранжирование прерываний по приоритету выполняется в течение следующего машинного цикла. Система прерываний сформирует аппаратно вызов LCALL соответствующей подпрограммы обслуживания, если она не заблокирована одним из условий:

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

Примечание. Если флаг прерывания был установлен, но по одному из перечисленных условий не получил обслуживания и к моменту окончания блокировки уже был сброшен, то запрос прерывания теряется.

По аппаратно сформированному коду команды LCALL система прерывания помещает в стек содержимое программного счётчика PC и загружает в PC адрес вектора прерывания соответствующей подпрограммы обслуживания. По этому адресу должна быть расположена команда безусловного перехода JMP к начальному адресу подпрограммы обслуживания прерывания. Эта подпрограмма в случае необходимости должна начинаться командами записи в стек PUSH слова состояния программы PSW, аккумулятора A, расширителя аккумулятора B, указателя данных DPTR и т.д. и заканчиваться командами восстановления из стека POP. Подпрограммы обслуживания прерывания обязательно завершаются командой RETI, по которой в программный счётчик перезагружается из стека сохранённый адрес возврата в основную программу. Команда RET также возвращает управление, но при этом не снимает блокировку прерывания.

1.11. СОВМЕСТИМОСТЬ СТРУКТУРЫ РЕГИСТРОВ С АРХИТЕКТУРОЙ XA.

Полностью отдельный файл регистров семейства XA приведен на рис. 3. В будущем он может быть расширен до 16 регистров размером в слово. (Только эти дополнительные 8 регистров не могут быть указателями.)

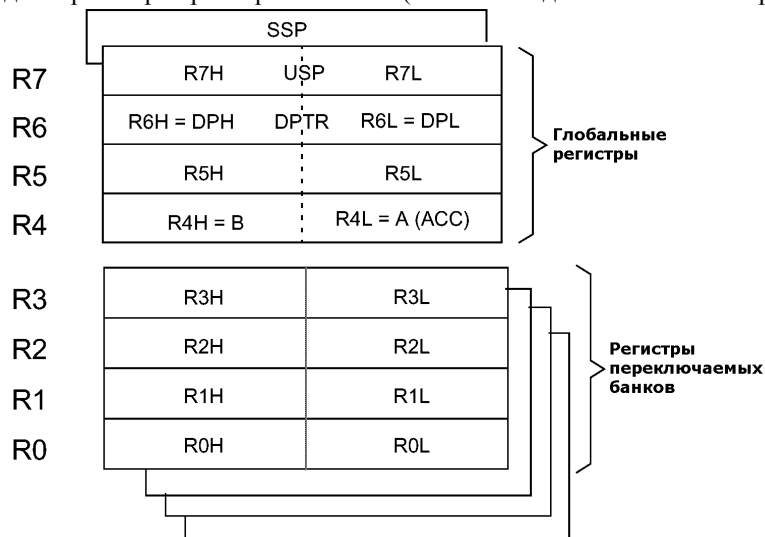


Рис. 3. Регистровый файл XA архитектуры.

Для XA все регистры доступны до бита, до байта и до слова. Все являются 16-разрядными указателями и могут появляться в соответствующих инструкциях. Младшие 4 регистра входят в состав 4 аналогичных переключаемых банков.

Аккумулятор A (ACC) транслируется в R4L. **Регистр B** – R4H. **Указатель данных DPTR** – R6.

Все 4 банка 8-разрядных регистров семейства 8051 транслируются в соответствующие регистры R0L – R3H. В режиме совместимости для XA эти регистры дополнительно дублируются в младшие 32 байта памяти данных. Косвенная адресация, использующая R0 и R1, работает с R0L и R0H, расширенными нулями до 16 разрядов.

PSW51 – это 8-разрядный аналог для совместимости (исходное PSW 16-разрядное).

2. СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРА INTEL 8051.

Система команд МК51 содержит 111 базовых команд, которые удобно разделить по функциональному признаку на пять групп: команды передачи данных, арифметических операций, логических операций, передачи управления и операций с битами. В её состав входят команды умножения, деления, вычитания, операций над битами, операций со стеком и расширенный набор команд передачи управления.

Большинство команд (94) имеют формат один или два байта и выполняются за один или два машинных цикла. При тактовой частоте 12 МГц длительность машинного цикла составляет 1 мкс.

Команды семейства MCS-51 фирмы Intel

Набор команд имеет 42 мнемонических обозначения (аббревиатур) инструкций для конкретизации 33 функций этой системы. Команды семейства MCS-51 фирмы Intel приведены в приложении.

Из 111 типов команд 64 выполняются за 12 тактов (1 мкс), 45 команд – за 24 такта (2 мкс) и 2 команды – умножение и деление выполняются за 48 тактов (4 мкс).

Всякая система команд покинута на режимах адресации данных и команд. Способы адресации:

[1] Адресация может быть **явная** и **неявная**. Неявная сокращает код, кодируется в поле КОП и соответствует некоторым режимам явной. Явная адресация – присутствует в кодировке инструкции дополнительными полями.

[2] Явная адресация – непосредственная, прямая и косвенная (разные уровни вложенности). **Непосредственная адресация** – данные в самой инструкции присутствуют дополнительными полями.

Прямая адресация – адрес данных в самой инструкции. Есть две памяти данных – ОЗУ и СОЗУ (регистры и их мало). **Регистровая адресация** – прямая в СОЗУ (мало бит для кодирования). Собственно **прямая адресация** – прямая в ОЗУ.

Простая **косвенная адресация** – в инструкции адрес объекта, который содержит действительный или часть адреса нужных данных. Эксплуатирует две предыдущие адресации и имеет множество вариаций.

Регистровая косвенная – регистр содержит адрес данных, = **индексная**, = **базовая**.

Регистровая относительная = **относительная** – к адресу в регистре добавляется непосредственное смещение; = **индексная относительная**, = **базовая относительная**.

Индексно-базовая адресация – адрес суммируется из содержимого двух регистров.

[3] Явная адресация может быть полной и краткой (неявная – всегда полная). При **краткой** адресации часть адреса неявно подразумевается, что сокращает код инструкций. Пример **полной** адресации – адрес с явным сегментом (селектором) памяти.

[4] Адресация может быть обычной и автоматической. **Автоматическая адресация** подразумевает изменение объекта, указующего на данные, без дополнительных команд в программе. Увеличение адреса – **автоинкрементная** адресация. Уменьшение адреса – **автодекрементная** адресация.

1. Адресация команд для семейства MCS-51 – 1) адресация последовательных команд + 2) адресация переходов. Для 1-го – автоинкрементная (регистровая косвенная) через регистр PC (неявная).

Для 2-го – реализовано 4 режима, 1 для прямой и 3 для косвенной. **Прямая адресация** в двух вариантах: - дальний переход (2 команды – LJMP и LCALL); - абсолютный переход в пределах страницы 2048 байт (2 команды – AJMP и ACALL).

Относительная адресация через регистр PC – это безусловный короткий переход (SJMP) и все условные переходы в пределах от -128 до +127.

Автодекрементная (регистровая косвенная) через регистр SP (неявная) – это команды RET и RETI.

Базово-индексная косвенная адресация (неявная, 1 база и 1 индекс) – JMP @A+DPTR.

2. Адресация данных для семейства MCS-51. 1). **Непосредственная** адресация в двух вариантах: для байта и для слова.

2). **Регистровая адресация** применяется как явная, так и неявная. Явная регистровая адресация применяется для доступа к 8 регистрам общего назначения. Неявная адресация – это обращения к **аккумулятору**. Роль битового аккумулятора выполняет флаг переноса – C. 8-разрядный аккумулятор – A (ACC) с адресом E0h. Дополнительный аккумулятор для команд деления и умножения – регистр B. Для 16-разрядных адресов используется регистр указатель – DPTR, доступный по частям.

3). **Прямая адресация** применяется в двух вариантах: для байтовых и для битовых данных во внутренней памяти и регистрах SFR. В обоих случаях размер адреса – 8 разрядов.

4). **Регистровая косвенная** адресация с регистрами R0 и R1 применяется для доступа к данным внутренней и внешней памяти. Для внешней памяти доступна 0 страница.

5). **Регистровая косвенная** адресация (неявная) с регистром DPTR для доступа к внешней памяти.

6). **Базово-индексная** косвенная адресация (неявная) с регистром DPTR и аккумулятором для доступа к данным в памяти программ (MOVC A,@A+DPTR).

7). **Базово-индексная** косвенная адресация (неявная) с регистром PC и аккумулятором для доступа к данным в памяти программ (MOVC A,@A+PC).

В двухадресных командах возможны лишь некоторые сочетания этих режимов адресации для операндов.

Совместимость команд семейства MCS-51 с архитектурой XA.

Инструкция 80C51	Перевод для XA
Арифметические операции	
ADD A, Rn	ADD.b R, R
ADD A, #data8	ADD.b R, #data8
ADD A, dir8	ADD.b R, direct
ADD A, @Ri	ADD.b R, [R]
ADDC A, Rn	ADDC.b R, R
ADDC A, #data8	ADDC.b R, #data8
ADDC A, dir8	ADDC.b R, direct
ADDC A, @Ri	ADDC.b R, [R]
SUBB A, Rn	SUBB.b R, R

Инструкция 80C51	Перевод для XA
Логические 8-р операции	
ANL A, Rn	AND.b R, R
ANL A, #data8	AND.b R, #data8
ANL A, dir8	AND.b R, direct
ANL A, @Ri	AND.b R, [R]
ANL dir8, A	AND.b direct, R
ANL dir8, #data8	AND.b direct, #data8
ORL A, Rn	OR.b R, R
ORL A, #data8	OR.b R, #data8
ORL A, dir8	OR.b R, direct

SUBB A, #data8	SUBB.b R, #data8
SUBB A, dir8	SUBB.b R, direct
SUBB A, @Ri	SUBB.b R, [R]
INC Rn	ADDS.b R, #1
INC dir8	ADDS.b direct, #1
INC @Ri	ADDS.b [R], #1
INC A	ADDS.b R, #1
INC DPTR	ADDS.w R, #1
DEC Rn	ADDS.b R, #-1
DEC dir8	ADDS.b direct, #-1
DEC @Ri	ADDS.b [R], #-1
DEC A	ADDS.b R, #-1
MUL AB	MULU.b R, R
DIV AB	DIVU.b R, R
DA A	DA R

Инструкция 80C51	Перевод для ХА
Пересылка данных	
MOV A, Rn	MOV.b R, R
MOV A, #data8	MOV.b R, #data8
MOV A, dir8	MOV.b R, direct
MOV A, @Ri	MOV.b R, [R]
MOV Rn, A	MOV.b R, R
MOV Rn, #data8	MOV.b R, #data8
MOV Rn, dir8	MOV.b R, direct
MOV dir8, A	MOV.b direct, R
MOV dir8, #data8	MOV.b direct, #data8
MOV dir8, Rn	MOV.b direct, R
MOV dir8, dir8	MOV.b direct, direct
MOV dir8, @Ri	MOV.b direct, [R]
MOV @Ri, A	MOV.b [R], R
MOV @Ri, dir8	MOV.b [R], direct
MOV @Ri, #data8	MOV.b [R], #data8
MOV DPTR, #data16	MOV.w R, #data16
XCH A, Rn	XCH.b R, R
XCH A, dir8	XCH.b R, direct
XCH A, @Ri	XCH.b R, R
XCHD A, @Ri	= программно
PUSH dir8	PUSH.b direct
POP dir8	POP.b direct
MOVX A, @Ri	MOVX.b R, [R]
MOVX A, @DPTR	MOVX.b R, [R]
MOVX @Ri, A	MOVX.b [R], R
MOVX @DPTR, A	MOVX.b [R], R
MOVC A, @A+DPTR	MOVC.bA, [A+DPTR]
MOVC A, @A+PC	MOVC.bA, [A+PC]

PUSH R4H ; Запомнить В.
 MOV R4H,(Ri) ; 2-ой операнд.
 RR R4H,#4 ; Обменять 3 и 4 тетрады.
 RR R4L,#4 ; Обменять тетрады аккумулятора.
 RL R4,#4 ; Обменять байты.
 MOV (Ri),R4H ; Записать результат.
 POP R4H ; Восстановить В.

ORL A, @Ri	OR.b R, [R]
ORL dir8, A	OR.b direct, R
ORL dir8, #data8	OR.b direct, #data8
XRL A, Rn	XOR.b R, R
XRL A, #data8	XOR.b R, #data8
XRL A, dir8	XOR.b R, direct
XRL A, @Ri	XOR.b R, [R]
XRL dir8, A	XOR.b direct, R
XRL dir8, #data8	XOR.b direct, #data8
CLR A	MOVS R, #0
CPL A	CPL.b R
SWAP A	RL.b R, #4
RL A	RL.b R, #1
RLC A	RLC.b R, #1
RR A	RR.b R, #1
RRC A	RRC.b R, #1
CLR C	CLR bit
CLR bit	CLR bit
SETB C	SETB bit
SETB bit	SETB bit
CPL C	XOR.b PSWL, #data8
CPL bit	XOR.b direct, #data8
ANL C, bit	AND C, bit
ANL C, /bit	AND C, /bit
ORL C, bit	OR C, bit
ORL C, /bit	OR C, /bit
MOV C, bit	MOV C, bit
MOV bit, C	MOV bit, C

Инструкция 80C51	Перевод для ХА
Относительные переходы	
SJMP rel8	BR rel8
CJNE A, dir8, rel	CJNE.b R, direct, rel
CJNE A, #data8, rel	CJNE.b R, #data8, rel
CJNE Rn, #data8, rel	CJNE.b R, #data8, rel
CJNE @Ri, #data8, rel	CJNE.b [R], #data8, rel
DJNZ Rn, rel	DJNZ.b R, rel
DJNZ dir8, rel	DJNZ.b direct, rel
JZ rel	JZ rel
JNZ rel	JNZ rel
JC rel	BCS rel
JNC rel	BCC rel
Jumps, Calls, Returns, и разные.	
NOP	NOP
AJMP addr11	JMP rel16
LJMP addr16	JMP rel16
JMP @A+DPTR	JUMP [A+DPTR]
ACALL addr11	CALL rel16
LCALL addr16	CALL rel16
RET	RET
RETI	RETI

3. ПРОЦЕДУРЫ И ПЕРЕДАЧА ПАРАМЕТРОВ В ПРОГРАММЕ.

3.1. Процедуры и функции

Принято разделять языки программирования на **процедурные** (C, Fortran, BASIC) и **непроцедурные** (LISP; FORTH, PROLOG), где процедуры – блоки кода программы, имеющие одну точку входа и одну точку выхода и возвращающие управление на следующую команду после команды передачи управления процедуре. Ассемблер одинаково легко можно использовать как процедурный язык и как непроцедурный, и в большинстве случаев можно успешно нарушать рамки и того, и другого подхода.

3.2 Передача параметров

На каком уровне абстракции понимать параметр, так и организовывать работу с ним. В противном случае работаем только со значением.

Процедуры могут получать или не получать параметры из вызывающей процедуры и могут возвращать или не возвращать результаты (*процедуры, которые что-либо возвращают, называются функциями* в языке Pascal, но ассемблер не делает каких-либо различий между ними).

Параметры можно передавать с помощью одного из шести **механизмов**:

- по значению;
- по ссылке;
- по возвращаемому значению;
- по результату;
- по имени;
- отложенным вычислением.

Параметры можно передавать в зависимости от вида носителя в одном из пяти **мест**:

- в регистрах;
- в глобальных переменных;
- в стеке;
- в потоке кода;
- в блоке параметров.

Так что всего в ассемблере возможно 30 различных способов передачи параметров для процедур.

[м1] Передача параметров по значению. Процедуре передаётся собственно значение параметра. При этом фактически значение параметра копируется, и процедура использует его копию, так что модификация исходного параметра оказывается невозможной. Этот механизм применяется для передачи небольших параметров, таких как байты или слова (данные и указатели на них).

Например, если параметры передаются в регистрах:

```
MOV    R6,vLoop      ; сделать копию значения в R6 - R7      / int, short, 16-разр. ptr
MOV    R7,vLoop+01H
LCALL  _?fFib        ; вызвать процедуру
```

[м2] Передача параметров по ссылке. Процедуре передается не значение переменной, а её адрес, по которому процедура должна сама прочитать значение параметра. Этот механизм удобен для передачи больших массивов данных и для тех случаев, когда процедура должна модифицировать параметры, хотя он и медленнее из-за того, что процедура будет выполнять дополнительные действия для получения собственно значений параметров.

```
MOV    DPTR,#fxMas    ; смещение массива fxMas
LCALL  ?C?LSTKXDATA   ; вызвать процедуру
```

[м3] Передача параметров по возвращаемому значению. Этот механизм объединяет передачу по значению и по ссылке. Процедуре передают адрес переменной, а процедура делает локальную копию параметра, затем работает с ней, а в конце записывает локальную копию обратно по переданному адресу. Этот метод эффективнее обычной передачи параметров по ссылке в тех случаях, когда процедура должна обращаться к параметру достаточно большое число раз, например, если используется передача параметров в глобальной переменной:

```
main:                                #define Numb 10
MOV    pVar,#LOW (pValue)           static char pdata *pVar, pdata pValue = 3, pdata pVal2;
LCALL  fFunc                         char fFunc( void );
MOV    R0,#LOW (pVal2)              main() {
MOV    A,R7                          pVar = &pValue;
MOVX   @R0,A                        pVal2 = fFunc();
; ... ..                             while ( 1 );
fFunc:                                }
MOV    R0,pVar                      char fFunc( void ) {
MOVX   A,@R0                        char pWork;
MOV    R7,A                          char iI;
; ... Накапливающий сумматор        pWork = *pVar;
; ... работа с R7 много раз.        // (команды, работающие с A в цикле десятки тысяч раз)
; ... ..                             // ... работа, накапливающий сумматор
MOV    R0,pVar                      for ( iI = 1; iI <= Numb; iI++ )
MOV    A,R7                          pWork += iI;
MOVX   @R0,A                        // ...
MOV    R7,#0AH                      *pVar = pWork;
RET                                return Numb; }
```

[м4] Передача параметров по результату. Этот механизм отличается от предыдущего только тем, что при вызове процедуры предыдущее значение параметра никак не определяется, а переданный адрес используется только для записи в него результата. Начального значения параметра на входе в процедуру нет.

[м5] Передача параметров по имени. Мы точно знаем имя параметра (символическое или даже просто текстовое), а что стоит за этим именем – вопрос второй. Это механизм, который используют макроопределения, директива EQU, а также например, препроцессор C при обработке директивы #define. При реализации этого механизма в компилирующем языке программирования (к которому относится и ассемблер) приходится заменять передачу параметра по имени другими механизмами при помощи, в частности, макроопределений.

//

Если определено макроопределение

```
#define PROC2( param ) pVar = & ## param;\
                                pValue = fFunc();

main() {
// .....
  PROC2( pVal2 )    // нет здесь «;»
  while ( 1 );
}
```

то теперь в программе можно передавать параметр так:

Примерно так же поступают языки программирования высокого уровня, поддерживающие этот механизм: процедура получает адрес специальной функции-заглушки, которая вычисляет адрес передаваемого по имени параметра. (Текстовые имена ресурсов; имя функции в таблице импорта; имя переменной окружения; ...)

[мб] Передача параметров отложенным вычислением. Как и в предыдущем случае, здесь процедура получает адрес функции, вычисляющей значение параметра (разные его приближения при разных обращениях). При передаче параметров отложенным вычислением функция получает адрес заглушки, которая при первом обращении к ней вычисляет значение параметра и сохраняет его во внутренней локальной переменной, а при дальнейших вызовах возвращает ранее вычисленное значение. В другом случае такой механизм удобен, если вычисление значения параметра требует много ресурсов или времени, например, если функция должна выбрать один из нескольких ходов при игре в шахматы, вычисление каждого параметра может занимать несколько минут. Если процедуре вообще не потребуются значения части параметров (например, если первый же ход приводит к мату), то использование отложенных вычислений способствует значительному выигрышу. Этот механизм чаще всего применяется в системах искусственного интеллекта и операционных системах.

Рассмотрев то, *как передавать* параметры процедуре, рассмотрим применяемые в ассемблере варианты, *где их передавать*.

[1] Передача параметров в регистрах. Если процедура получает небольшое число параметров, идеальным местом для их передачи оказываются регистры.

[2] Передача параметров в глобальных переменных. Когда не хватает регистров, один из способов обойти это ограничение – записать параметр в переменную, к которой затем обращаться из процедуры. Этот метод считается неэффективным, и его использование может привести к тому, что рекурсия и повторная входимость станут невозможными.

[3] Передача параметров в стеке. Параметры помещаются в стек сразу перед вызовом процедуры. Именно этот метод используют языки высокого уровня, такие как C и Pascal, для универсальных процессоров. Организуется структура под названием стековый кадр.

При внимательном анализе этого метода передачи параметров возникает сразу два вопроса: кто должен удалять параметры из стека, процедура или вызывающая её программа, и в каком порядке помещать параметры в стек. В обоих случаях оказывается, что оба варианта имеют свои «за» и «против», так, например, если стек освобождает процедура (командой RET число_байтов), то код программы получается меньшим, а если за освобождение стека от параметров отвечает вызывающая функция, то становится возможным вызвать несколько функций с одними и теми же параметрами просто последовательными командами CALL. Первый способ, более строгий, используется при реализации процедур в языке Pascal, а второй, дающий больше возможностей для оптимизации, – в языке C. Разумеется, если передача параметров через стек применяется и для возврата результатов работы процедуры, из стека не надо удалять все параметры, но популярные языки высокого уровня не пользуются этим методом. Кроме того, в языке C параметры помещают в стек в обратном порядке (справа налево), так что становятся возможными функции с изменяемым числом параметров (как, например, printf – первый параметр, считываемый из области аргументов стекового кадра, определяет число остальных параметров).

[4] Передача параметров в потоке кода. В этом необычном методе передаваемые процедуре данные размещаются прямо в коде программы, сразу после команды CALL (как реализована процедура print в одной из стандартных библиотек процедур для ассемблера UCRLIB):

```
call. print
db    "This ASCII-line will be printed."
(следующая команда)
```

Чтобы прочитать параметр, процедура должна использовать его адрес, который автоматически передается в стек как адрес возврата из процедуры. Разумеется, функция должна будет изменить адрес возврата на первый байт после конца переданных параметров перед выполнением команды RET.

Передача параметров в потоке кода, так же как и передача параметров в стеке в обратном порядке (справа налево), позволяет передавать различное число параметров, но этот метод – единственный, позволяющий передать по значению параметр различной длины, что и продемонстрировал этот пример. Доступ к параметрам, переданным в потоке кода, несколько медленнее, чем к параметрам, переданным в регистрах, глобальных переменных или стеке, и примерно совпадает со следующим методом.

Модификация по принимаемому результату. В этом случае необходимо иметь в памяти программ таблицы готовых решений. Возьмём следующий пример. Требуется составить подпрограмму, вычисляющую сложную специальную функцию в зависимости от заданного аргумента. Наиболее быстрое вычисление функции можно получить путём выборки готового значения из заранее посчитанной таблицы.

Пример: синус и косинус можно получить из значений синуса в диапазоне от 0 до 89 градусов. Такая таблица займёт 90 байтов при погрешности 0.4%. Каждый байт будет содержать дробную часть двоичного представления синуса.

180 байтов будут содержать значения синуса через 0.5 градуса. При этом Argument является значением, сдвинутым влево на один разряд.

Sinus:	INC	A		MOV	A, Argument
	MOVC	A, @A+PC		ACALL	Sinus
	RET				
ValueSinus:	DB	0	; = 0		
	DB	00000100b	; = 0.017		
				
	DB	1111111b	; = 0.999		

180 байтов могут содержать значения синуса с большей точностью в 16 разрядов дробной части. Тогда обращение к таблице несколько усложняется для выборки 2 байт.

Можно использовать значения синуса в таблице от 0 до 45 градусов. Это 46 значений для целых отсчётов. Если взять шаг 0.2 градуса, то таблица займёт 230 байт. При вызове аргумент является умноженным на 5 с помощью двух сдвигов и одного сложения.

[6] Передача параметров в блоке параметров. Блок параметров – это (структура) участок памяти, содержащий параметры, так же как и в предыдущем примере, но располагающийся обычно в сегменте данных. Процедура получает адрес начала этого блока при помощи любого метода передачи параметров (в регистре, в переменной, в стеке, в коде или даже в другом блоке параметров). Сам параметр – поле в структуре. В качестве примеров использования этого метода можно назвать многие функции DOS и BIOS, например поиск файла или загрузка (и исполнение) программы.

Блоки параметров разнообразны. Пример – инициализация переменных для текста:

```
char data cdMas[] = "1234567890-=qwertyuiop[]";
char pdata cpMas[] = "asdfghjkl;zxcvbnm,./";
char xdata cxMas[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                      11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
                      21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 };
```

Для осуществления этой инициализации для различных типов памяти поддерживаются следующие управляющие блоки:

```
RSEG ?C_INITSEG ; сегмент для значений инициализации
DB 019H ; 00 – IDATA, 01 – XDATA, 10 – PDATA, 11 – Bit
DB cdMas
DB '1','2','3','4','5','6','7','8','9','0' ; таблица значений
DB '-'','='','q','w','e','r','t','y','u','i'
DB 'o','p','[',']',000H

DB 096H
DB cpMas
DB 'a','s','d','f','g','h','j','k','l',';' ; таблица значений
DB 027H,'z','x','c','v','b','n','m',' ',' '
DB '/',000H

DB 05FH
DW cxMas
DB 001H ; таблица значений
DB 002H
DB 003H
DB 004H
DB 005H
.....
.....
DB 01BH
DB 01CH
DB 01DH
DB 01EH
DB 01FH
```

4. ЗАПУСК ПРОГРАММЫ, НАПИСАННОЙ НА СИ.

В системах микропрограммирования на языке высокого уровня Си требуется стартовый код, который исполняется после **Reset**. В каждой системе стартовый код несколько варьируется, но цель у них одна. (IAR Systems поддерживает большее количество моделей памяти, нежели KEIL ELEKTRONIK GmbH.)

Система микропрограммирования от фирмы KEIL ELEKTRONIK GmbH с компилятором C51 поставляет ряд загрузочных файлов:

START751.A51	2 151	06.09.99	7:34	START751.A51	/ без внешней памяти.
STARTUP.A51	3 838	06.09.99	7:34	STARTUP.A51	/ семейство 80C51.
START_MX.A51	6 627	22.09.00	10:41	START_MX.A51	/ для Philips 80C51MX.

Один из этих файлов надо отредактировать под свои задачи, включить в проект и скомпоновать со своей программой, написанной на языке Си.

4.1 STARTUP.A51 для семейства MCS-51.

Этот файл можно дополнить по своему усмотрению в соответствии с своей архитектурой МК и режимом пуска.

```

;-----
; Поставляется вместе с компилятором C51.
;-----
; STARTUP.A51: этот код выполняется после сигнала reset. Транслируется компилятором A51 в группе загрузки:
;   A51 STARTUP.A51
; Чтобы скомпоновать полученный модифицированный STARTUP.OBJ со своим приложением, используется
; линкер BL51, который вызывается следующей строкой:
;   BL51 <список файлов .obj>, STARTUP.OBJ <опции управления>
; (Интегрированная среда выполняет все эти действия с соответствующими опциями компилятора и линкера.)
;-----
; Инициализация памяти, используемой в программе.
; Определения псевдонимов для последующей инициализации. Задаются для своей архитектуры МК.
;
;                               ; память IDATA всегда начинается с адреса 0.
IDATALEN    EQU    80H        ; размер памяти IDATA в байтах.
;
XDATASTART  EQU    0H        ; адрес начала внешней памяти XDATA.
XDATALEN    EQU    0H        ; размер памяти XDATA в байтах.
;
PDATASTART  EQU    0H        ; адрес начала 0 страницы внешней памяти PDATA.
PDATALEN    EQU    0H        ; размер памяти PDATA в байтах.
;
; Память IDATA включает в себе DATA и BIT области 8051 CPU. Для компилятора C51 допустимо не иметь
; этот регион, т.е. длина =0. Но это самая быстрая память.
;-----
; Инициализация реентерантных стеков.
; Определения псевдонимов для последующей инициализации указателей стеков для реентерантных функций.
; Стек реентерантных функций малой модели памяти SMALL.
IBPSTACK    EQU    0          ; флаг, если 1, стек используется.
IBPSTACKTOP EQU    0FFH+1     ; верхушка стека, location+1.
;
; Стек реентерантных функций большой модели памяти LARGE.
XBPSTACK    EQU    0          ; флаг, если 1, стек используется.
XBPSTACKTOP EQU    0FFFFH+1   ; верхушка стека, location+1.
;
; Стек реентерантных функций компактной модели памяти COMPACT.
PBPSTACK    EQU    0          ; флаг, если 1, стек используется.
PBPSTACKTOP EQU    0FFFFH+1   ; верхушка стека, location+1.
;-----
; Определения для компактной модели 64 Кбайт внешней XDATA ОЗУ.
; Используются переменные класса памяти pdata. Для компоновки нужна опция PPAGE.
PPAGEENABLE EQU    0          ; флаг, если установлен в 1, то используются объекты pdata
PPAGE       EQU    0          ; определяет номер рабочей страницы PPAGE.
;-----

NAME    ?C_STARTUP

?C_C51STARTUP    SEGMENT    CODE    ; объявление сегмента кода для инициализации.
?STACK           SEGMENT    IDATA   ; объявление рабочего сегмента для стека.

RSEG    ?STACK
DS      1                                ; 1 байт рабочего стека.

EXTRN CODE (?C_START) ; начало Си программы из библиотек.
PUBLIC  ?C_STARTUP    ; начало данного инициализатора.

```

```

        CSEG  AT      0                ; абсолютный сегмент кода. (Обойти входы в прерывания.)
?C_STARTUP: LJMP  STARTUP1

        RSEG  ?C_C51STARTUP          ; сегмент инициализации.
STARTUP1:

IF IDATALEN <> 0                      ; инициализация глобальных переменных IDATA.
    MOV     R0, #IDATALEN - 1
    CLR     A
IDATALOOP: MOV     @R0, A
    DJNZ    R0, IDATALOOP
ENDIF

IF XDATALEN <> 0                      ; инициализация глобальных переменных XDATA.
    MOV     DPTR, #XDATASTART
    MOV     R7, #LOW (XDATALEN)
    IF (LOW (XDATALEN)) <> 0
        MOV     R6, # (HIGH XDATALEN) + 1
    ELSE
        MOV     R6, #HIGH (XDATALEN)
    ENDIF
    CLR     A
XDATALOOP: MOVX    @DPTR, A
    INC     DPTR
    DJNZ    R7, XDATALOOP
    DJNZ    R6, XDATALOOP
ENDIF

IF PPAGEENABLE <> 0                  ; включить страницу.
    MOV     P2, #PPAGE
ENDIF

IF PDATALEN <> 0                      ; инициализация глобальных переменных PDATA.
    MOV     R0, #PDATASTART
    MOV     R7, #LOW (PDATALEN)
    CLR     A
PDATALOOP: MOVX    @R0, A
    INC     R0
    DJNZ    R7, PDATALOOP
ENDIF

IF IBPSTACK <> 0                      ; реентерантный стек малой модели.
EXTRN DATA (?C_IBP)

    MOV     ?C_IBP, #LOW IBPSTACKTOP
ENDIF

IF XBPSTACK <> 0                      ; реентерантный стек большой модели.
EXTRN DATA (?C_XBP)

    MOV     ?C_XBP, #HIGH XBPSTACKTOP
    MOV     ?C_XBP+1, #LOW XBPSTACKTOP
ENDIF

IF PBPSTACK <> 0                      ; реентерантный стек компактной модели.
EXTRN DATA (?C_PBP)
    MOV     ?C_PBP, #LOW PBPSTACKTOP
ENDIF

    MOV     SP, #?STACK-1
    LJMP    ?C_START

    END

```


5. ОТЛАДЧИК KEIL ELEKTRONIK GmbH.

Характеристики: 1). Отладчик поддерживает 5 категорий команд. 2). Команды имеют аббревиатуры для ускоренного ввода. 3). Во время ввода команд работает синтаксический анализатор и по первой введённой букве предлагает возможные варианты ввода. 4). Окна отладчика содержат контекстные меню, вызываемые правой кнопкой мыши, в дополнение к основному меню.

Категории команд: 1. команды ведения точек останова; 2. команды общей отладки; 3. команды работы с областями памяти; 4. команды исполнения программы; 5. команды окна наблюдения.

Для ряда команд отладки существуют соответствующие окна диалогов, более наглядные. Встроенный Ассемблер – **Debug – Inline Assembly ...** или в локальном меню дизассемблера. И т.д..

Команды исполнения программы.

7 команд позволяют запускать программу на выполнение с заданной точки и трассировать её.

Команды	Краткое описание
<u>COVERAGE</u>	Выдать обзорную информацию о работе отладчика. (Сколько инструкций отработало.)
<u>Esc</u>	Остановить исполнение программы в отладочном режиме.
<u>GO</u>	Запустить исполнение программы в отладочном режиме.
<u>Performance Analyzer</u>	Работать с анализатором производительности отдельных блоков программы.
<u>PSTEP</u>	Исполняет пошагово одну или несколько строк программы, перешагивая процедуры.
<u>OSTEP</u>	Исполнить машинные инструкции с выходом из текущей функции.
<u>TSTEP</u>	Исполняет пошагово одну или несколько строк программы с входом в процедуры.

Команды работы с областями памяти.

9 команд позволяют визуализировать и/или изменять содержимое области памяти любого класса.

Команды	Краткое описание
<u>ASM</u>	Ассемблирование встроенным ассемблером. Из мнемкода получается исполнительный код.
<u>DEFINE</u>	Определение символического имени, псевдонима.
<u>DISPLAY</u>	Визуализировать содержимое указанной памяти в заданном окне.
<u>ENTER</u>	Ввод значений в указанную область памяти.
<u>EVALUATE</u>	Вычислить выражение и отобразить его результат.
<u>MAP</u>	Указывает параметры доступа для области памяти.
<u>UNASSEMBLE</u>	Дизассемблирует фрагмент программного кода.
<u>WATCHSET</u>	Добавляет переменную в окно наблюдения при отладке.
<u>WATCHKILL</u>	Удаляет все переменные из окна наблюдения.

Команды ведения точек останова.

5 команд обеспечивают обслуживание точек останова в программе. Есть **три типа точек останова**: исполнительные в коде команд, условные точки останова и точки останова по доступу к памяти.

Команды	Краткое описание
<u>BREAKDISABLE</u>	Запрещает одну или более точек останова в процессе отладки.
<u>BREAKENABLE</u>	Разрешает одну или более точек останова в процессе отладки.
<u>BREAKKILL</u>	Удаляет одну или более точек останова для процесса отладки.
<u>BREAKLIST</u>	Распечатывает список всех точек останова с их характеристиками и свойствами.
<u>BREAKSET</u>	Добавляет точку останова для процесса отладки.

Общие команды отладчика.

15 команд дополняют весь сервис при выполнении отладки.

Команды	Краткое описание
<u>ASSIGN</u>	Задать источники ввода/вывода для окна последовательного порта.
<u>DEFINE</u>	Определяет (добавляет) кнопку на панель инструментов.
<u>DIR</u>	Справочная информация о всех символических именах, включая внутренние имена.
<u>EXIT</u>	Выход из режима отладки.
<u>INCLUDE</u>	Выполнить последовательность команд из заданного командного файла.
<u>KILL</u>	Удаляет функции отладчика и кнопки с панели инструментов.
<u>LOAD</u>	Загрузить .obj файл на отладку, или HEX файл или символьную информацию.
<u>LOG</u>	Ведёт журнал выполнения команд отладки.
<u>MODE</u>	Задаёт установки для PC COM портов.
<u>RESET</u>	Сбрасывает отладчик в исходное состояние, переустанавливает доступ к регионам памяти и т.д.
<u>SAVE</u>	Сохраняет содержимое памяти в файле формата Intel HEX386.
<u>SCOPE</u>	Отображает адреса функций в программе и составляющих модулях.
<u>SET</u>	Устанавливает строковое значение предопределённым переменным.
<u>SIGNAL</u>	Отображает список или удаляет сигнальные функции.
<u>SLOG</u>	Обеспечивает ведение файла с содержимым окна последовательного порта.

Команды исполнения программы:

1.1. COVERAGE выдать обзорную информацию о целом приложении. Какие фрагменты кода и сколько отработали на текущий момент. **DETAILS** – что ещё не отработало.

- 1.2. **COVERAGE** \module выдать обзорную информацию о указанном модуле в приложении.
- 1.3. **COVERAGE** \module\func выдать обзорную информацию о функции в указанном модуле.
- 2. **Esc** остановить исполнение программы в отладочном режиме.
- 3. **GO** startaddr, stopaddr старт программы с указанного адреса startaddr, если есть, и останов в точке stopaddr, если таковая указана. Если старт не указан, то выполнение с текущего РС.
- 4.1. **PerformanceAnalyze** отображает минимальное, максимальное и среднее время исполнения заданных блоков кода на текущий момент.
- 4.2. **PerformanceAnalyze** start, end определяет блок кода для анализатора производительности.
- 4.3. **PerformanceAnalyze KILL** * удаляет все блоки кода из анализатора производительности.
- 4.4. **PerformanceAnalyze KILL** item, item... удаляет лишь только указанные блоки кода из анализатора производительности.
- 4.5. **PerformanceAnalyze RESET** сбрасывает анализатор производительности в исходное состояние.
- 5. **PSTEP** expression выполняет указанное число строк программы, перешагивая процедуры.
- 6. **QSTEP** выполняет инструкции с выходом из текущей функции; останов после вызова.
- 7. **TSTEP** expression пошагово выполняет указанное число строк программы.

Команды работы с областями памяти:

- 1.1. **ASM** отображает текущий адрес встроенного ассемблирования.
- 1.2. **ASM** start address устанавливает новый адрес для встроенного ассемблирования.
- 1.3. **ASM** instruction ассемблирует указанную инструкцию в текущем месте кода.
- 2. **DEFINE** type identifier определяет рабочее символическое имя указанного типа. Типом могут быть: 1). CHAR – знаковый символ 1 байт; 2). DOUBLE – число с плавающей запятой, двойной точности, 8 байт; 3). FLOAT – число с плавающей запятой, 4 байта; 4). INT – знаковое целое число 2 байта; 5). LONG – длинное знаковое целое число 4 байта.
- 3. **DISPLAY** startaddr, endaddr выдаёт дампы указанной области памяти в различных представлениях (через контекстное меню). Если начало не указано, то с адреса от предыдущей команды или с 0 вначале. Если конечный адрес не указан, то отображается до ближайшей 256-байтовой границы. Адреса задаются с соответствующим префиксом класса памяти. Так >D main выводит байты основной процедуры; >D X:0,0x100 выводит 256 байт из внешней памяти.
- 4. **ENTER** type address = expr, expr ... вводит данные заданного типа с указанного адреса. Типом могут быть: 1). CHAR – знаковый или беззнаковый символ 1 байт; 2). DOUBLE – число с плавающей запятой, двойной точности, 8 байт; 3). FLOAT – число с плавающей запятой, 4 байта; 4). INT – знаковое или беззнаковое целое число 2 байта; 5). LONG – длинное знаковое или беззнаковое целое число 4 байта. Так >E CHAR x:0 = 1,2,"-µVision2-" вводит 12 байт.
- 5. **EVALUATE** expression вычисляет выражение в 10-й, 8-й и 16-й системах счисления и в ASCII формате. Без ключевого слова вычисления выполняются в текущей системе счисления (переменная RADIX).
- 6.1. **MAP** отображает текущую карту памяти для отлаживаемой программы.
- 6.2. **MAP** start, end READWRITEEXECVNM задаёт права доступа для указанного региона памяти.
- 6.3. **MAP** start, end CLEAR удаляет указанный регион памяти из текущей структуры памяти программы.
- 7. **UNASSEMBLE** address дизассемблирует область памяти, начиная с указанного адреса. Если адреса нет, то дизассемблируется область от предыдущего РС.
- 8. **WATCHKILL** windownr удаляет все переменные из указанного окна наблюдения (их до четырёх).
- 9. **WATCHSET** windownr, expression, base добавляет в указанное окно наблюдения переменную с указанием системы счисления (10 или 16) её отображения.

Команды ведения точек останова:

- 1.1. **BREAKDISABLE** number, number... запретить лишь указанные точки останова.
- 1.2. **BREAKDISABLE** * запретить все точки останова.
- 2.1. **BREAKENABLE** number, number... разрешить лишь указанные точки останова.
- 2.2. **BREAKENABLE** * разрешить все точки останова.
- 3.1. **BREAKKILL** number, number... удалить лишь указанные точки останова.
- 3.2. **BREAKKILL** * удалить все точки останова.
- 4. **BREAKLIST** выводит список всех точек останова с их характеристиками: номер точки (от 0 для ссылки в командах); тип останова (исполняемая точка **E**; условная точка **C**; точка доступа **A**, где доступ только по чтению – **RD**, только по записи – **WR**, по чтению и записи – **RW**); выражение останова (условие останова для точки); счётчик останова (пропускать события останова столько раз, чтобы остановиться на указанном; по умолчанию – 1); флаг состояния (активна – **enabled** или пассивна – **disabled**); команда для события данной точки (если указана, то останова не будет, а выполнится данная команда; если не указана – будет останова). Пример: >BL
0: (E C: 0xFF01EF) 'main', CNT=1, enabled
- 5.1. **BREAKSET** exp, cnt, "cmd" задать исполнительную или условную точку останова. Если выражение приводится к адресу, то получается исполнительная точка останова. Если выражение не является только ад-

ресом, то будет условная точка останова. В выражении допустимы только операции &, &&, <, <=, >, >=, ==, и !=. Пример: >BS main исполнительная точка, а >BS ind != 8 условная точка.

5.2. BREAKSET READ exp, cnt, "cmd"

BREAKSET WRITE exp, cnt, "cmd"

BREAKSET READWRITE exp, cnt, "cmd" задаёт соответствующие точки доступа с указанными правами.

Например: >BS READ interval == 3

ДОПОЛНЕНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ №1

Вычисление Дискретного Преобразования Фурье действительных последовательностей:

// 1-я программа ДПФ действительных последовательностей

//===== константы в коде

#pragma CODE

#include <math.h>

#define M_PI 3.14159265358979323846

#define M_PI_2 1.57079632679489661923

#define M_PI_4 0.785398163397448309616

// максимальный размер

#define N 128

#define Nm1 (N - 1)

#define Nd2 (N >> 1)

#define Nd4 (Nd2 >> 1)

static float code fMasCos[Nd4 + 1] = { /* COS for 128 */

1.0, 0.9987954562, 0.9951847267, 0.98917651,
0.9807852804, 0.9700312532, 0.9569403357, 0.9415440652,
0.9238795325, 0.9039892931, 0.8819212643, 0.85772861,
0.8314696123, 0.8032075315, 0.7730104534, 0.7409511254,
0.7071067812, 0.6715589548, 0.6343932842, 0.5956993045,
0.555570233, 0.5141027442, 0.4713967368, 0.4275550934,
0.3826834324, 0.3368898534, 0.2902846773, 0.2429801799,
0.195090322, 0.1467304745, 0.09801714033, 0.04906767433,
0.0 };

unsigned char n;

unsigned char nm1, nd2, nd4; // резерв

float xdata SReal[N];

//float xdata SImag[N];

float xdata SReal_out[N];

float xdata SImag_out[N];

//=====

// случай любой n <= N, отдельно 0 индекс.

#define maskai (~(Nd2 | Nd4))

void FunR_DPF(void) {

unsigned char i, j, ij;

unsigned char is, dis;

unsigned char k, wdi, di = N / n;

float WorkR, WorkI, fCos, fSin;

// постоянная

SImag_out[0] = 0;

WorkR = 0.0;

for (i = 0 ; i < n ; i++)

WorkR += SReal[i];

SReal_out[0] = WorkR;

// остальные гармоники

for (i = 1, wdi = di; i < n ; i++, wdi += di) {

WorkR = 0.0;

WorkI = 0.0;

for (j = 0, ij = 0 ; j < n ; j++, ij = (ij + wdi) & Nm1) {

is = ij & maskai;

dis = Nd4 - is;

Пример 1

if (!(k = ij ^ is)) {

fCos = fMasCos[is];

fSin = -fMasCos[dis];

} else if (k == Nd4) {

fCos = -fMasCos[dis];

fSin = -fMasCos[is];

} else if (k == Nd2) {

fCos = -fMasCos[is];

fSin = fMasCos[dis];

} else {

fCos = fMasCos[dis];

fSin = fMasCos[is];

}

WorkR += SReal[j] * fCos; // cos

WorkI += SReal[j] * fSin; // -sin

}

SReal_out[i] = WorkR;

SImag_out[i] = WorkI;

}

}

***** main *****

void main(){

unsigned char i;

n = N;

nm1 = n - 1;

#if N != 256

nd2 = n >> 1;

nd4 = nd2 >> 1;

#else

nd2 = 0x80;

nd4 = 0x40;

#endif

// входной сигнал

// for (i = 0; i <= Nm1; i++)

for (i = 0; i < N; i++)

SReal[i] = 0;

SReal[1] = 1.0;

FunR_DPF();

n >>= 1;

FunR_DPF(); // 1 волна

SReal[1] = 0;

SReal[3] = 1.0;

FunR_DPF(); // 3 волны

// ...

n >>= 1;

FunR_DPF();

while(1){};

}

Часть 1.

1. Реализовать вычисление ДПФ оптимальным по скорости выполнения.

2. Реализовать вычисление ДПФ с байтовыми индексами для размера 256 отсчётов включительно.

3. Реализовать вычисление ДПФ для размеров последовательностей до 4096 отсчётов включительно.

Часть 2.

1. Реализовать вычисление обратного ДПФ оптимальным по скорости выполнения с байтовыми индексами для размера 256 отсчётов включительно.
2. Реализовать вычисление ДПФ для комплексных входных последовательностей оптимальным по скорости выполнения произвольных размеров.