Министерство образования и науки Российской Федерации
Муромский институт (филиал)
федерального государственного бюджетного образовательного учреждения
высшего образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(МИ ВлГУ)**

Отделение среднего профессионального образования

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ ПО ДИСЦИПЛИНЕ АРХИТЕРТУРА МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВ

для студентов специальности

11.02.01 Радиоаппаратостроение

Программа подготовки специалистов среднего звена

Составитель
Романов Д.Н.

Муром 2018

Содержание

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ.

1 Обзор эмулятора 8085 Simulator IDE

Главное окно

8085 Simulator IDE представляет собой полноценный программный эмулятор процессора Intel 8085. На рисунке 1 изображено главное окно эмулятора.

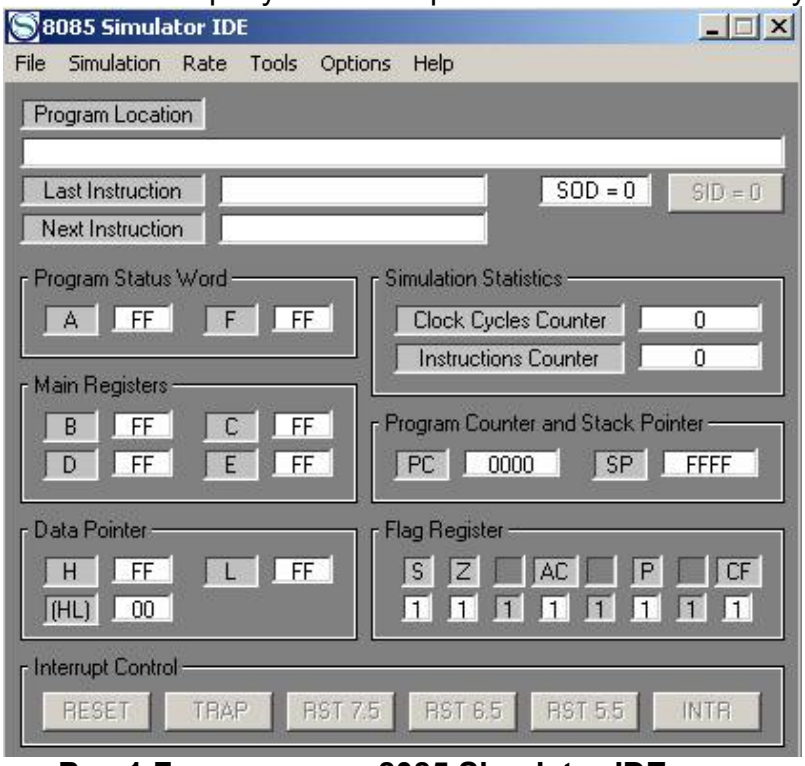


Рис.1 Главное окно 8085 Simulator IDE

Главное окно состоит из следующих элементов.

Program Location – расположение на диске файла с программой написанной на языке ассемблера.

Last Instruction – последняя выполненная команда программы

Next Instruction – следующая выполняемая команда.

Simulation Statistics – включает в себя счетчик тактов, за которое выполняется команды и счетчик выполненных команд.

Program Counter and Stack Pointer – включает в себя счетчик команд PC (в нем хранится адрес следующей команды) и указатель стека.

Flag Register – регистр флагов включающий: флаг знака (S); флаг нулевого результата (Z); флаг дополнительного переноса при арифметических операциях (AC); флаг четности результата (P); флаг переноса при арифметических операциях (CF)

Program Status Word, Main Registers, Data Pointer – программно доступные регистры.

Interrupt Control – управление прерываниями.

Главное меню

Меню **File** содержит

✓ Пункт **Clear Memory** – восстанавливает начальное состояние 64 КБ памяти 64 КБ с адресным диапазоном 0000-FFFF(H). Вся память заполняется командами NOP с кодом операции 00.

✓ Пункт **Load Memory** – загружает программный файл в память центрального процессора. Программный файл должен быть либо в формате HEX, в дво-

ичном формате (с расширением *.OBJ) памяти, начинающейся с 0000-ого адреса, с максимальной длиной 64 КБ. Файл может быть сгенерирован внутренним ассемблером или с использованием TASM.

✓ Пункт **Save Memory** – сохраняет содержимое памяти в файл.

Меню **Simulation** содержит:

✓ Пункт **Start** – начинает выполнение команд, начиная с адреса в памяти, которое соответствует стартовому адресу. Значение по умолчанию для этого параметра является 0000, и это значение может быть изменено, при помощи команды **Change Starting Address** в меню **Options.**

✓ Пункт **Step** – выполняет очередную команду программу при нажатии на клавишу F2. Работает только при использовании режима **Step By Step.**

✓ Пункт **Stop** – выходит из режима моделирования и представляет информацию об общем количестве выполненных команд, продолжительность моделирования и продолжительности в реальном времени моделирования в тактовых циклах.

Меню **Rate** дает возможность пользователю изменить скорость моделирования и содержит:

✓ Пункт **Step By Step** – интервал между командами определяется пользователем. Когда эмулятор находится в режиме Step By Step, имеется возможность значения всех внутренних регистров центрального процессора, нажимая на соответствующие поля на интерфейсе программы. Когда выбран данный способ запуска программы, в меню **Simulation** становится доступным пункт **Step**.

✓ Пункт **Slow** – интервал между командами составляет 1500мс.

✓ Пункт **Normal** – интервал между командами составляет 250мс.

✓ Пункт **Fast** – интервал между командами составляет 50мс.

✓ Пункт **Extremely Fast** – интервал максимально короткий и линейно зависит от скорости компьютера.

✓ Пункт **Ultimate (No Refresh)** – основное окно эмулятор не обновляется, что значительно увеличивает скорость моделирования. Этот пункт должен использоваться в комбинации с пунктом **Breakpoints Manager** (Менеджер Контрольных точек), для моделирования длинных задержек подпрограммы, с наиболее высокой скоростью.

Меню **Tools** содержит:

✓ Пункт **Memory Editor** – определяет доступ к графическому интерфейсу для 64 КБ памяти центрального процессора эмулятора. Значение каждой ячейки памяти может быть изменено, при помощи клика мышки на соответствующую ячейку. Введенное значение подтверждается, нажатием клавиши ENTER, при этом окно редактирования автоматически передвигается в следующую ячейку памяти.

✓ Пункт **Disassembler** – 8085 Simulator IDE имеет очень мощный внутренний деассемблер. Деассемблер независим от эмулятора, и имеет собственную память программы. Для начала работы необходимо загрузить программный файл (*.hex-файл или *.obj-файл) в память деассемблера. Процесс деассемблирования инициализируется соответствующей командой. Деассемблирование будет всегда начинаться с 0000-ого адреса. Результатом будет программа написанная на языке ассемблера.

✓ Пункт **Peripheral Devices** – Инструмент, для контроля и управления командами IN и OUT. Определены четыре основные устройства для ввода/вывода данных, и один терминал вывода, для, чтобы выводов символов ASCII, посланные одному из портов. Байты, посланные командой OUT отображены, графически показывая конкретные биты. Если устройство сконфигуриро-

вано как устройство ввода данных, его значение можно задать в окне значений или переключая включая биты графического интерфейса.

✓ Пункт **I/O Ports Editor** –доступ к графическому интерфейсу для управления диапазоном портов ввода/вывода. Значение определенного порта ввода/вывода может быть изменено. Информация о портах ввода/вывода в пункте **Ports Editor** совместима с пунктом **Peripheral Devices**.

✓ Пункт **External Modules** – Этот инструмент используется для установки автоматического интерфейс с внешними клиент-серверными модулями. Для подключения необходимо ввести имя класса, в форме ApplicationName. Внешние клиент-серверные приложения стартуют и закрываются автоматически с 8085 Simulator IDE.

✓ Пункт **Assembler** – команда запускает интегрированный ассемблер. Исходные файлы ассемблера могут быть отредактированы, откомпилированы и загружены в память в той же самой графической среде. После успешной трансляции генерируется два новых файла. Один с расширением *.OBJ, который является программой в двоичном коде и может быть загружен в память центрального процессора, второй с расширением *.lst, которое является листингом ассемблера, используемого отладчиком.

✓ Пункт **Breakpoints Manager** – команда запускает интегрированный отладчик, который используется, для отладки и контроля выполнения программы. Если листинга программы, находящейся в памяти, не существует, альтернативный листинг сгенерирован внутренним дизассемблером. Возможно определение до 10 контрольных точек, нажатием на определенные строки в загруженной программе. Если моделирование начинается в более быстрых режимах, то оно автоматически переключается в режим **Step By Step** при достижении любой из этих контрольных точек. Контрольные точки отмечены красными кругами, и текущее значение регистра PC отмечено желтой стрелкой. Есть опция, чтобы сохранить указатель PC в центре в течение моделирования.

✓ Пункт **BASIC Compiler** – команда запускает интегрированный компилятор языка высокого уровня БЕЙСИК.

✓ Пункт **Simulation Log Viewer** – запуск интегрированного графического инструмента, который регистрирует все исполняемые команды, содержимое регистров и состояние регистра флагов.

✓ Пункт **Interactive Assembler Editor** – запуск интегрированного графического инструмента, который дает возможность новичкам записывать свои первые программы ассемблера в интерактивном режиме, без необходимости запоминания мнемоники индивидуальных команд.

Меню **Options** содержит:

✓ Пункт **Enable Logging** – Принудительная регистрация в файле LOG.TXT все исполняемые команды вместе с содержимым регистров и состоянием флагов. Эта опция не пересекается с интегрированным **Simulation Log Viewer.**

✓ Пункт **HLT Stops Simulation** – Если эта опция выбрана, исполнение программы автоматически остановится, при достижении команды *HLT*. Если опция не выбрана, эмулятор, также, как реальные процессоры 8085, выполнит эту команду неоднократно, пока не получает прерывание. После возвращения от прерывания выполнение продолжится со следующей команды.

✓ Пункт **FF Power On Defaults** – Переключение этой опции переключает начальные значения внутренних регистров с 00H на FFH и наоборот.

✓ Пункт **Refresh Memory Editor** – Если эта опция выбрана и **Memory Editor**, открыт, отображенный диапазон памяти будет обновляться после каждой исполняемой команды, во всех режимах скорости. Это необходимо, для кон-

троля содержимого стека или в другом месте памяти в течение исполнения программы.

✓ Пункт **Refresh Breakpoints Manager** – Если эта опция выбрана и **Breakpoints Manager**, открыт, то он будет обновляться после каждой исполняемой команды.

✓ Пункт **Save Positions** – Если эта опция выбрана, позиции окон на экране будут запомнены.

✓ Пункт **Save Positions** – команда позволяет изменять параметр генератора тактовой частоты, использующегося для вычисления времени исполнения команды. Значение по умолчанию – 4 МГц.

✓ Пункт **Change Starting Address** – команда позволяет изменять стартовый адрес для исполнения программы. Введенное значение запоминается для будущих сессий. Значение по умолчанию – 0000.

✓ Пункт **Prompt For Value Before IN Instruction** – опция вынудит программу всегда запрашивать пользователя вручную вводить все входящие байты во всех портах. Если опция отключена, значения будут взяты из **Peripheral Devices** или **I/O Ports Editor.**

✓ Пункт **Enable IN/OUT Instructions Logging** – Если эта опция выбрана, программа регистрирует все команды IN и OUT в файле IO.TXT, расположенном папке приложения. Каждая команда IN или OUT добавит новую строку в этом файле.

✓ Пункт **Show Confirmation Boxes** – Если эта опция выбрана, после выполнения программы будет показываться окно результатов.

✓ Пункт **Use Simple Editor in Basic Compiler and Assembler** – Если эта опция выбрана, основные окна редактора ассемблера и компилятора BASIC будут запускаться в упрощенном режимах.

✓ Пункт **Change Color Theme** – команда открывает список доступных цветных тем.

2 Система команд процессора 8085.

Все команды ассемблера 8085 можно разделить на несколько групп:

✓ Команды пересылки данных;
✓ Арифметические команды:
✓ Логические команды;
✓ Команды условного и безусловного перехода;
✓ Команды вызова подпрограмм и возврата из подпрограмм;
✓ Специальные команды.

Команды пересылки.

Пересылка данных между регистрами

Команды пересылки данных между регистрами кодируются в одном байте следующим образом: 01 DDD SSS, где DDD - номер регистра назначения; SSS - номер регистра приемника. Соответственно 01 - код операции пересылки. Никаких флагов команды пересылки не устанавливают. На выполнение команды тратится один машинный цикл.

Пересылка из ячейки памяти в регистр и из регистра в память осуществляется с помощью косвенно регистровой адресации. Это означает, что адрес ячейки памяти загружается в регистровую пару HL, а в командах типа MOV A,M в регистр A будет загружено содержимое ячейки памяти, адрес которой содержится в HL. Так как в таких командах требуется обращение к памяти, то на их выполнение нужно два машинных цикла.

Очень полезна группа команд LXI непосредственной загрузки регистровых пар непосредственным значением. Она позволяет одной командой переместить сразу два байта данных и широко используется программистами как в операциях адресной арифметики, так и при выполнении целочисленных вычислений.

Команда кодируется так: 00RP0001 xxxxxxxx zzzzzzzz, где RP - регистровая пара; xxxxxxxx - младший байт данных, zzzzzzzzz - старший байт данных.

При исполнении команды, требующей трех машинных циклов, старший байт данных грузится в старший регистр регистровой пары, а младший байт - в младший регистр. Название старшего регистра стоит в названии пары первым.

Команда прямой загрузки аккумулятора позволяет загрузить в него данные, на которые указывает адрес, содержащийся в самой команде.

Длина команды три байта. Ее код: 001110010 xxxxxxxx zzzzzzzz, где xxxxxxxx - младшая часть адреса; zzzzzzzz - старшая часть адреса.

Исполнение занимает четыре машинных цикла.

Симметричная по действию команда STA.

Команда LHLD addr (мнемоника расшифровывается Load H and L Direct) загружает в L содержимое ячейки памяти по адресу, кодируемому во втором и третьем байтах команды (т. е. адресация прямая). В H загружается байт из ячейки addr+1.

Команда выполняется за пять машинных циклов. Обратная ей по действию команда SHLD (Store H and L Direct).

Команда LDAX reg (мнемоника от Load accumulator indirect). Содержимое ячейки памяти, адресуемой регистровой парой BC или DE, за два цикла загружается в аккумулятор.

Обратная по действию команда STAX reg.

Арифметические команды

Сложение.

Выполняется за один цикл. Обратите внимание, что все арифметические команды изменяют флаги: Z,S,P,CY,AC.

Для выполнения многобайтового сложения необходимо учитывать перенос. Поэтому младшие байты слагаемых складываются с помощью команды ADD, а все последующие в помощью команды ADC reg. Она учитывает при сложении содержимое флага переноса. Аналогично устроены команды вычитания.

Разновидностью команды сложения является команда инкремента регистра, необходимая для адресной арифметики и организации циклов.

INR reg. Код: 00ККК100. Устанавливаются все флаги, кроме CY.

Обратная по действию команда DCR reg.

Довольно редко используется команда DAA (мнемоника от Decimal Adjust Accumulator) выполняет следующие действия: если содержимое младшего полубайта (нибла) аккумулятора меньше 9 или установлен флаг CY, то (A) + 6, если значение старшего полубайта больше 9 или установлен флаг CY, то 6 добавляется к содержимому старшего полубайта.

При этом устанавливаются все флаги.

Логические команды

ANA reg

Флаг CY сбрасывается, а AC устанавливается

В командах групп XRA и ORA флаги CY и AC сбрасываются.

CMP reg - сравнить регистр (Compare register), вычитает содержимое регистра из аккумулятора. Содержимое аккумулятора не изменяется. Флаги устанавливаются как при вычитании. Z=1, если (A) = (reg). CR = 1, если (A) < (reg).

Циклический сдвиг влево RCL работает следующим образом. Содержимое аккумулятора сдвигается на одну позицию влево, т. е. каждый старший бит получает значение стоящего рядом с ним младшего бита. Содержимое седьмого бита переходит в нулевой бит аккумулятора:

(CY)←(b7).

RRC - сдвигает содержимое аккумулятора вправо. (CY)←{b0}, (b7)←(b0).

RAL - (Rotate Left trough Carry). Сначала (CY) заносится в младший бит аккумулятора, а потом в CY записывается содержимое старшего его бита.

Формально: (b0)←(CY), (CY)←(b7).

Также и со сдвигом вправо: (Bn)←(Bn+1), (CY)←(b0), (b7)←(CY).

Команда CMA (мнемоника от Complement Accumulator) инвертирует каждый бит аккумулятора, т. е. 0 становится 1 и наоборот. Флаги не устанавливаются.

Команды переходов.

Команды переходов предназначены для организации всевозможных циклов, ветвлений, вызовов подпрограмм и т.д., то есть они нарушают последовательный ход выполнения программы. Эти команды записывают в регистр-счетчик команд новое значение и тем самым вызывают переход процессора не к следующей по порядку команде, а к любой другой команде в памяти программ.

Некоторые команды переходов предусматривают в дальнейшем возврат назад, в точку, из которой был сделан переход, другие не предусматривают этого. Если возврат предусмотрен, то текущие параметры процессора сохраняются в стеке. Если возврат не предусмотрен, то текущие параметры процессора не сохраняются.

Команды переходов без возврата делятся на две группы:

- ✓ команды безусловных переходов;
- ✓ команды условных переходов.

В обозначениях этих команд используются слова Branch (ветвление) и Jump (прыжок).

Команды безусловных переходов вызывают переход в новый адрес независимо ни от чего. Они могут вызывать переход на указанную величину смещения (вперед или назад) или же на указанный адрес памяти. Величина смещения или новое значение адреса указываются в качестве входного операнда.

Команды условных переходов вызывают переход не всегда, а только при выполнении заданных условий. В качестве таких условий обычно выступают значения флагов в регистре состояния процессора (PSW). То есть условием перехода является результат предыдущей операции, меняющей значения флагов. Всего таких условий перехода может быть от 4 до 16. Несколько примеров команд условных переходов:

- ✓ переход, если равно нулю;
- ✓ переход, если не равно нулю;
- ✓ переход, если есть переполнение;
- ✓ переход, если нет переполнения;
- ✓ переход, если больше нуля;
- ✓ переход, если меньше или равно нулю.

Если условие перехода выполняется, то производится загрузка в регистр-счетчик команд нового значения. Если же условие перехода не выполняется, счетчик команд просто наращивается, и процессор выбирает и выполняет следующую по порядку команду.

Специально для проверки условий перехода применяется команда сравнения (CMP), предшествующая команде условного перехода (или даже нескольким командам условных переходов). Но флаги могут устанавливаться и любой другой командой, например командой пересылки или любой арифметической или логической командой.

Отметим, что сами команды переходов флаги не меняют, что как раз и позволяет ставить несколько команд переходов одну за другой. Совместное использование нескольких команд условных и безусловных переходов позволяет процессору выполнять разветвленные алгоритмы любой сложности.

Для выполнения подпрограмм, то есть вспомогательных программ используются команды переходов с дальнейшим возвратом в точку, из которой был произведен переход. Эти команды называются также командами вызова подпрограмм (распространенное название – CALL). Использование подпрограмм позволяет упростить структуру основной программы, сделать ее более логичной, гибкой, легкой для написания и отладки. В то же время надо учитывать, что широкое использование подпрограмм, как правило, увеличивает время выполнения программы.

Все команды переходов с возвратом предполагают безусловный переход (они не проверяют никаких флагов). При этом они требуют одного входного операнда, который может указывать как абсолютное значение нового адреса, так и смещение, складываемое с текущим значением адреса. Текущее значение счетчика команд (текущий адрес) сохраняется перед выполнением перехода в стеке.

Для обратного возврата в точку вызова подпрограммы (точку перехода) используется специальная команда возврата (RET или RTS). Эта команда извлекает из стека значение адреса команды перехода и записывает его в регистр-счетчик команд.

Полный справочник команд процессора Intel 8085 находится в приложении 1.

3 Работа с эмулятором процессора Intel 8085

Для примера работы с эмулятором напишем и запустим следующую программу.

1. Переслать в аккумулятор число 01.
2. Переслать в регистр B число 02.
3. Сложить содержимое аккумулятора и регистра B.
4. Переслать результат во внешнее устройство, связанное с портом 01.

Алгоритм выполнения

1. Открыть эмулятор.
2. Открыть редактор ассемблера (**Tools→Assembler**).
3. Написать в редакторе программу, пользуясь приведенным справочником команд:

   ```
   MVI A,01H // Пересылка в аккумулятор 16-ричного числа 01.
   MVI  B,02H // Пересылка в регистр B 16-ричного числа 02.
   ADD  B // Сложение содержимого аккумулятора и регистра B, результат сохраняется в аккумуляторе.
   OUT  01 // Пересылка содержимого аккумулятора в порт 01.
   ```

4. Сохранить написанную программу (**File→Save**).
5. Транслировать написанную программу (**Tools→Assemble**). Если программа написана без ошибок, то внизу редактора ассемблера появится

надпись **Number of errors=0**. Эмулятор создаст бинарный файл с именем, аналогичным сохраненному, и с расширением *.obj.

6. Закрыть редактор ассемблера.
7. В основном окне программы открыть созданный бинарный файл (**File→Load Program**).
8. Выбрать пошаговый режим исполнения программы (**Rate→Step by Step**) .
9. Открыть окно периферийных устройств (**Tools→Peripheral Devices**).
10. На любом из имеющихся 4 портов нажать кнопку **OFF** для включения порта. В появившемся окне написать номер используемого порта – 01.
11. Включить обязательную запись в LOG файл.
12. Запустить исполнение программы (**Simulation→Start**)
13. Выполнить все имеющиеся команды (**Simulation→Step**)
14. После выполнения последней команды остановить исполнение (**Simulation→Stop**).
15. На экране появится LOG файл с результатом исполнения. **Сохранить для отчета.**

Лабораторная работа № 1

ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕР 8085

Цель работы: Изучить принципы программирования при помощи арифметических, логических команд и команд пересылки данных.

Пример.

Вычислить значение выражения: $\overline{2x \vee 3y}$ при x=05 и y=06. Значение ***x*** должно находиться в ячейке памяти **0101h**, значение ***y*** должно находиться в ячейке памяти **0102**h, результат переслать в ячейку памяти **0A00**h.

PC	Мнемоника	Примечание
0000	LXI H,0101H	// пересылка в регистровую пару **HL** адреса ячейки памяти **0101**h
0003	MVI M,05H	// пересылка в ячейку, адрес которой лежит в регистровой паре **HL**, значения **05**h
0005	INX H	// увеличение адреса ячейки памяти в **HL** на 1 (**0102**h)
0006	MVI M,06H	// пересылка в ячейку, адрес которой лежит в регистровой паре **HL**, значения **05**h
0008	DCX H	// уменьшение адреса ячейки памяти в **HL** на 1 (**0101**h)
0009	ADD M	// Поскольку в системе команд процессора 8085 нет команды умножения, то операция 2****x*** выполняется сложением (**A+M+M**)
000A	ADD M	
000B	MOV M,A	// пересылка в ячейку, адрес которой лежит в регистровой паре **HL**, значения **A**
000C	INX H	// увеличение адреса ячейки памяти в **HL** на 1 (**0102**h)
000D	MVI A,00H	// обнуление **A**
000F	ADD M	// умножение значения в ячейке памяти **0102**h на 3. Результат остается в **A**
0010	ADD M	
0011	ADD M	
0012	DCX H	// уменьшение адреса ячейки памяти в **HL** на 1 (**0101**h)
0013	ORA M	// Логическое сложение содержимого **A** и ячейки памяти **0101**h
0014	CMA	// Инверсия содержимого **A**
0015	STA 00A0H	// Пересылка содержимого аккумулятора в ячейку памяти **00A0**h
0018	HLT	// Остановка программы

Задания для выполнения.

1. Вычислить значение выражения: $\overline{2 \wedge x + 2 \vee y}$ при x=01 и y=02. Значение ***x*** должно находиться в ячейке памяти **0010**h, значение ***y*** должно находиться в ячейке памяти **0100**h, результат переслать в ячейку памяти **0110**h.

2. Вычислить значение выражения: $\overline{x} \wedge y - \overline{y} \vee x$ при x=10 и y=20. Значение ***x*** должно находиться в ячейке памяти **01A0**h, значение ***y*** должно находиться в ячейке памяти **00B0**h, результат переслать в ячейку памяти **01C0**h.

3. Вычислить значение выражения: $x \cdot 5 + \overline{y} \vee 2$ при x=03 и y=04. Значение ***x*** должно находиться в ячейке памяти **1000**h, значение ***y*** должно находиться в ячейке памяти **1001**h, результат переслать в ячейку памяти **1010**h.

4. Вычислить значение выражения: $\overline{2 \cdot x \wedge 2 \cdot y}$ при x=05 и y=06. Значение ***x*** должно находиться в ячейке памяти **0250**h, значение ***y*** должно находиться в ячейке памяти **0350**h, результат переслать в ячейку памяти **0300**h.

5. Вычислить значение выражения: $(4 + x) \vee \overline{x - y}$ при x=07 и y=08. Значение ***x*** должно находиться в ячейке памяти **0111h**, значение ***y*** должно находиться в ячейке памяти **0122**h, результат переслать в ячейку памяти **0A10**h.

6. Вычислить значение выражения: $2 \wedge \overline{x} + 2 \vee \overline{y}$ при x=04 и y=02. Значение ***x*** должно находиться в ячейке памяти **0014**h, значение ***y*** должно находиться в ячейке памяти **0200**h, результат переслать в ячейку памяти **011A**h

7. Вычислить значение выражения: $(5 - x) \vee x - \overline{y}$ при x=07 и y=04. Значение ***x*** должно находиться в ячейке памяти **0117h**, значение ***y*** должно находиться в ячейке памяти **0102**h, результат переслать в ячейку памяти **0A1A**h.

8. Вычислить значение выражения: $2 + \overline{x \wedge 2 \cdot \overline{y}}$ при x=09 и y=06. Значение ***x*** должно находиться в ячейке памяти **025F**h, значение ***y*** должно находиться в ячейке памяти **0350**h, результат переслать в ячейку памяти **003D**h.

9. Вычислить значение выражения: $\overline{x \cdot 5 - \overline{y} \vee 2}$ при x=08 и y=04. Значение ***x*** должно находиться в ячейке памяти **1050**h, значение ***y*** должно находиться в ячейке памяти **1009**h, результат переслать в ячейку памяти **1070**h.

10. Вычислить значение выражения: $3 \cdot \overline{x} - \overline{y} \vee 6$ при x=08 и y=03. Значение ***x*** должно находиться в ячейке памяти **1D00**h, значение ***y*** должно находиться в ячейке памяти **10AA**h, результат переслать в ячейку памяти **101C**h.

11. Вычислить значение выражения: $\overline{\overline{x} \wedge y + \overline{y} \vee x}$ при x=10 и y=20. Значение ***x*** должно находиться в ячейке памяти **01AF**h, значение ***y*** должно находиться в ячейке памяти **01BF**h, результат переслать в ячейку памяти **01CF**h.

12. Вычислить значение выражения: $\overline{\overline{x + y - y}} \vee x$ при x=10 и y=0A. Значение ***x*** должно находиться в ячейке памяти **02AC**h, значение ***y*** должно находиться в ячейке памяти **0CB0**h, результат переслать в ячейку памяти **02DF**h.

13. Вычислить значение выражения: $\overline{2 \cdot \overline{x} \wedge 2 + y}$ при x=2F и y=06. Значение ***x*** должно находиться в ячейке памяти **0256**h, значение ***y*** должно находиться в ячейке памяти **0356**h, результат переслать в ячейку памяти **03AA**h.

14. Вычислить значение выражения: $(9 - x) \vee \overline{\overline{x - y}}$ при x=03 и y=07. Значение ***x*** должно находиться в ячейке памяти **01B1h**, значение ***y*** должно находиться в ячейке памяти **01B2**h, результат переслать в ячейку памяти **0A1B**h.

15. Вычислить значение выражения: $5 \wedge \overline{\overline{x}+2} \vee \overline{y}$ при x=04 и y=07. Значение ***x*** должно находиться в ячейке памяти **0F14**h, значение ***y*** должно находиться в ячейке памяти **020F**h, результат переслать в ячейку памяти **01FA**h

16. Вычислить значение выражения: $\overline{(5+y)} \vee x - \overline{y}$ при x=07 и y=14. Значение ***x*** должно находиться в ячейке памяти **0157h**, значение ***y*** должно находиться в ячейке памяти **0105**h, результат переслать в ячейку памяти **0A1F**h.

17. Вычислить значение выражения: $(2+x) \wedge (2 \vee \overline{y})$ при x=0D и y=06. Значение ***x*** должно находиться в ячейке памяти **045F**h, значение ***y*** должно находиться в ячейке памяти **0358**h, результат переслать в ячейку памяти **053D**h.

18. Вычислить значение выражения: $\overline{2 \vee x + \overline{x} \vee y}$ при x=04 и y=02. Значение ***x*** должно находиться в ячейке памяти **0214**h, значение ***y*** должно находиться в ячейке памяти **0160**h, результат переслать в ячейку памяти **0113**h.

19. Вычислить значение выражения: $\overline{x} \vee y - \overline{y \vee x}$ при x=1D и y=20. Значение ***x*** должно находиться в ячейке памяти **03AD**h, значение ***y*** должно находиться в ячейке памяти **03BF**h, результат переслать в ячейку памяти **03CD**h.

20. Вычислить значение выражения: $y \cdot 3 + \overline{y} \vee 2$ при x=06 и y=04. Значение ***x*** должно находиться в ячейке памяти **1070**h, значение ***y*** должно находиться в ячейке памяти **1401**h, результат переслать в ячейку памяти **1015**h.

21. Вычислить значение выражения: $\overline{2 + x \wedge 2 - y}$ при x=15 и y=06. Значение ***x*** должно находиться в ячейке памяти **0257**h, значение ***y*** должно находиться в ячейке памяти **0353**h, результат переслать в ячейку памяти **0340**h.

22. Вычислить значение выражения: $(5-x) \vee \overline{\overline{x} - y}$ при x=37 и y=28. Значение ***x*** должно находиться в ячейке памяти **0114h**, значение ***y*** должно находиться в ячейке памяти **01A2**h, результат переслать в ячейку памяти **0A1F**h.

23. Вычислить значение выражения: $2 \cdot \left(\overline{\overline{x} \wedge 2}\right) \vee \overline{y}$ при x=04 и y=03. Значение ***x*** должно находиться в ячейке памяти **0014**h, значение ***y*** должно находиться в ячейке памяти **0200**h, результат переслать в ячейку памяти **011A**h

24. Вычислить значение выражения: $\overline{(y-x)} \vee (x+y)$ при x=0A и y=1B. Значение ***x*** должно находиться в ячейке памяти **2000**h, значение ***y*** должно находиться в ячейке памяти **2100**h, результат переслать в ячейку памяти **2101**h.

25. Вычислить значение выражения: $2 \cdot x - x \wedge \overline{y}$ при x=06 и y=04. Значение ***x*** должно находиться в ячейке памяти 2**070**h, значение ***y*** должно находиться в ячейке памяти **2080**h, результат переслать в ячейку памяти **1015**h.

26. Вычислить значение выражения: $\overline{(x \vee 2) \wedge (y \vee 3)}$ при x=15 и y=06. Значение ***x*** должно находиться в ячейке памяти **0257**h, значение ***y*** должно находиться в ячейке памяти **0353**h, результат переслать в ячейку памяти **0340**h.

27. Вычислить значение выражения: $\overline{(2+x)} - \overline{(0A-y)}$ при x=3A и y=2B. Значение ***x*** должно находиться в ячейке памяти **0114h**, значение ***y*** должно находиться в ячейке памяти **01A2**h, результат переслать в ячейку памяти **0A1F**h.

Лабораторная работа № 2

ОРГАНИЗАЦИЯ ВЕТВЛЕНИЙ НА ЯЗЫКЕ АССЕМБЛЕР 8085

Цель работы: Изучить принципы организации ветвлений при помощи команд условного и безусловного переходов.

Пример.

Вычислить значение выражения: $2x \vee 3y$ при x=05 и y=06. Результат записать в регистр **B**. Затем результат инвертировать и сравнить с исходным. Если исходный результат больше инвертированного, то записать его в ячейку памяти **0111(h)**. Если меньше, то записать инвертированный результат в ячейку памяти **0112(h)**.

PC	Мнемоника	Примечание
0000	LXI H,0101H	// пересылка в регистровую пару **HL** адреса ячейки памяти **0101**h
0003	MVI M,05H	// пересылка в ячейку, адрес которой лежит в регистровой паре **HL**, значения **05**h
0005	INX H	// увеличение адреса ячейки памяти в **HL** на 1 (**0102**h)
0006	MVI M,06H	// пересылка в ячейку, адрес которой лежит в регистровой паре **HL**, значения **05**h
0008	DCX H	// уменьшение адреса ячейки памяти в **HL** на 1 (**0101**h)
0009	ADD M	// Поскольку в системе команд процессора 8085 нет команды умножения, то операция 2****x*** выполняется сложением (**A+M+M**)
000A	ADD M	
000B	MOV M,A	// пересылка в ячейку, адрес которой лежит в регистровой паре **HL**, значения **A**
000C	INX H	// увеличение адреса ячейки памяти в **HL** на 1 (**0102**h)
000D	MVI A,00H	// обнуление **A**
000F	ADD M	// умножение значения в ячейке памяти **0102**h на 3. Результат остается в **A**
0010	ADD M	
0011	ADD M	
0012	DCX H	// уменьшение адреса ячейки памяти в **HL** на 1 (**0101**h)
0013	ORA M	// Логические сложение содержимого **A** и ячейки памяти **0101**h
0014	MOV B,A	// Пересылка содержимого аккумулятора в регистр **B**
0015	CMA	// Инверсия содержимого **A**
0016	CMP B	// Сравнение содержимого A и регистра B. По результатам сравнения содержимое регистров не меняется, но выставляются флаги.
0017	JM 001EH	// Переход по адресу 001E(h) если флаг S=1. (B>A).
001A	STA 0111H	// Пересылка содержимого аккумулятора в ячейку памяти **0111**h
001D	HLT	// Остановка программы
001E	MOV A,B	// Пересылка содержимого регистра **B** в аккумулятор.
001F	STA 0102H	// Пересылка содержимого аккумулятора в ячейку памяти **0112**h
0021	HLT	// Остановка программы

Задания для выполнения.

1. Вычислить значение выражения: $2x \wedge 4y$ при x=04 и y=03. Результат записать в регистр **D**. Затем результат инвертировать и сравнить с исходным. Если исходный результат больше инвертированного, то записать его в ячейку памяти **011F(h)**. Если меньше, то записать инвертированный результат в ячейку памяти **0115(h)**.

2. Вычислить значение выражения: $2\bar{x} \vee 2 \wedge y$ при x=05 и y=02. Результат записать в регистр **B**. Затем результат сложить с самим собой. Если произошло переполнение, то записать его в ячейку памяти **0151(h)**. Если нет, то записать инвертированный результат в ячейку памяти **0162(h)**.

3. Вычислить значение выражения: $2 \wedge \overline{x \vee 3} \wedge y$ при x=05 и y=04. Результат записать в регистр **C**. Затем результат инвертировать и логически сложить с исходным. Если результат равен нулю, то записать его в ячейку памяти **0111(h)**. Если нет, то записать инвертированный результат в ячейку памяти **0112(h)**.

4. Вычислить значение выражения: $(5-x) \vee x-\bar{y}$ при x=03 и y=06. Результат записать в регистр **B**. Затем результат инвертировать и сравнить с исходным. Если исходный результат больше инвертированного то записать его в ячейку памяти **0511(h)**. Если меньше, то записать инвертированный результат в ячейку памяти **0512(h)**.

5. Вычислить значение выражения: $2+\overline{x \wedge 2 \cdot \bar{y}}$ при x=03 и y=07. Результат записать в регистр **C**. Затем результат сложить с самим собой. Если произошло переполнение, то записать его в ячейку памяти **0251(h)**. Если нет, то записать инвертированный результат в ячейку памяти **0182(h)**.

6. Вычислить значение выражения: $x+\bar{x}-\bar{y} \vee 6$ при x=02 и y=04. Результат записать в регистр **B**. Затем результат инвертировать и логически сложить с исходным. Если результат равен нулю, то записать его в ячейку памяти **0113(h)**. Если нет, то записать инвертированный результат в ячейку памяти **011A(h)**.

7. Вычислить значение выражения: $\overline{2 \wedge x+2 \vee y}$ при x=01 и y=02. Результат записать в регистр **D**. Затем результат инвертировать и сравнить с исходным. Если исходный результат больше инвертированного, то записать его в ячейку памяти **041F(h)**. Если меньше, то записать инвертированный результат в ячейку памяти **0715(h)**.

8. Вычислить значение выражения: $\bar{x} \wedge y-\bar{y} \vee x$ при x=10 и y=20. Результат записать в регистр **B**. Затем результат сложить с самим собой. Если произошло переполнение, то записать его в ячейку памяти **0A51(h)**. Если нет, то записать инвертированный результат в ячейку памяти **0A62(h)**.

9. Вычислить значение выражения: $x \cdot 5+\bar{y} \vee 2$ при x=03 и y=04. Результат записать в регистр **C**. Затем результат инвертировать и логически сложить с исходным. Если результат равен нулю, то записать его в ячейку памяти **01D1(h)**. Если нет, то записать инвертированный результат в ячейку памяти **01D2(h)**.

10. Вычислить значение выражения: $\overline{2 \cdot x \wedge 2 \cdot y}$ при x=05 и y=06. Результат записать в регистр **B**. Затем результат инвертировать и сравнить с исходным. Если исходный результат больше инвертированного, то записать его в ячейку памяти **0A11(h)**. Если меньше, то записать инвертированный результат в ячейку памяти **051A(h)**.

11. Вычислить значение выражения: $(4+x) \vee \overline{x-y}$ при x=07 и y=08. Результат записать в регистр **C**. Затем результат сложить с самим собой. Если

произошло переполнение, то записать его в ячейку памяти **0B51(h)**. Если нет, то записать инвертированный результат в ячейку памяти **0B82(h)**.

12. Вычислить значение выражения: $2 \wedge \bar{x} + 2 \vee \bar{y}$ при x=04 и y=02. Результат записать в регистр **B**. Затем результат инвертировать и логически сложить с исходным. Если результат равен нулю, то записать его в ячейку памяти **0413(h)**. Если нет, то записать инвертированный результат в ячейку памяти **014A(h)**.

13. Вычислить значение выражения: $\overline{2 + x \wedge 2 \vee \bar{y}}$ при x=05 и y=06. Результат записать в регистр **D**. Затем результат инвертировать и сравнить с исходным. Если исходный результат больше инвертированного, то записать его в ячейку памяти **017F(h)**. Если меньше, то записать инвертированный результат в ячейку памяти **0175(h)**.

14. Вычислить значение выражения: $\overline{\bar{x} \cdot y - \bar{y}} \vee x$ при x=10 и y=0A. Результат записать в регистр **B**. Затем результат сложить с самим собой. Если произошло переполнение, то записать его в ячейку памяти **0251(h)**. Если нет, то записать инвертированный результат в ячейку памяти **0262(h)**.

15. Вычислить значение выражения: $\overline{2 \cdot \bar{x} \wedge 2 + y}$ при x=2F и y=06. Результат записать в регистр **C**. Затем результат инвертировать и логически сложить с исходным. Если результат равен нулю, то записать его в ячейку памяти **01CC(h)**. Если нет, то записать инвертированный результат в ячейку памяти **01C2(h)**.

16. Вычислить значение выражения: $(9 - x) \vee \overline{\bar{x} - y}$ при x=03 и y=07. Результат записать в регистр **B**. Затем результат инвертировать и сравнить с исходным. Если исходный результат больше инвертированного, то записать его в ячейку памяти **1111(h)**. Если меньше, то записать инвертированный результат в ячейку памяти **1212(h)**.

17. Вычислить значение выражения: $5 \wedge \overline{\bar{x} + 2} \vee \bar{y}$ при x=04 и y=07. Результат записать в регистр **C**. Затем результат сложить с самим собой. Если произошло переполнение, то записать его в ячейку памяти **0251(h)**. Если нет, то записать инвертированный результат в ячейку памяти **0182(h)**.

18. Вычислить значение выражения: $\overline{(5 + y)} \vee x - \bar{y}$ при x=07 и y=14. Результат записать в регистр **B**. Затем результат инвертировать и логически сложить с исходным. Если результат равен нулю, то записать его в ячейку памяти **0193(h)**. Если нет, то записать инвертированный результат в ячейку памяти **019A(h)**.

19. Вычислить значение выражения: $(2 + x) \wedge 2 \vee \bar{y}$ при x=0D и y=06. Результат записать в регистр **D**. Затем результат инвертировать и сравнить с исходным. Если исходный результат больше инвертированного то записать его в ячейку памяти **AA1F(h)**. Если меньше, то записать инвертированный результат в ячейку памяти **AA15(h)**.

20. Вычислить значение выражения: $\overline{2 \vee x + \bar{x} \vee y}$ при x=04 и y=02. Результат записать в регистр **B**. Затем результат сложить с самим собой. Если произошло результат отрицательный, то записать его в ячейку памяти **1151(h)**. Если положительный, то записать инвертированный результат в ячейку памяти **1162(h)**.

21. Вычислить значение выражения: $\overline{\bar{x} \vee y - \bar{y}} \vee x$ при x=1D и y=20. Результат записать в регистр **C**. Затем результат инвертировать и логически умножить на исходный. Если результат равен нулю, то записать его в ячейку па-

мяти **0111(h)**. Если нет, то записать инвертированный результат в ячейку памяти **0112(h)**.

22. Вычислить значение выражения: $x \cdot 7 + \overline{y} \vee 2$ при x=06 и y=04. Результат записать в регистр **B**. Затем результат инвертировать и сравнить с исходным. Если исходный результат больше инвертированного то записать его в ячейку памяти **4511(h)**. Если меньше, то записать инвертированный результат в ячейку памяти **1023(h)**.

23. Вычислить значение выражения: $\overline{2 + x \wedge 2 - y}$ при x=15 и y=06. Результат записать в регистр **C**. Затем результат сложить с самим собой. Если произошел перенос, то записать его в ячейку памяти **2251(h)**. Если нет, то записать инвертированный результат в ячейку памяти **2182(h)**.

24. Вычислить значение выражения: $(5 - x) \vee \overline{\overline{x} - y}$ при x=37 и y=28. Результат записать в регистр **B**. Затем результат инвертировать и логически сложить с исходным. Если результат равен нулю, то записать его в ячейку памяти **7113(h)**. Если нет, то записать инвертированный результат в ячейку памяти **711A(h)**.

25. Вычислить значение выражения: $\overline{(5 + y)} \vee \overline{x} + y$ при x=07 и y=14. Результат записать в регистр **B**. Затем результат инвертировать и логически сложить с исходным. Если результат равен нулю, то записать его в ячейку памяти **0193(h)**. Если нет, то записать инвертированный результат в ячейку памяти **019A(h)**.

26. Вычислить значение выражения: $(2 + x) \wedge (2 - \overline{y})$ при x=0D и y=06. Результат записать в регистр **D**. Затем результат инвертировать и сравнить с исходным. Если исходный результат больше инвертированного то записать его в ячейку памяти **AA1F(h)**. Если меньше, то записать инвертированный результат в ячейку памяти **AA15(h)**.

27. Вычислить значение выражения: $\overline{2 \vee x - \overline{x} \wedge y}$ при x=04 и y=02. Результат записать в регистр **B**. Затем результат сложить с самим собой. Если произошло переполнение, то записать его в ячейку памяти **1151(h)**. Если нет, то записать инвертированный результат в ячейку памяти **1162(h)**.

Лабораторная работа № 3

ОРГАНИЗАЦИЯ ЦИКЛОВ НА ЯЗЫКЕ АССЕМБЛЕР 8085

Цель работы: Изучить принципы организации циклов при помощи команд условного и безусловного переходов.

Теоретические сведения.

Операторы цикла используются для вычислений, повторяющихся многократно.

Блок, ради выполнения которого и организуется цикл, называется *телом цикла*. Остальные операторы служат для управления процессом повторения вычислений: это начальные установки, проверка условия продолжения цикла и модификация параметра цикла (рис. 1). Один проход цикла называется *итерацией*.

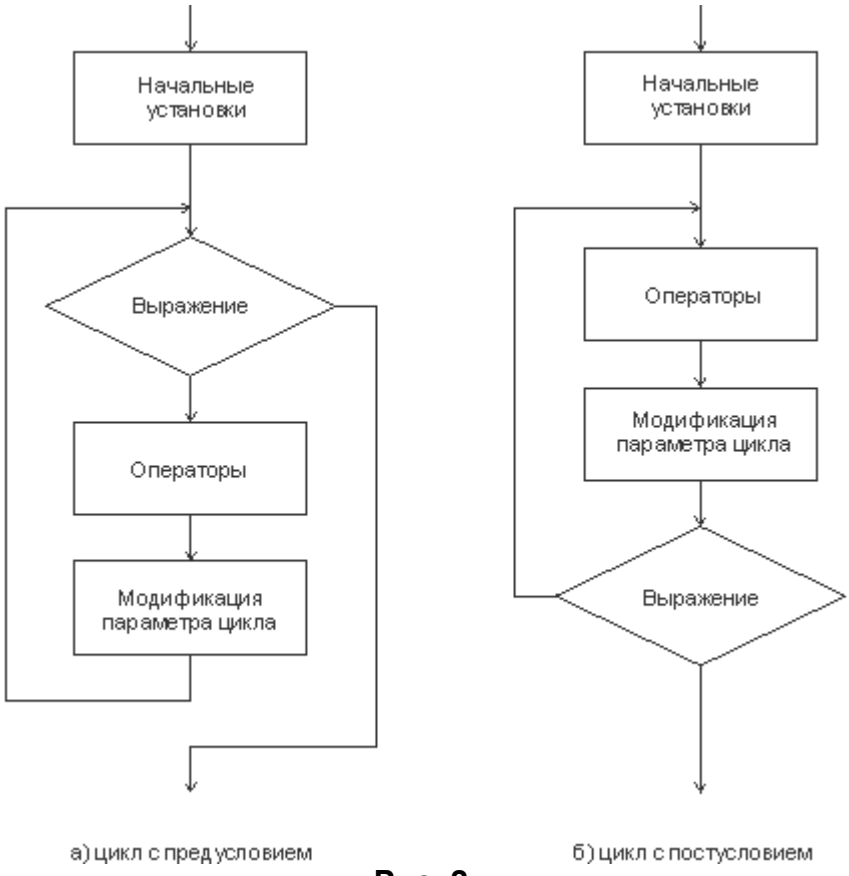


Рис. 2.

Начальные установки служат для того, чтобы до входа в цикл задать значения переменных, которые в нем используются.

Проверка условия продолжения цикла выполняется на каждой итерации либо до тела цикла (тогда говорят о цикле *с предусловием*, рис. 1, а), либо после тела цикла (цикл *с постусловием*, рис. 1, б). Разница между ними состоит в том, что тело цикла с постусловием всегда выполняется хотя бы один раз, после чего проверяется, надо ли его выполнять еще раз. Проверка необходимости выполнения цикла с предусловием делается до тела цикла, поэтому возможно, что он не выполнится ни разу.

Параметром цикла называется переменная, которая используется при проверке условия цикла и принудительно изменяется на каждой итерации, причем, как правило, на одну и ту же величину. Если параметр цикла целочислен-

ный, он называется счетчиком цикла. Количество повторений такого цикла можно определить заранее. Параметр есть не у всякого цикла.

Цикл завершается, если условие его продолжения не выполняется. Возможно принудительное завершение, как текущей итерации, так и цикла в целом.

Для организации циклических конструкций в языке ассемблер, необходимо использовать, следующие конструкции:

- ✓ Команды, предназначенные для многократного автоматического повторения.
- ✓ Счетчик цикла. Представляет собой регистр, в который помещают количество итераций цикла. Постепенное уменьшение счетчика позволяет контролировать количество выполненных действий.
- ✓ Команда условного перехода. Проверяет состояние счетчика и по результатам проверки, либо продолжает цикл, либо прерывает его и выполняет следующую команду.

Пример.

Вычислить значение выражения: $x \cdot y + x \wedge y$ при *x*=05 и *y*=06. Результат записать в ячейку памяти **1A2C(h)**.

Поскольку в системе команд процессора 8085 отсутствует команда умножения, операцию *x·y* можно осуществить только поэтапным сложением операндов друг с другом. Так как изначально значения операндов нам неизвестно, то определить количество сложений мы не можем. Для решения этой проблемы можно воспользоваться циклической конструкцией.

PC	Мнемоника	Примечание
0000	MVI C,06H	// пересылка в регистр **C** операнда ***y*=06h**
0002	MVI B,05H	// пересылка в регистр **B** операнда ***x*=05h (**Данное значения является счетчиком операций сложения. Именно столько раз мы должны сложить между собой число 06, что бы получилось 05·06)
0004	MOV A,C	// пересылка в **A** операнда ***y*=06h**
0005	ANA B	// логическое умножение ***x*∧*y***
0006	MOV D,A	// пересылка результата логического умножения в регистр **D.**
0007	MOV A,C	// заново пересылка в **A** операнда ***y*=06h**
0008	L1: ADD A	// сложение содержимого **A** самого с собой.
0009	DCR B	// уменьшения счетчика количества сложений на 1.
000A	JNZ L1	// проверка содержимого счетчика. Если **B=0**, то переход на следующую команду. Если **B>0,** то переход к метке L1 и повтор операции сложения аккумулятора
000D	ADD D	// Сложение результатов операций ***x*∧*y*** и ***x*·*y***
000E	LXI H,1A1CH	// Пересылка в регистровую пару HL адреса ячейки памяти, в которую будет пересылаться результат вычислений.
0011	MOV M,A	// Пересылка содержимого аккумулятора, по адресу находящемуся в регистровой паре **HL**.
0012	HLT	// Остановка программы

Задания для выполнения.

1. Вычислить значение выражения: $x \cdot y - x \vee y$ при *x*=05 и *y*=06. Результат записать в ячейку памяти **001A(h)**.
2. Записать последовательно в ячейки памяти с 1020,1021,…,1040 числа от 00,01,02…
3. Вычислить значение выражения: $\overline{(2 \wedge x) - (x \cdot y)}$ при x=01 и y=02, результат переслать в ячейку памяти **0110**h.
4. Записать последовательно в ячейки памяти с 1040,1041,…,1060 числа от 00,02,04…
5. Вычислить значение выражения: $\overline{x} \cdot \left(y - \overline{y} \vee x\right)$ при x=10 и y=20, результат переслать в ячейку памяти **00C0**h.
6. Записать последовательно в ячейки памяти с 1030,1031,… числа от 01,03,…,21.
7. Вычислить значение выражения: $x \cdot \left(5 + \overline{y} \vee 2\right)$ при x=03 и y=04, результат переслать в ячейку памяти **1010**h.
8. Записать последовательно в ячейки памяти с 1010,1011,… числа от 01,02,…,1F.
9. Вычислить значение выражения: $(x \wedge 2) \cdot y + x$ при x=05 и y=06, результат переслать в ячейку памяти **0300**h.
10. Записать последовательно в ячейки памяти с 100A,100B,…,1030 числа от 00,03,06….
11. Вычислить значение выражения: $(4 + x) \cdot \left(\overline{x - y}\right)$ при x=07 и y=08, значение ***y*** должно находиться в ячейке памяти **0122**h, результат переслать в ячейку памяти **0A10**h.
12. Записать последовательно в ячейки памяти с 100A,100B,… числа от 00,02,…,20
13. Вычислить значение выражения: $\left(2 \wedge \overline{x} + 2\right) \cdot \overline{y}$ при x=04 и y=02, результат переслать в ячейку памяти **011A**h
14. Записать последовательно в ячейки памяти с 100A,100B,… числа от 01,03,05…,30
15. Вычислить значение выражения: $(5 - x) \cdot \left(x - \overline{y}\right)$ при x=02 и y=04, результат переслать в ячейку памяти **0A1A**h.
16. Записать последовательно в ячейки памяти с 1010,1011,…,1020 числа от 01,02,04,08,…
17. Вычислить значение выражения: $\left(2 + x \wedge 2\right) \cdot \overline{y}$ при x=09 и y=06, результат переслать в ячейку памяти **003D**h.
18. Записать последовательно в ячейки памяти с 1020,1021,… числа от 01,02,04,08,…,40
19. Вычислить значение выражения: $x \cdot \left(5 - \overline{y} \vee 2\right)$ при x=08 и y=04, результат переслать в ячейку памяти **1070**h.
20. Записать последовательно в ячейки памяти с 1020,1021,… числа от 00,01,02,…,40
21. Вычислить значение выражения: $x + \left(x^2 - \overline{y} \vee 6\right)$ при x=08 и y=03, результат переслать в ячейку памяти **101C**h.
22. Записать последовательно в ячейки памяти с 1010,1011,… числа от 01,03,06,09,…,1B

23. Вычислить значение выражения: $\overline{x \cdot \left(y + \bar{y} \vee x\right)}$ при x=10 и y=20, результат переслать в ячейку памяти **00CF**h.
24. Записать последовательно в ячейки памяти с 1010,1011,…,0022 числа от 01,03,06,09,…
25. Вычислить значение выражения: $(\bar{x} \cdot y) - \left(\bar{y} \vee x\right)$ при x=10 и y=0A, результат переслать в ячейку памяти **00DF**h.
26. Записать последовательно в ячейки памяти с 101A,101B,…,1032 числа от 01,02,…
27. Вычислить значение выражения: $\left(2 + \bar{x} \wedge 2\right) \cdot y$ при x=2F и y=06, результат переслать в ячейку памяти **003A**h.

Лабораторная работа № 4

ОРГАНИЗАЦИЯ ПОДПРОГРАММ НА ЯЗЫКЕ АССЕМБЛЕР 8085

Цель работы: Изучить принципы вызова подпрограмм и возврата из подпрограмм при помощи специализированных команд.

Теоретические сведения.

Подпрограммы представляют собой важный инструмент любого языка программирования, позволяющий писать хорошо структурированные программы. В структурированных программах обычно легко прослеживается основной алгоритм, их нетрудно понять любому читателю, они проще в отладке и менее чувствительны к ошибкам программирования. Все эти свойства являются следствием важной особенности подпрограмм, каждая из которых представляет собой во многом самостоятельный фрагмент программы, связанный с основной программой лишь с помощью нескольких параметров. Самостоятельность подпрограмм позволяет локализовать в них все детали программной реализации того или иного алгоритмического действия и поэтому изменение этих деталей, например, в процессе отладки обычно не приводит к изменениям основной программы.

Сразу после активизации процедуры начинают выполняться входящие в нее операторы, после выполнения последнего из них управление возвращается обратно в основную программу, и выполняются команды, стоящие непосредственно за командой вызова процедуры (рис.2).

Рис.3 Взаимодействие вызывающей программы и процедуры

Пример.

Вычислить значение выражения: $y = ax + b$ при a=02h, b=03, *x* – сумма чисел от 1 до 7. Результат записать в ячейку памяти **00AA(h)**. Вычисление суммы от 1 до 7 оформить в виде подпрограммы.

Структурно подпрограмма может находиться в любой области памяти. Что бы отделить выполнение подпрограммы от выполнения основной программы, необходимо разместить подпрограмму адресно выше основной, а в качестве первоначального адреса установить адрес основной программы.

PC	Мнемоника	Примечание
	Подпрограмма	
0000	MVI D,01H	// пересылка в регистр **D** числа **01** – начальный элемент суммирования.
0002	MVI C,07H	// пересылка в регистр **C** числа **07h (**Данное значение является счетчиком операций сложения.)
0004	ADD D	// сложение содержимого аккумулятор с регистром **D**
0005	MOV E,A	// Сохранение результата суммирования в регистре **E**.
0006	INR D	// Увеличение содержимого регистра **D** на 1**.**
0007	DCR C	// уменьшения счетчика количества сложений на 1.

0008	RZ	// Если счетчик равен 0, то выход из подпрограммы.
0009	JMP 0004H	// Переход вновь на операцию сложения.
Основная программа		
000C	LXI H,0051H	// пересылка операнда b=03H по адресу 0051H
000F	MVI M,03H	
0011	LXI H,0050H	// пересылка операнда a=02H по адресу 0050H
0014	MVI M,02H	
0016	CALL 0000H	// Вызов подпрограммы
0019	MVI A,00H	// Обнуление аккумулятора
001B	ADD M	// Умножение операнда ***a*** на результат суммирования ***x***. Результат суммирования находится в регистре E и представляет собой счетчик сложений числа 02 с собой.
001C	DCR E	
001D	JNZ 001BH	
0020	LXI H,0051H	// Сложение операнда **b** с результатом умножения ***a*×*x***
0023	ADD M	
0024	STA 00AAH	// Пересылка получившегося значения по адресу **00AA**
0027	HLT	// Остановка программы

Задания для выполнения.

1. Вычислить значение выражения: $y = x \cdot (a + b) - c$ при a=03h, b=04h, c=02h, *x* – сумма чисел от 1 до 8. Вычисление суммы от 1 до 8 оформить в виде подпрограммы.

2. Вычислить значение выражения: $y = a \cdot b - x$ при a=05h, b=08h, *x* – сумма чисел от 1 до 10. Вычисление суммы от 1 до 10 оформить в виде подпрограммы.

3. Вычислить значение выражения: $y = x \cdot (a - b)$ при a=03h, b=01h, *x* – сумма чисел от 1 до 6. Вычисление суммы от 1 до 6 оформить в виде подпрограммы.

4. Вычислить значение выражения: $y = a \cdot b + x$ при a=05h, b=02h, *x* – сумма чисел от 1 до 9. Вычисление суммы от 1 до 9 оформить в виде подпрограммы.

5. Вычислить значение выражения: $y = x \cdot (a + b \cdot c)$ при a=03h, b=04h, c=02h, *x* – сумма чисел от 1 до 7. Вычисление суммы от 1 до 7 оформить в виде подпрограммы.

6. Вычислить значение выражения: $y = a \cdot x - b$ при a=05h, b=f8h, *x* – сумма чисел от 1 до 10. Вычисление суммы от 1 до 10 оформить в виде подпрограммы.

7. Вычислить значение выражения: $y = x \cdot (a + b)$ при a=07h, b=02h, *x* – сумма чисел от 1 до 8. Вычисление суммы от 1 до 8 оформить в виде подпрограммы.

8. Вычислить значение выражения: $y = a \cdot (b + x)$ при a=05h, b=02, *x* – сумма чисел от 1 до 9. Вычисление суммы от 1 до 9 оформить в виде подпрограммы.

9. Вычислить значение выражения: $y = x \cdot a + b$ при a=03h, b=04h, *x* – сумма чисел от 1 до 8. Вычисление суммы от 1 до 8 оформить в виде подпрограммы.

10. Вычислить значение выражения: $y = a \cdot (b - x)$ при a=05h, b=f2h, *x* – сумма чисел от 1 до 7. Вычисление суммы от 1 до 7 оформить в виде подпрограммы.

11. Вычислить значение выражения: $y = x \cdot (a - b) + c$ при a=03h, b=01h, c=04h, *x* – сумма чисел от 1 до 6. Вычисление суммы от 1 до 6 оформить в виде подпрограммы.

12. Вычислить значение выражения: $y = (a - c) \cdot (b + x)$ при a=05h, b=02h, c=02h, *x* – сумма чисел от 1 до 9. Вычисление суммы от 1 до 9 оформить в виде подпрограммы.

13. Вычислить значение выражения: $y = x \cdot (a - b \cdot c)$ при a=03h, b=02h, c=02h, *x* – сумма чисел от 1 до 7. Вычисление суммы от 1 до 7 оформить в виде подпрограммы.

14. Вычислить значение выражения: $y = a \cdot (x - b)$ при a=05h, b=f0h, *x* – сумма чисел от 1 до 10. Вычисление суммы от 1 до 10 оформить в виде подпрограммы.

15. Вычислить значение выражения: $y = (x - c) \cdot (a + b)$ при a=07h, b=02h, c=d4h, *x* – сумма чисел от 1 до 8. Вычисление суммы от 1 до 8 оформить в виде подпрограммы.

16. Вычислить значение выражения: $y = (a - c) \cdot (b + x)$ при a=05h, b=02, c=03h, *x* – сумма чисел от 1 до 9. Вычисление суммы от 1 до 9 оформить в виде подпрограммы.

17. Вычислить значение выражения: $y = x \cdot (a \cdot b - c)$ при a=03h, b=02h, c=09h, *x* – сумма чисел от 1 до 7. Вычисление суммы от 1 до 7 оформить в виде подпрограммы.

18. Вычислить значение выражения: $y = a \cdot (x - b \cdot c)$ при a=05h, b=35h, c=02h, *x* – сумма чисел от 1 до 10. Вычисление суммы от 1 до 10 оформить в виде подпрограммы.

19. Вычислить значение выражения: $y = (x - b) \cdot (a - b)$ при a=07h, b=02h, *x* – сумма чисел от 1 до 8. Вычисление суммы от 1 до 8 оформить в виде подпрограммы.

20. Вычислить значение выражения: $y = (a - c) \cdot (b - x)$ при a=05h, b=02, c=03h, *x* – сумма чисел от 1 до 9. Вычисление суммы от 1 до 9 оформить в виде подпрограммы.

21. Вычислить значение выражения: $y = (x - b) \cdot (a + b)$ при a=07h, b=02h, *x* – сумма чисел от 1 до 8. Вычисление суммы от 1 до 8 оформить в виде подпрограммы.

22. Вычислить значение выражения: $y = (a + c)/(b - x)$ при a=05h, b=02, c=03h, *x* – сумма чисел от 1 до 9. Вычисление суммы от 1 до 9 оформить в виде подпрограммы.

23. Вычислить значение выражения: $y = a/(b - x)$ при a=50h, b=f2h, *x* – сумма чисел от 1 до 7. Вычисление суммы от 1 до 7 оформить в виде подпрограммы.

24. Вычислить значение выражения: $y = x \cdot (a \cdot b) - c$ при a=03h, b=01h, c=04h, *x* – сумма чисел от 1 до 6. Вычисление суммы от 1 до 6 оформить в виде подпрограммы.

25. Вычислить значение выражения: $y = (a - c) \cdot (b + x)$ при a=05h, b=02h, c=02h, *x* – сумма чисел от 1 до 9. Вычисление суммы от 1 до 9 оформить в виде подпрограммы.

26. Вычислить значение выражения: $y = x \cdot (a - b/c)$ при a=03h, b=02h, c=02h, *x* – сумма чисел от 1 до 7. Вычисление суммы от 1 до 7 оформить в виде подпрограммы.

Лабораторная работа № 5

РАБОТА С ПОРТАМИ ВВОДА/ВЫВОДА.

Цель работы: Изучить принципы отсылки данных на внешний порт и приема данных и внешнего порта.

Теоретические сведения.

1. Порты ввода/вывода.

Для работы с портами ввода вывода в эмуляторе 8085 Simulator IDE имеется два окна:

- ✓ **I/O Ports Editor** – выводит на экран диапазон адресуемых портов. (00H – FFH)
- ✓ **Peripheral Devices** – Выводит на экран 4 8-разрядных порта ввода/вывода и один порт для вывода символов кода ASCII.

В системе команд процессора 8085 специально выделяются функции обмена с устройствами ввода/вывода. Команда **IN** используется для ввода (чтения) информации из устройства ввода/вывода, а команда **OUT** используется для вывода (записи) в устройство ввода/вывода. Обмен информацией в этом случае производится между регистром-аккумулятором и устройством ввода/вывода.

Пример.

1. Организовать в порту с адресом 02H, режим «бегающей единицы».

Для работы необходимо:

1. Открыть окно периферийных устройств (**Tools→ Peripheral Devices**).
2. По умолчанию, все порты находятся в выключенном состоянии (**OFF**). Для включения порта необходимо нажать кнопку **OFF** и в появившемся окне вписать номер порта (в нашем случае 02).
3. Так как порт планируется использовать для вывода данных, необходимо поставить переключатель на пункт **OUT.**

Программа для выполнения задания.

PC	Мнемоника	Примечание
0000	MVI A,01H	// пересылка в аккумулятор числа 00000001
0002	OUT 02H	// пересылка содержимого аккумулятора в порт вывода 02H
0004	RAL	// циклических сдвиг содержимого аккумулятора влево.
0005	JMP 0002H	// переход по адресу 0002 для повторения итераций.

2. Переслать в порт 03 слово «Hello»

Для работы необходимо:

1. Открыть окно периферийных устройств (**Tools→ Peripheral Devices**).
2. Включить порт для вывода символов кода ASCII (**Output Terminal**). В появившемся окне написать номер порта (в нашем случае 03).

Программа для выполнения задания.

PC	Мнемоника	Примечание
0000	MVI A,22H	// Пересылка в **A** шестнадцатеричного числа 22, соответствующего символу **“**
0002	OUT 03H	// Пересылка символа в порт вывода 03

0004	MVI A,48H	// Пересылка в **A** шестнадцатеричного числа 48, соответствующего символу **H**
0006	OUT 03H	// Пересылка символа в порт вывода 03
0008	MVI A,65H	// Пересылка в **A** шестнадцатеричного числа 65, соответствующего символу **e**
000A	OUT 03H	// Пересылка символа в порт вывода 03
000C	MVI A,6CH	// Пересылка в **A** шестнадцатеричного числа 6C, соответствующего символу **l**
000E	OUT 03H	// Пересылка символа в порт вывода 03
0010	OUT 03H	// Пересылка символа в порт вывода 03
0012	MVI A,6FH	// Пересылка в **A** шестнадцатеричного числа 6F, соответствующего символу **o**
0014	OUT 03H	// Пересылка символа в порт вывода 03
0016	MVI A,22H	// Пересылка в **A** шестнадцатеричного числа 22, соответствующего символу **“**
0018	OUT 03H	// Пересылка символа в порт вывода 03
001A	HLT	// Остановка программы.

Таблица кодов ASCII.

Шестнадцатеричное обозначения символа	Символ кода ASCII	Шестнадцатеричное обозначения символа	Символ кода ASCII
00	NUL	41	A
01	SOH	42	B
02	STX	43	C
03	ETX	44	D
04	EOT	45	E
05	ENQ	46	F
06	ACK	47	G
07	BEL	48	H
08	BS	49	I
09	HT	4A	J
0A	LF	4B	K
0B	VT	4C	L
0C	FF	4D	M
0D	CR	4E	N
0E	SO	4F	O
0F	SI	50	P
10	DLE	51	Q
11	C1	52	R
12	DC2	53	S
13	DC3	54	T
14	DC4	55	U
15	NAK	56	V
16	SYN	57	W
17	TB	58	X
18	CAN	59	Y
19	EM	5A	Z
1A	SUB	5B	[

1B	ESC	5C	/
1C	FS	5D	]
1D	GS	5E	^
1E	RS	5F	-
1F	US	60	.
20	SP	61	a
21	!	62	b
22	“	63	c
23	#	64	d
24	$	65	e
25	½	66	f
26	&	67	g
27	‘	68	h
28	(	69	i
29	)	6A	j
2A	*	6B	k
2B	+	6C	l
2C	’	6D	m
2D	-	6E	n
2E	.	6F	o
2F	/	70	p
30	0	71	q
31	1	72	r
32	2	73	s
33	3	74	t
34	4	75	u
35	5	76	v
36	6	77	w
37	7	78	x
38	8	79	y
39	9	7A	z
3A	:	7B	R
3B	;	7C	/
3C	<	7D	T
3D	=	7E	~
3E	>	7F	DEL
3F	?		
40	@		

Задания для выполнения.

1. Осуществить в порту 01 режим однократного пробега единицы справа налево (Значение порта должно начинаться с **00000001** и заканчиваться **10000000**).
2. Вывести в порт 02 свои фамилию и инициалы, записанные транслитом.
3. Организовать в порту 02 режим «бегающего нуля» справа налево.
4. Осуществить в порту 02 режим однократного пробега единицы слева направо (Значение порта должно начинаться с **10000000** и заканчиваться **000000001**).
5. Вывести в порт 05 свои фамилию и инициалы, записанные транслитом.
6. Организовать в порту 02 режим «бегающего нуля» слева направо

7. Осуществить в порту 01 режим однократного пробега единицы справа налево через разряд (Значение порта должно начинаться с **00000001** и заканчиваться **10000000**).
8. Вывести в порт 02 свои фамилию и инициалы, записанные транслитом.
9. Организовать в порту 04 режим «бегающего нуля» справа налево через разряд.
10. Осуществить в порту 01 режим однократного пробега единицы слева направо через разряд (Значение порта должно начинаться с **10000000** и заканчиваться **000000001**).
11. Вывести в порт 02 свои фамилию и инициалы, записанные транслитом.
12. Организовать в порту 02 режим «бегающего нуля» слева направо через разряд
13. Осуществить в порту 01 режим однократного пробега единицы справа налево через два разряда (Значение порта должно начинаться с **00000001** и заканчиваться **10000000**).
14. Вывести в порт 03 свои фамилию и инициалы, записанные транслитом.
15. Организовать в порту 02 режим «бегающего нуля» справа налево через два разряда.
16. Осуществить в порту 01 режим однократного пробега единицы слева направо через два разряда (Значение порта должно начинаться с **10000000** и заканчиваться **000000001**).
17. Вывести в порт 02 свои фамилию и инициалы, записанные транслитом.
18. Организовать в порту 02 режим «бегающего нуля» слева направо через два разряда
19. Осуществить в порту 01 режим однократного пробега единицы слева направо и справа налево (Значение порта должно начинаться с **10000000** и заканчиваться **000000001**).
20. Вывести в порт 02 свои фамилию и инициалы, записанные транслитом.
21. Организовать в порту 02 режим «бегающего нуля» слева направо и справа налево.

Лабораторная работа № 6

ОРГАНИЗАЦИЯ ОБРАБОТКИ ПРЕРЫВАНИЙ НА ЯЗЫКЕ АССЕМБЛЕР 8085.

Цель работы: Изучить принципы обработки прерываний и организацию обработки прерываний при помощи специализированных команд языка ассемблер для процессора 8085.

Теоретические сведения.

Обмен по прерываниям используется тогда, когда необходима реакция микропроцессорной системы на какое-то внешнее событие, на приход внешнего сигнала. В случае компьютера внешним событием может быть, например, нажатие на клавишу клавиатуры или приход по локальной сети пакета данных. Компьютер должен реагировать на это, соответственно, выводом символа на экран или же чтением и обработкой принятого по сети пакета.

В общем случае организовать реакцию на внешнее событие можно тремя различными путями:

- ✓ с помощью постоянного программного контроля факта наступления события (так называемый метод опроса флага или polling);
- ✓ с помощью прерывания, то есть насильственного перевода процессора с выполнения текущей программы на выполнение экстренно необходимой программы;
- ✓ с помощью прямого доступа к памяти, то есть без участия процессора при его отключении от системной магистрали.

Команды прерываний относятся к командам перехода с возвратом. Эти команды в качестве входного операнда требуют номер прерывания (адрес вектора). Обслуживание таких переходов осуществляется точно так же, как и аппаратных прерываний. То есть для выполнения данного перехода процессор обращается к таблице векторов прерываний и получает из нее по номеру прерывания адрес памяти, в который ему необходимо перейти. Адрес вызова прерывания и содержимое регистра состояния процессора (PSW) сохраняются в стеке. Сохранение PSW – важное отличие команд прерывания от команд переходов с возвратом.

Команды прерываний во многих случаях оказываются удобнее, чем обычные команды переходов с возвратом. Сформировать таблицу векторов прерываний можно один раз, а потом уже обращаться к ней по мере необходимости. Номер прерывания соответствует номеру подпрограммы, то есть номеру функции, выполняемой подпрограммой. Поэтому команды прерывания гораздо чаще включаются в системы команд процессоров, чем обычные команды переходов с возвратом.

Для возврата из подпрограммы, вызванной командой прерывания, используется команда возврата из прерывания. Эта команда извлекает из стека сохраненное там значение счетчика команд и регистра состояния процессора (PSW).

Отметим, что у некоторых процессоров предусмотрены также команды условных прерываний, например, команда прерывания при переполнении.

Вызов обработки прерывания можно организовать тремя способами:

1. Командами вызова.
   - ♦ RST 0 – Запуск программы с адреса 0h
   - ♦ RST 1 – Запуск программы с адреса 8h
   - ♦ RST 2 – Запуск программы с адреса 10h

- RST 3 – Запуск программы с адреса 18h
- RST 4 – Запуск программы с адреса 20h
- RST 5 – Запуск программы с адреса 28h
- RST 6 – Запуск программы с адреса 30h
- RST 7 – Запуск программы с адреса 38h

2. Нажатием кнопки INTR. После нажатия кнопки появляется окно с запросом номера прерывания.
   - #1 – RST 0
   - #2 – RST 1
   - #3 – RST 2
   - #4 – RST 3
   - #5 – RST 4
   - #6 – RST 5
   - #7 – RST 6
   - #8 – RST 7
3. Нажатием кнопок RST5.5, RST6.5, RST7.5.
   - Кнопка RST5.5 – Запуск программы с адреса 2Ch
   - Кнопка RST6.5 – Запуск программы с адреса 34h
   - Кнопка RST7.5 – Запуск программы с адреса 3Ch

Пример.

Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 01H и вывод этого символа в порт 02H. Окончанием ввода служит ввод символа **Enter.** Программа обработки располагается по адресу 0034H. Запрос на обработку должен осуществляться нажатием кнопки **RST6.5.**

PC	Мнемоника	Примечание
0000	JMP 0100H	// Переход к основной программе
Программа обработки прерываний		
0003	.ORG 0034H	// Установка регистра **PC** равное 0034H
0034	L1: IN 01H	// Прием кода ASCII из порта 01H.
0036	CPI 0DH	// Сравнение принятого кода с кодом символа **Enter**
0038	OUT 02H	// Пересылка приятого кода ASCII в порт 02
003A	JNZ L1	// Если принятый код равен коду символа **Enter**
003D	EI	// Разрешение прерываний
003E	RET	// Возврат из команды обработки прерываний
Основная программа		
003F	.ORG 0100H	// Установка регистра **PC** равное 0034H
0100	EI	// Разрешение прерываний
0101	L2: JMP L2	// Бесконечный цикл. Ожидание запроса на прерывание.
0104	HLT	// Остановка программы.

Задания для выполнения.

1. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 02H и вывод этого символа в порт 03H. Окончанием ввода служит ввод символа **Space.** Программа

обработки располагается по адресу 0008H. Запрос на обработку должен осуществляться специальной командой в коде программы.

2. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 05H и вывод этого символа в порт 06H. Окончанием ввода служит ввод символа **Tab.** Программа обработки располагается по адресу 0010H. Запрос на обработку должен осуществляться нажатием кнопки **INTR** и вводом номера прерывания.

3. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 01H и вывод этого символа в порт 03H. Окончанием ввода служит ввод символа **A.** Программа обработки располагается по адресу 003CH. Запрос на обработку должен осуществляться нажатием кнопки **RST7.5** и вводом номера прерывания.

4. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 01H и вывод этого символа в порт 03H. Окончанием ввода служит ввод символа **b.** Программа обработки располагается по адресу 0008H. Запрос на обработку должен осуществляться специальной командой в коде программы.

5. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 02H и вывод этого символа в порт 06H. Окончанием ввода служит ввод символа **C.** Программа обработки располагается по адресу 0010H. Запрос на обработку должен осуществляться нажатием кнопки **INTR** и вводом номера прерывания.

6. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 05H и вывод этого символа в порт 02H. Окончанием ввода служит ввод символа **d.** Программа обработки располагается по адресу 002CH. Запрос на обработку должен осуществляться нажатием кнопки **RST5.5** и вводом номера прерывания.

7. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 05H и вывод этого символа в порт 03H. Окончанием ввода служит ввод символа **e.** Программа обработки располагается по адресу 0008H. Запрос на обработку должен осуществляться специальной командой в коде программы.

8. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 01H и вывод этого символа в порт 06H. Окончанием ввода служит ввод символа **f.** Программа обработки располагается по адресу 0010H. Запрос на обработку должен осуществляться нажатием кнопки **INTR** и вводом номера прерывания.

9. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 03H и вывод этого символа в порт 06H. Окончанием ввода служит ввод символа **g.** Программа обработки располагается по адресу 0034H. Запрос на обработку должен осуществляться нажатием кнопки **RST6.5** и вводом номера прерывания.

10. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 02H и вывод этого символа в порт 05H. Окончанием ввода служит ввод символа **h.** Программа обработки располагается по адресу 0008H. Запрос на обработку должен осуществляться специальной командой в коде программы.

11. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 05H и вывод этого символа в порт 01H. Окончанием ввода служит ввод символа **i.** Программа обработки располагается по адресу 0010H. Запрос на обработку должен осуществляться нажатием кнопки **INTR** и вводом номера прерывания.

12. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 05H и вывод этого символа в порт 04H. Окончанием ввода служит ввод символа **j.** Программа обработки располагается по адресу 003CH. Запрос на обработку должен осуществляться нажатием кнопки **RST7.5** и вводом номера прерывания.

13. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 02H и вывод этого символа в порт 04H. Окончанием ввода служит ввод символа **k.** Программа обработки располагается по адресу 0008H. Запрос на обработку должен осуществляться специальной командой в коде программы.

14. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 02H и вывод этого символа в порт 01H. Окончанием ввода служит ввод символа **l.** Программа обработки располагается по адресу 0010H. Запрос на обработку должен осуществляться нажатием кнопки **INTR** и вводом номера прерывания.

15. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 03H и вывод этого символа в порт 02H. Окончанием ввода служит ввод символа **m.** Программа обработки располагается по адресу 0034H. Запрос на обработку должен осуществляться нажатием кнопки **RST6.5** и вводом номера прерывания.

16. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 04H и вывод этого символа в порт 01H. Окончанием ввода служит ввод символа **n.** Программа обработки располагается по адресу 0008H. Запрос на обработку должен осуществляться специальной командой в коде программы.

17. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 01H и вывод этого символа в порт 02H. Окончанием ввода служит ввод символа **Ctrl.** Программа обработки располагается по адресу 0010H. Запрос на обработку должен осуществляться нажатием кнопки **INTR** и вводом номера прерывания.

18. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 04H и вывод этого символа в порт 02H. Окончанием ввода служит ввод символа **Alt.** Программа обработки располагается по адресу 002CH. Запрос на обработку должен осуществляться нажатием кнопки **RST5.5** и вводом номера прерывания.

19. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 02H и вывод этого символа в порт 01H. Окончанием ввода служит ввод символа **Shift.** Программа обработки располагается по адресу 0008H. Запрос на обработку должен осуществляться специальной командой в коде программы.

20. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 02H и вывод этого символа в порт 03H. Окончанием ввода служит ввод символа **F1.** Программа обработки располагается по адресу 0010H. Запрос на обработку должен осуществляться нажатием кнопки **INTR** и вводом номера прерывания.

21. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 02H и вывод этого символа в порт 03H. Окончанием ввода служит ввод символа **F2.** Программа обработки располагается по адресу 003CH. Запрос на обработку должен осуществляться нажатием кнопки **RST7.5** и вводом номера прерывания.

Приложение.

СПРАВОЧНИК ПО СИСТЕМЕ КОМАНД

МИКРОПРОЦЕССОРА INTEL 8080

Команды этого процессора бывают одно-, двух- и трехбайтными. В первом байте всегда содержится код операции.

Обозначения.

A, B, ..., L - названия 8-разрядных регистров.

BC, DE, HL - названия регистровых пар, образующих 16-разрядные регистры.

SP - 16-разрядный указатель стека.

PSW - слово состояния программы, содержит регистр флагов.

a16 - двухбайтовый адрес.

d8 - байт непосредственных данных.

d16 - два байта непосредственных данных.

pp - номер порта ввода-вывода.

Команда	Код	Описание
ADD A	87	A←(A)+(A)
ADD B	80	A← (B)+(A)
ADD C	81	A← (C)+(A)
ADD D	82	A← (D)+(A)
ADD E	83	A← (E)+(A)
ADD H	84	A← (H)+(A)
ADD L	85	A← (L)+(A)
ADD M	86	A←@ (HL)+(A)
ADI d8	C6	A←d8+(A)
ADC A	8F	A← (A)+(A)+CY
ADC B	88	A← (B)+(A)+CY
ADC C	89	A← (C)+(A)+CY
ADC D	8A	A← (D)+(A)+CY
ADC E	8B	A← (E)+(A)+CY
ADC H	8C	A← (H)+(A)+CY
ADC L	8D	A← (L)+(A)+CY
ADC M	8E	A←@ (HL)+(A)+CY
ACI d8	CE	A←d8+(A)+CF
ANA A	A7	Проверка A

Команда	Код	Описание
ANA B	A0	Логическое “И” B с A
ANA C	A1	Логическое “И” C с A
ANA D	A2	Логическое “И” D с A
ANA E	A3	Логическое “И” E с A
ANA H	A4	Логическое “И” H с A
ANA L	A5	Логическое “И” L с A
ANA M	A6	Логическое “И” @(HL) с A
ANI d8	E6	Логическое “И” непосредственные данные с A
CALL a16	CD	Передать управление подпрограмме по адресу aa
CZ a16	CC	Вызвать подпрограмму по адресу aa, если нуль (флаг нулевого результата Z=1).
CNZ a16	C4	То же, если не нуль (флаг нулевого результата Z=0)
CP a16	F4	То же, если плюс (флаг знака S=0)
CM a16	FC	То же, если минус (флаг знака S=1)
CC a16	DD	То же, если перенос (флаг переноса при арифметических операциях CF=1)
CNC a16	D4	То же, если нет переноса (флаг переноса при арифметических операциях CF=0)
CPE a16	EC	То же, если четно (флаг четности результата P=1)
CPO a16	E4	То же, если нечетно (флаг четности результата P=0)
CMA	2F	Инвертировать A
CMC	3F	Инвертировать перенос
CMP A	BF	Установить флаг FZ
CMP B	B8	Сравнить A с B
CMP C	B9	Сравнить A с C
CMP D	BA	Сравнить A с D
CMP E	BB	Сравнить A с E
CMP H	BC	Сравнить A с H

Команда	Код	Описание
CMP L	BD	Сравнить A с L
CMP M	BE	Сравнить A с @(HL)
CPI d8	FC	Сравнить A с непосредственными данными, заданными в команде
DAA	27	Десятичная коррекция аккумулятора
DAD	B	Сложить BC с HL
DAD	D	Сложить DE с HL
DAD	H	Сложить HL с HL (удвоение HL)
DAD	SP	Сложить SP с HL
DCR A	3D	A← (A) - 1 (декремент A)
DCR B	05	B← (B) - 1
DCR C	0D	C← (C) - 1
DCR D	15	D← (D) - 1
DCR E	1D	E← (E) - 1
DCR H	25	H← (H) - 1
DCR L	2D	L← (L) - 1
DCR M	3D	@(HL) ← (@(HL)) -1
DCX B	0B	BC← (BC) - 1
DCX D	1B	DE← (DE) -1
DCX H	2B	HL← (HL) - 1
DCX SP	0B	SP← (SP) -1
DI	F3	Запретить прерывания
EI	FB	Разрешить прерывания
HLT	76	Останов процессора
IN pp	DB	Ввести данные из порта pp
INR A	3C	A← (A) + 1 (инкремент A)
INR B	04	B← (B) + 1
INR C	0C	C← (C) + 1
INR D	3C	D← (D) + 1
INR E	3C	E← (E) + 1

Команда	Код	Описание
INR H	3C	H← (H) + 1
INR L	3C	L← (L) + 1
INR M	34	@(HL) ← (@(HL)) +1
INX B	03	BC← (BC) + 1
INX D	13	DE← (DE) + 1
INX H	23	HL← (HL) + 1
INX SP	33	SP← (SP) + 1
JMP a16	C3	Перейти по адресу a16
JZ a16	CA	Перейти по адресу a16, если нуль (флаг нулевого результата Z=1)
JNZ a16	C2	Перейти по адресу a16, если не нуль (флаг нулевого результата Z=0)
JP a16	F2	Перейти по адресу a16, если плюс (флаг знака S=0)
JM a16	FA	Перейти по адресу a16, если минус (флаг знака S=1)
JC a16	DA	То же, если перенос (флаг переноса при арифметических операциях CF=1)
JNC a16	D2	То же, если нет переноса (флаг переноса при арифметических операциях CF=0)
JPE a16	EA	Перейти по адресу a16, если паритет четный (флаг четности результата P=1)
JPO a16	E2	Перейти по адресу a16, если паритет нечетный (флаг четности результата P=0)
LDA a16	3A	Загрузить в A данные из ячейки с адресом a16
LDAX B	0A	Загрузить в A данные из ячейки с адресом @(BC)
LDAX D	1A	Загрузить в A данные из ячейки с адресом @(DE)
LHLD a16	2A	Загрузить в HL содержимое ячейки с адресом a16
LXI B,d16	01	Загрузить в BC непосредственные данные d16

Команда	Код	Описание
LXI H,d16	21	Загрузить в HL непосредственные данные d16
LXI SP,d16	31	Загрузить в SP непосредственные данные d16
MOV A,B	78	Переслать в A из B (A← (B))
MOV A,C	79	Переслать в A из C
MOV A,D	7A	Переслать в A из D
MOV A,E	7B	Переслать в A из E
MOV A,H	7C	Переслать в A из H
MOV A,L	7D	Переслать в A из L
MOV A,M	7E	Переслать в A из @(HL)
MOV B,A	47	Переслать в B из A
MOV B,C	41	Переслать в B из C
MOV B,D	42	Переслать в B из D
MOV B,E	43	Переслать в B из E
MOV B,H	44	Переслать в B из H
MOV B,L	45	Переслать в B из L
MOV B,M	46	Переслать в B из @(HL)
MOV C,A	4F	Переслать в C из A
MOV C,B	48	Переслать в C из B
MOV C,D	4A	Переслать в C из D
MOV C,E	4B	Переслать в C из E
MOV C,H	4C	Переслать в C из H
MOV C,L	4D	Переслать в C из L
MOV C,M	4E	Переслать в C в @(HL)
MOV D,A	57	Переслать в D из A
MOV D,B	50	Переслать в D из B
MOV D,C	51	Переслать в D из C
MOV D,E	53	Переслать в D из E
MOV D,H	54	Переслать в D из H
MOV D,L	55	Переслать в D из L

Команда	Код	Описание
MOV D,M	56	Переслать в D из @(HL)
MOV E,A	5F	Переслать в E из A
MOV E,B	58	Переслать в E из B
MOV E,C	59	Переслать в E из C
MOV E,D	5A	Переслать в E из D
MOV E,H	5C	Переслать в E из H
MOV E,L	5D	Переслать в E из L
MOV E,M	5E	Переслать в E из @(HL)
MOV H,A	67	Переслать в H из A
MOV H,B	60	Переслать в H из B
MOV H,C	61	Переслать в H из C
MOV H,D	62	Переслать в H из D
MOV H,E	63	Переслать в H из E
MOV H,L	65	Переслать в H из L
MOV H,M	66	Переслать в H из @(HL)
MOV L,A	6F	Переслать в L из A
MOV L,B	68	Переслать в L из B
MOV L,C	69	Переслать в L из C
MOV L,D	6A	Переслать в L из D
MOV L,E	6B	Переслать в L из E
MOV L,H	6C	Переслать в L из H
MOV L,M	6E	Переслать в L из @(HL)
MOV M,A	77	Переслать в @(HL) из A
MOV M,B	70	Переслать в @(HL) из B
MOV M,C	71	Переслать в @(HL) из C
MOV M,D	72	Переслать в @(HL) из D
MOV M,E	73	Переслать в @(HL) из E
MOV M,H	74	Переслать в @(HL) из H
MOV M,L	75	Переслать в @(HL) из L
MVI A,d8	3E	Переслать d8 в A
MVI B,d8	06	Переслать d8 в B

Команда	Код	Описание
MVI C,d8	0E	Переслать d8 в C
MVI D,d8	16	Переслать d8 в D
MVI E,d8	1E	Переслать d8 в E
MVI H,d8	26	Переслать d8 в H
MVI L,d8	2E	Переслать d8 в L
MVI M,d8	36	Переслать d8 в @(HL)
NOP	00	Нет операции
ORA A	B7	Проверить A и сбросить перенос
ORA B	B0	Логическая операция B “ИЛИ” A
ORA C	B1	Логическая операция C “ИЛИ” A
ORA D	B2	Логическая операция D “ИЛИ” A
ORA E	B3	Логическая операция E “ИЛИ” A
ORA H	B4	Логическая операция H “ИЛИ” A
ORA L	B5	Логическая операция L “ИЛИ” A
ORA M	B6	Логическая операция M “ИЛИ” A
ORI d8	F6	Логическая операция d8 “ИЛИ” A
OUT pp	D3	Записать A в порт PP
PCHL	E9	Передать управление по адресу в HL
POP B	C1	Извлечь слово из стека в BC
POP D	D1	Извлечь слово из стека в DE
POP H	E1	Извлечь слово из стека в HL
POP PSW	F1	Извлечь слово из стека в PSW
PUSH B	C5	Поместить в стек содержимое BC
PUSH D	D5	Поместить в стек содержимое DE
PUSH H	E5	Поместить в стек содержимое HL
PUSH PSW	F5	Поместить в стек содержимое PSW
RAL	17	Циклический сдвиг CY + A влево
RAR	1F	Циклический сдвиг CY + A вправо
RLG	07	Сдвинуть A влево на один разряд с переносом

Команда	Код	Описание
RRG	0F	Сдвинуть A вправо на один разряд с переносом
RIM	20	Считать маску прерывания (в 8085)
RET	C9	Возврат из подпрограммы
RZ	C8	Возврат из подпрограммы, если FZ=1
RNZ	C0	Возврат из подпрограммы, если FZ=0
RP	F0	Возврат из подпрограммы, если FP=1
RM	F8	Возврат из подпрограммы, если FP=0
RC	D8	Возврат из подпрограммы, если FC=1
RNC	D0	Возврат из подпрограммы, если FC=0
RPE	E8	Возврат из подпрограммы, если паритет чет-ный
RPO	E0	Возврат из подпрограммы, если паритет нечет-ный
RST 0	C7	Запуск программы с адреса 0
RST 1	CF	Запуск программы с адреса 8h
RST 2	D7	Запуск программы с адреса 10h
RST 3	DF	Запуск программы с адреса 18h
RST 4	E7	Запуск программы с адреса 20h
RST 5	EF	Запуск программы с адреса 28h
RST 6	F7	Запуск программы с адреса 30h
RST 7	FF	Запуск программы с адреса 38h
SIM	30	Установить маску прерывания
SPHL	F9	Загрузить SP из HL
SHLD a16	22	Записать HL по адресу a16
STA a16	32	Записать A по адресу a16
STAX B	02	Записать A по адресу @(BC)
STAX D	12	Записать A по адресу @(DE)
STC	37	Установить флаг переноса (CF=1)
SUB A	9F	Вычесть A из A (очистить A)
SUB B	98	Вычесть B из A
SUB C	99	Вычесть C из A

Команда	Код	Описание
SUB D	9A	Вычесть D из A
SUB E	9B	Вычесть E из A
SUB H	9C	Вычесть H из A
SUB L	9D	Вычесть L из A
SUB M	9E	Вычесть M из A
SUI d8	DE	Вычесть d8 из A
SBB A	9F	Вычесть A из A (очистить A)
SBB B	98	Вычесть с заемом B из A
SBB C	99	Вычесть с заемом C из A
SBB D	9A	Вычесть с заемом D из A
SBB E	9B	Вычесть с заемом E из A
SBB H	9C	Вычесть с заемом H из A
SBB L	9D	Вычесть с заемом L из A
SBB M	9E	Вычесть с заемом M из A
SBI d8	DE	Вычесть с заемом d8 из A
XCHG	EB	Обмен содержимым DE и HL
XTHL	E3	Обмен содержимого вершины стека с содержимым HL
XRA A	AF	Исключающее ИЛИ A с A (очистка A)
XRA B	A8	Исключающее ИЛИ B с A
XRA C	A9	Исключающее ИЛИ C с A
XRA D	AA	Исключающее ИЛИ D с A
XRA E	AB	Исключающее ИЛИ E с A
XRA H	AC	Исключающее ИЛИ H с A
XRA L	AD	Исключающее ИЛИ L с A
XRA M	AE	Исключающее ИЛИ @(HL) с A
XRI d8	EE	Исключающее ИЛИ d8 с A

Библиографический список рекомендуемых источников

1. Гуров В.В. Архитектура микропроцессоров [Электронный ресурс] / В.В. Гуров. – Электрон. текстовые данные. – М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. – 115 с. – 978-5-9963-0267-3. – Режим доступа: http://www.iprbookshop.ru/56313.html

2. Электроника и микропроцессорная техника : учебник / В.Г. Гусев, Ю.М. Гусев. – Москва : КноРус, 2018. – 798 с. – Для бакалавров. – ISBN 978-5-406-06106-0. – Режим доступа: https://www.book.ru/book/926521

3. Вычислительные системы, сети и телекоммуникации (для бакалавров). Учебное пособие : учебное пособие / Л.П. Гудыно. – Москва : КноРус, 2019. – 372 с. – ISBN 978-5-406-06790-1. – Режим доступа: https://www.book.ru/book/930419

Министерство образования и науки Российской Федерации
Муромский институт (филиал)
федерального государственного бюджетного образовательного учреждения
высшего образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(МИ ВлГУ)**

Отделение среднего профессионального образования

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ПРОВЕДЕНИЯ ПРАКТИЧЕСКИХ ЗАНЯТИЙ ПО ДИСЦИПЛИНЕ АРХИТЕКТУРА МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВ

для студентов специальности

11.02.01 Радиоаппаратостроение

Программа подготовки специалистов среднего звена

Составитель
Романов Д.Н.

Муром 2018

Содержание

Практическая работа № 1

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ В ДВОИЧНЫХ КОДАХ

Цель работы:

- закрепление знаний о способах определения прямого, обратного и дополнительного кода числа;
- формирование практических навыков по нахождению прямого обратного и дополнительного кода числа;
- закрепление алгоритма определения числа по его прямому, обратному и дополнительному коду.

Теоретические сведения

Для представления информации в памяти ЭВМ (как числовой, так и нечисловой) используется двоичный способ кодирования.

Элементарная ячейка памяти ЭВМ имеет длину 8 бит (байт). Каждый байт имеет свой номер (его называют ***адресом***). Наибольшую последовательность бит, которую ЭВМ может обрабатывать как единое целое, называют ***машинным словом***. Длинные машинные слова зависит от разрядности процессора и может быть равной 16, 32 битам и т. д.

Для кодирования символов достаточно одного байта. При этом можно представить 256 символов (с десятичными кодами от 0 до 255). Набор символов персональных ЭВМ IBM PC чаще всего является расширением кода ASCII (American Standart Code for Information Interchange – стандартный американский код для обмена информацией).

Прямой код (представление в виде абсолютной величины со знаком) двоичного числа – это само двоичное число, в котором все цифры, изображающие его значение, записываются как в математической записи, а знак числа записывается двоичной цифрой.

Обратный код положительного числа совпадает с прямым, а при записи отрицательного числа все его цифры, кроме цифры, изображающей знак числа, заменяются на противоположные (0 заменяется на 1, а 1 – на 0).

Дополнительный код (представление в виде дополнения до двойки) положительного числа совпадает с прямым, а код отрицательного числа образуется как результат увеличения на 1 его обратного кода.

Ход работы

Представление числа в привычной форме «знак» - «величина», при которой старший разряд ячейки отводится под знак, а остальные - под запись числа в двоичной системе, называется прямым кодом двоичного числа. Например, прямой код двоичных чисел 1001 и -1001 для 8-разрядной ячейки равен 00001001 и 10001001 соответственно.

Положительные числа в ЭВМ всегда представляются с помощью прямого кода. Прямой код числа полностью совпадает с записью самого числа в ячейке машины. Вообще, положительные числа в прямом, обратном и дополнительном кодах изображаются одинаково - двоичными кодами с цифрой 0 в знаковом разряде.

Пример 1.

Число $1_{10}=1_2$

Число $127_{10}=1111111_2$

Прямой код отрицательного числа отличается от прямого кода соответствующего положительного числа лишь содержимым знакового разряда. Но от-

рицательные целые числа не представляются в ЭВМ с помощью прямого кода, для их представления используется так называемый дополнительный код.

Прямой код двоичного числа (а это либо мантисса, либо порядок) образуется по такому алгоритму:

1. Определить данное двоичное число - оно либо целое (порядок), либо правильная дробь (мантисса).
2. Если это дробь, то цифры после запятой можно рассматривать как целое число.
3. Если это целое и положительное двоичное число, то вместе с добавлением 0 в старший разряд число превращается в код. Для отрицательного двоичного числа перед ним ставится единица.

Пример 2.

Обратный код положительного двоичного числа совпадает с прямым кодом. Для отрицательного числа все цифры числа заменяются на противоположные (1 на 0, 0 на 1), а в знаковый разряд заносится единица.

Пример 3.

Дополнительный код положительного числа равен прямому коду этого числа. Дополнительный код отрицательного числа m равен 2k - |m|, где k - количество разрядов в ячейке. Также дополнительный код отрицательного числа образуется путём прибавления 1 к обратному коду.

При представлении целых чисел со знаком старший (левый) разряд отводится под знак числа, и под собственно число остаётся на один разряд меньше.

Алгоритм получения дополнительного кода отрицательного числа:

1. модуль отрицательного числа представить прямым кодом в k двоичных разрядах;
2. значение всех бит инвертировать: все нули заменить на единицы, а единицы на нули (таким образом, # получается k-разрядный обратный код исходного числа);
3. к полученному обратному коду прибавить единицу.

Дополнительный код используется для упрощения выполнения арифметических операций. Если бы вычислительная машина работала с прямыми кодами положительных и отрицательных чисел, то при выполнении арифметических операций следовало бы выполнять ряд дополнительных действий. Например, при сложении нужно было бы проверять знаки обоих операндов и определять знак результата. Если знаки одинаковые, то вычисляется сумма операндов и ей присваивается тот же знак. Если знаки разные, то из большего по абсолютной величине числа вычитается меньшее и результату присваивается знак большего числа. То есть при таком представлении чисел (в виде только прямого кода), операция сложения реализуется через достаточно сложный алгоритм. Если же отрицательные числа представлять в виде дополнительного кода, то операция сложения, в том числе и разного знака, сводится к из поразрядному сложению.

Для компьютерного представления целых чисел обычно используется один, два или четыре байта, то есть ячейка памяти будет состоять из восьми, шестнадцати или тридцати двух разрядов соответственно.

Для компьютерного представления целых чисел обычно используется один, два или четыре байта, то есть ячейка памяти будет состоять из восьми, шестнадцати или тридцати двух разрядов соответственно.

Пример 4.

Дополнительный код числа - 1

1	1	1	1	1	1	1	1

Дополнительный код числа - 127

1	0	0	0	0	0	0	1

Задания к практической работе

Задание 1. Запишите дополнительный код числа, интерпретируя его как восьмибитовое целое со знаком:

1. а) 115; б) -34; в) -70.
2. а) 81; б) -40; в) -24.
3. а) 98; б) -111; в) -95.
4. а) 89; б) -65; в) -8.
5. а) 64; б) -104; в) -47.
6. а) 55; б) -89; в) -22.
7. а) 95; б) -68; в) -77.
8. а) 82; б) -13; в) -109.

Задание 2. Запишите прямой код числа, интерпретируя его как шестнадцатибитовое целое без знака.

1. а) 22491; б) 23832.
2. а) 18509; б) 28180.
3. а) 19835; б) 22248.
4. а) 29407; б) 25342.
5. а) 30539; б) 26147.
6. а) 17863; б) 25893.
7. а) 28658; б) 29614.
8. а) 27898; б) 24268.

Задание 3. Запишите дополнительный код числа, интерпретирую его как шестнадцатибитовое целое со знаком.

1. а) 20850; б) -18641.
2. а) 28882; б) -19070.
3. а) 18156; б) -28844.
4. а) 23641; б) -23070.
5. а) 22583; б) -28122.
6. а) 24255; б) -26686.
7. а) 31014; б) -24013.
8. а) 19518; б) -16334.

Задание 4. Запишите в десятичной системе счисления целое число, если дан его дополнительный код.

1. а) 0011010111010110; б) 1000000110101110.
2. а) 0110010010010101; б) 1000011111110001.
3. а) 0111100011001000; б) 1111011101101101.
4. а) 0111011101000111; б) 1010110110101110.
5. а) 0100011011110111; б) 1011101001100000.
6. а) 0000010101011010; б) 1001110100001011.
7. а) 0001101111111001; б) 1011101101001101.
8. а) 0000110100001001; б) 1001110011000000.

Практическая работа № 2

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ С ПЛАВАЮЩЕЙ ТОЧКОЙ

Цель работы: Изучить формат IEEE - 754 .

Представление числа в нормализованном экспоненциальном виде.

Возьмем, к примеру, десятичное число 155,625

Представим это число в нормализованном экспоненциальном виде : $1{,}55625 \cdot 10^{+2} = 1{,}55625 \cdot exp_{10}^{+2}$

Число $1{,}55625 \cdot exp_{10}^{+2}$ состоит из двух частей: мантиссы M=1.55625 и экспоненты exp_{10}=+2

Если мантисса находится в диапазоне 1<=M<10, то число считается нормализованным.

Экспонента представлена основанием системы исчисления (в данном случае 10) и порядком (в данном случае +2).

Порядок экспоненты может иметь отрицательное значение, например число $0{,}0155625 = 1{,}55625 \cdot exp_{10}^{-2}$.

Представление числа в денормализованном экспоненциальном виде.

Возьмем, к примеру, десятичное число 155,625

Представим это число в денормализованном экспоненциальном виде: $0{,}155625 \cdot 10^{+3} = 0{,}155625 \cdot exp_{10}^{+3}$

Число $0{,}155625 \cdot exp_{10}^{+3}$ состоит из двух частей: мантиссы M=0,155625 и экспоненты exp_{10}=+3

Если мантисса находится в диапазоне 0,1<=M<1, то число считается денормализованным.

Экспонента представлена основанием системы исчисления (в данном случае 10) и порядком (в данном случае +3).

Порядок экспоненты может иметь отрицательное значение, например число $0{,}0155625 = 0{,}155625 \cdot exp_{10}^{-3}$.

Преобразование десятичного числа в двоичное число с плавающей точкой.

Наша задача сводится к представлению десятичного числа с плавающей точкой, в двоичное число с плавающей точкой в экспоненциальном нормализованном виде. Для этого разложим заданное число по двоичным разрядам:

$155{,}625 = 1 \cdot 2^{7} + 0 \cdot 2^{6} + 0 \cdot 2^{5} + 1 \cdot 2^{4} + 1 \cdot 2^{3} + 0 \cdot 2^{2} + 1 \cdot 2^{1} + 1 \cdot 2^{0} + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$

155,625 =128 + 0 + 0 + 16 + 8 + 0 + 2 + 1 + 0,5 + 0 + 0,125

$155{,}625_{10} = 10011011{,}101_{2}$ - число в десятичной и в двоичной системе с плавающей точкой

Приведем полученное число к нормализованному виду в десятичной и двоичной системе:

$1{,}55625 \cdot exp_{10}^{+2} = 1{,}0011011101 \cdot exp_{2}^{+111}$

В результате мы получили основные составляющие экспоненциального нормализованного двоичного числа:

Мантиссу M=1,0011011101

Экспоненту exp_2= +111

Преобразование двоичного нормализованного числа в 32 битный формат IEEE 754

Основное применение в технике и программирование получили форматы 32 и 64 бита.

Например, в VB используют типы данных single (32 бита) и double (64 бита).

В Си аналогично используют float (32 бита) и double (64 бит)

Рассмотрим преобразование двоичного числа 10011011,101 в формат single-precision (32 бита) стандарта IEEE 754.

Остальные форматы представления чисел в IEEE 754 являются увеличенной копией single-precision.

Чтобы представить число в формате single-precision IEEE 754 необходимо привести его к двоичному нормализованному виду. В §3 мы проделали это преобразование над числом 155,625. Теперь рассмотрим, как двоичное нормализованное число преобразуется к 32 битному формату IEEE 754.

Описание преобразования в 32 битный формат IEEE 754:

1. Число может быть + или - . Поэтому отводится 1 бит для обозначения знака числа: 0-положительное, 1-отрицательное. Этот самый старший бит в 32 битной последовательности.
2. Далее пойдут биты экспоненты, для этого выделяют 1 байт (8 бит). Экспонента может быть, как и число, со знаком + или -. Для определения знака экспоненты, чтобы не вводить ещё один бит знака, добавляют смещение к экспоненте в половину байта +127(0111 1111). То есть, если наша экспоната = +7 (+111 в двоичной), то смещенная экспонента = 7+127=134. А если бы, наша экспонента была -7 , то смещенная экспонента=127-7 =120. Смещенную экспоненту записывают в отведенные 8 бит. При этом, когда нам будет нужно получить экспоненту двоичного числа, мы просто отнимем 127 от этого байта.
3. Оставшиеся 23 бита отводят для мантиссы. Но, у нормализованной двоичной мантиссы первый бит всегда равен 1, так как число лежит в диапазоне 1<=M<2. Нет смыла, записывать единицу в отведенные 23 бита, поэтому в отведенные 23 бита записывают остаток от мантиссы.

В таблице представлено десятичное число 155,625 в 32-х битном формате IEEE754

31 бит	30-23 биты	22-0 биты	IEEE 754
0	1000 0110	001 1011 1010 0000 0000 0000	431BA000 (hex)
0(dec)	134(dec)	1810432(dec)	
знак числа	смещенная экспонента	остаток от мантиссы	число 155,625 в формате IEEE754

Преобразования числа формата 32 бит IEEE 754 в десятичное число

Чтобы записать число в стандарте IEEE 754 или восстановить его, необходимо знать три параметра:

- S- бит знака (31-й бит)
- E- смещенная экспонента (30-23 биты)
- M - остаток от мантиссы (22-0 биты)

Это целые числа которые записанные в числе IEEE 754 в двоичном виде.

Приведём формулу для получения десятичного числа из числа IEEE754 одинарной точности:

$$F = (-1)^S \cdot 2^{E-127} \cdot \left(1 + \frac{M}{2^{23}}\right)$$

Общая формула вычисления десятичных чисел с плавающей точкой, из чисел, представленных в стандарте IEEE754:

Задания к практической работе

Задание 1. Приведите число кнормализованной форме и запишите в формате IEEE754

Задание 2. Приведите число кнормализованной форме и запишите в формате IEEE754

Задание 3. Преобразуйте число в десятичную систему счисления

№ вари-анта	Задание №1	Задание №2	Задание №3
1	240,125	0,24125	0100 0010 1111 1010 1011 0011 0011 0011
2	408,5	0,40185	1100 0010 0101 1110 1000 0000 0000 0000
3	336,375	0,33375	0100 0011 0101 0111 1111 1101 1011 0010
4	289,25	0,281925	1100 0011 0010 0000 0101 1001 1001 1001
5	300,0625	0,30625	0100 0100 0000 1101 1001 0000 0000 0000
6	399,5	0,39095	1100 0011 1100 1000 0100 0000 0000 0000
7	588,25	0,581825	0100 0011 0000 0000 1000 1110 0001 0100
8	512,875	0,51275	1100 0010 1001 0111 1000 0000 0000 0000
9	480,625	0,48625	0100 0011 0001 0100 0011 1010 1110 0001
10	462,375	0,46375	1100 0011 1001 1100 0001 1011 0100 0011
11	390,3125	0,393125	1100 0011 0100 0010 1110 0001 1110 0001
12	533,125	0,53325	0100 0010 1111 1010 1110 0001 1110 0001
13	437,25	0,430725	1100 0010 0101 1110 0011 1010 1110 0001
14	416,375	0,41375	0100 0011 0101 0111 1000 0000 0000 0000
15	375,875	0,370875	1100 0011 0010 0000 1000 1110 0001 0100
16	123,75	0,121375	0100 0100 0000 1101 0100 0000 0000 0000
17	75,25	0,71525	1100 0011 1100 1000 1001 0000 0000 0000
18	101,5	0,12015	0100 0011 0000 0000 0101 1001 1001 1001
19	112,125	0,11225	1100 0010 1001 0111 1111 1101 1011 0010
20	95,375	0,950375	0100 0011 0001 0100 1000 0000 0000 0000

21	115,625	0,1105625	1100 0011 1001 1100 1011 0011 0011 0011
22	211,875	0,21875	1111 1010 1011 0011 1001 1100 0001 1011
23	99,0625	0,99625	0101 1110 1000 0000 0100 0010 1110 0001
24	65,1875	0,65185	0101 0111 1111 1101 1111 1010 1110 0001
25	85,3125	0,85325	0010 0000 0101 1001 0101 1110 0011 1010
26	512,675	0,5121675	0000 1101 1001 0000 0010 0000 1000 1110
27	480,825	0,48825	1100 1000 0100 0000 0000 110 10100 0000
28	462,175	0,4620175	0000 0000 1000 1110 1100 1000 1001 0000
29	90,3125	0,93125	1001 0111 1000 0000 0000 0000 0101 1001
30	533,625	0,5331625	0001 0100 0011 1010 1001 0111 1111 1101

Практическая работа № 3

ВНУТРЕНЕЕ УСТРОЙСТВО МПУ

Цель работы: Изучить внутреннее устройство микропроцессорного устройства.

Структура 32-разрядного универсального микропроцессора

Рассмотрение архитектуры *IA-32* начнем с микропроцессора i486. В нем впервые появились те блоки, которых не было на кристалле первого 32-разрядного микропроцессора i386, - кэш-*память* и *процессор* обработки чисел с плавающей точкой. Именно его архитектуру можно рассматривать как базовую для *IA-32*. Структура микропроцессора i486 представлена на рис. 1.3.

Рассмотрим состав и назначение основных блоков этого микропроцессора.

Процессор обработки чисел с фиксированной точкой содержит 32-разрядное *АЛУ* и блок **регистров общего назначения**. *АЛУ*предназначено для обработки двоичных чисел длиной 1, 2 или 4 байта без знака или со знаком, а также двоично-десятичных чисел, не превышающих 99. Двоичные числа со знаком представляются в *дополнительном коде*. Блок регистров общего назначения содержит восемь 32-разрядных регистров, часть из которых допускает 16- и 8-разрядное обращение.

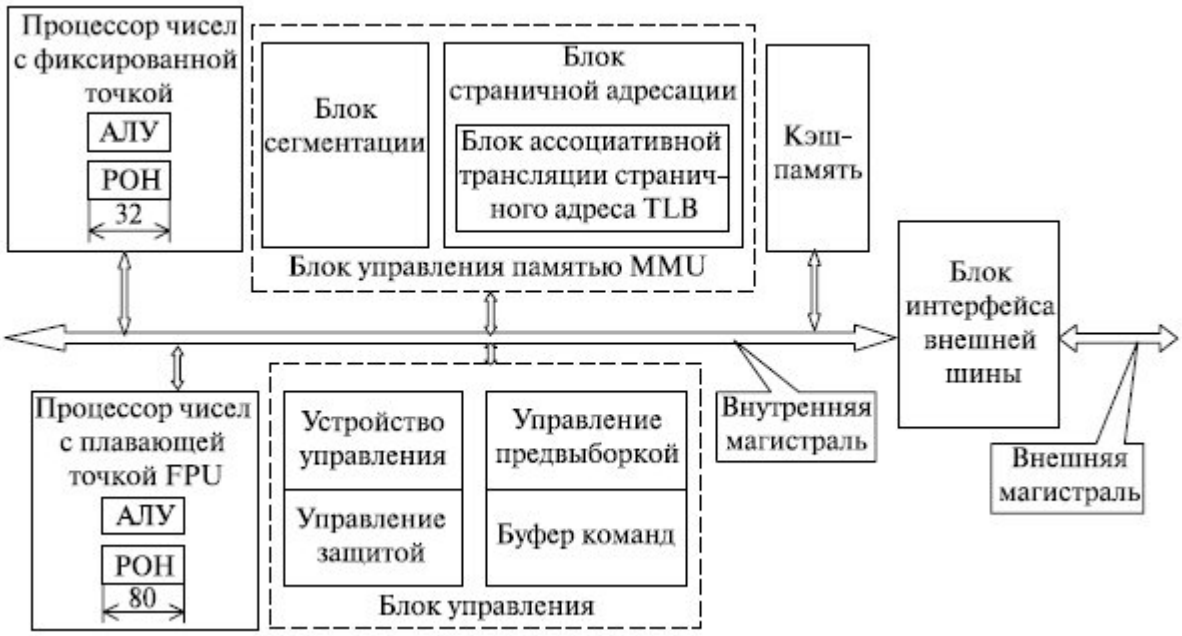


Рис. 1.3. Структура универсального микропроцессора

Процессор обработки чисел с плавающей точкой состоит из 80-разрядного *АЛУ*, блока из восьми 80-разрядных регистров общего назначения, а также управляющих регистров. Главным образом он предназначен для обработки чисел с плавающей точкой, но также используется для обработки целых чисел со знаком длиной 8 *байт* и двоично-десятичных чисел величиной от 100 до 99…9 (18 цифр). На первых этапах развития *SIMD*-обработки регист-

ры **FPU** использовались для хранения операндов, представленных в новых форматах.

Блок управления памятью (*Memory Management* Unit - *MMU*) состоит из двух основных блоков в соответствии с *организацией памяти*.

В общем случае *память* в микропроцессоре делится на *сегменты*, которые, в свою *очередь*, делятся на страницы. В соответствии с этим, *MMU*содержит блок *сегментации* (или блок сегментного преобразования адреса) и блок *страничного преобразования*, в состав которого входит так называемый **буфер ассоциативной трансляции адресов стра- ниц** (*TLB*).

Кэш-память представляет собой промежуточную ступень между оперативной памятью и регистрами микропроцессора и предназначена для хранения наиболее часто используемой информации.

В состав **блока управления** входят:

- собственно *устройство управления*, то есть та классическая схема, которая под действием кода команды вырабатывает набор управляющих сигналов, поступающих на разные узлы как самого микропроцессора, так и на блок интерфейса внешней шины;
- управление *защитой памяти*: обеспечивает аппаратную защиту программ и данных при управлении памятью и по привилегиям;
- блок управления предвыборкой команд: реализует опережающее заполнение буфера команд, представляющего собой некоторую*буферную память*. Буфер команд имеет емкость 32 байта и заполняется командами из следующих ячеек памяти команд по мере своего освобождения. Этим обеспечивается ускорение обработки микропроцессором следующей команды. Данный блок подвергался, пожалуй, наиболее существенным переработкам по мере развития архитектуры *IA-32* - причина в широком последующем использовании конвейерной организации работы МП и связанной с этим необходимости постоянного совершенствования блока предсказания адреса следующей команды.

Блок интерфейса внешней шины осуществляет электрическое согласование параметров внутренней магистрали с сигналами внешних*магистралей*, формирование необходимых сигналов на внешнюю *магистраль* и прием сигналов извне. Внешняя *магистраль* микропроцессора состоит из *шины адреса*, *шины данных* и сигналов управления:

- *шина данных* имеет ширину 32 разряда;
- 32-разрядный адрес передается по 34-разрядной шине A31...A2+(B3,B2,B1,B0). Чтобы с минимальными потерями согласовывать 32-разрядную *шину данных* с передачей данных меньшей разрядности, младшие разряды адреса (A1 и A0) передаются в дешифрированном виде (B3, B2, B1, B0). Они показывают, какие байты из 32-разрядной *шины данных* в данный момент реально востребованы: 1 байт, 2 младших байта, 2 старших байта либо все 32 разряда данных;
- шина управления - 32-разрядная. По ней передаются сигналы записи и чтения содержимого оперативной памяти и внешних устройств, сигналы *запросов прерываний*, прямого доступа к памяти и т. д.

Особый интерес представляют три режима работы микропроцессора: реальный, защищенный и режим виртуального МП i8086. В **реальном режиме** обеспечивается совместимость на уровне объектных кодов с микропроцессором i8086 и микропроцессором i286, работающем в реальном режиме. В этом режиме *архитектура* 32-разрядного микропроцессора почти полностью идентична архитектуре 16-разрядного МП. Для программиста же он вообще представляется как МП i8086, выполняющий написанные программы с большей скоростью и обладающий расширенной системой команд и регистрами. Благодаря этим качествам *фирма* Intel сохранила прежних клиентов, которые хотели модернизировать свои системы, не отказываясь от имевшегося задела в области программного обеспечения, и привлекла тех, кому изначально требовалась высокая скорость обработки информации.

Одно из основных ограничений реального режима было связано с предельной емкостью адресуемой памяти, равной 1 Мбайт. От него свободен**защищенный режим**, позволяющий воспользоваться всеми преимуществами архитектуры нового МП. Размер адресного пространства в этом случае увеличивается до 4 Гбайт, а общий объем поддерживаемого адресного пространства - до 64 терабайт (1 Тбайт = 2^{40} *байт*). МП, работающие в защищенном режиме, обладают более высоким быстродействием и возможностями организации истинной *многозадачности*.

Наконец, **режим виртуального МП** открывает возможность одновременного исполнения программ, написанных для МП i8086, i286 и i386.

Поскольку *емкость памяти*, адресуемой микропроцессором, не ограничена значением 1 Мбайт, этот режим позволяет формировать несколько виртуальных сред i8086.

Практическая работа № 4

ЦИКЛЫ ОБМЕНА ДАННЫМИ В МПУ

Цель работы: Изучить основные фазы циклов обмена данными в микропроцессорных системах.

Обмен информацией в микропроцессорных системах происходит в *циклах обмена информацией*. Под циклом обмена информацией понимается временной интервал, в течение которого происходит выполнение одной элементарной операции обмена по шине. Например, пересылка кода данных из процессора в память или же пересылка кода данных из устройства ввода/вывода в процессор. В пределах одного цикла также может передаваться и несколько кодов данных, даже целый массив данных, но это встречается реже.

Циклы обмена информацией делятся на два основных типа:

- ✓ **Цикл записи (вывода)**, в котором процессор записывает (выводит) информацию;
- ✓ **Цикл чтения (ввода)**, в котором процессор читает (вводит) информацию.

В некоторых микропроцессорных системах существует также цикл «чтение-модификация-запись» или же «ввод-пауза-вывод». В этих циклах процессор сначала читает информацию из памяти или устройства ввода/вывода, затем как-то преобразует ее и снова записывает по тому же адресу. Например, процессор может прочитать код из ячейки памяти, увеличить его на единицу и снова записать в эту же ячейку памяти. Наличие или отсутствие данного типа цикла связано с особенностями используемого процессора.

Особое место занимают циклы прямого доступа к памяти (если режим ПДП в системе предусмотрен) и циклы запроса и предоставления прерывания (если прерывания в системе есть). Когда в дальнейшем речь пойдет о таких циклах, это будет специально оговорено.

Во время каждого цикла устройства, участвующие в обмене информацией, передают друг другу информационные и управляющие сигналы в строго установленном порядке или, как еще говорят, в соответствии с принятым *протоколом обмена информацией*.

Длительность цикла обмена может быть постоянной или переменной, но она всегда включает в себя несколько периодов сигнала тактовой частоты системы. То есть даже в идеальном случае частота чтения информации процессором и частота записи информации оказываются в несколько раз меньше тактовой частоты системы.

Чтение кодов команд из памяти системы также производится с помощью циклов чтения. Поэтому в случае одношинной архитектуры на системной магистрали чередуются циклы чтения команд и циклы пересылки (чтения и записи) данных, но протоколы обмена остаются неизменными независимо от того, что передается – данные или команды. В случае двухшинной архитектуры циклы чтения команд и записи или чтения данных разделяются по разным шинам и могут выполняться одновременно.

Шины микропроцессорной системы

Прежде чем переходить к особенностям циклов обмена, остановимся подробнее на составе и назначении различных шин микропроцессорной системы.

Как уже упоминалось, в системную магистраль (системную шину) микропроцессорной системы входит три основные информационные шины: адреса, данных и управления.

Шина данных – это основная шина, ради которой и создается вся система. Количество ее разрядов (линий связи) определяет скорость и эффективность информационного обмена, а также максимально возможное количество команд.

Шина данных всегда двунаправленная, так как предполагает передачу информации в обоих направлениях. Наиболее часто встречающийся тип выходного каскада для линий этой шины – выход с тремя состояниями.

Обычно шина данных имеет 8, 16, 32 или 64 разряда. Понятно, что за один цикл обмена по 64-разрядной шине может передаваться 8 байт информации, а по 8-разрядной – только один байт. Разрядность шины данных определяет и разрядность всей магистрали. Например, когда говорят о 32-разрядной системной магистрали, подразумевается, что она имеет 32-разрядную шину данных.

Шина адреса – вторая по важности шина, которая определяет максимально возможную сложность микропроцессорной системы, то есть допустимый объем памяти и, следовательно, максимально возможный размер программы и максимально возможный объем запоминаемых данных. Количество адресов, обеспечиваемых шиной адреса, определяется как 2N, где N – количество разрядов. Например, 16-разрядная шина адреса обеспечивает 65 536 адресов. Разрядность шины адреса обычно кратна 4 и может достигать 32 и даже 64. Шина адреса может быть однонаправленной (когда магистралью всегда управляет только процессор) или двунаправленной (когда процессор может временно передавать управление магистралью другому устройству, например контроллеру ПДП). Наиболее часто используются типы выходных каскадов с тремя состояниями или обычные ТТЛ (с двумя состояниями).

Как в шине данных, так и в шине адреса может использоваться *положительная логика* или *отрицательная логика*. При положительной логике высокий уровень напряжения соответствует логической единице на соответствующей линии связи, низкий – логическому нулю. При отрицательной логике – наоборот. В большинстве случаев уровни сигналов на шинах – ТТЛ.

Для снижения общего количества линий связи магистрали часто применяется ***мультиплексирование*** шин адреса и данных. То есть одни и те же линии связи используются в разные моменты времени для передачи как адреса, так и данных (в начале цикла – адрес, в конце цикла – данные). Для фиксации этих моментов (стробирования) служат специальные сигналы на шине управления. Понятно, что мультиплексированная шина адреса/данных обеспечивает меньшую скорость обмена, требует более длительного цикла обмена (рис. 4.1). По типу шины адреса и шины данных все магистрали также делятся на мультиплексированные и немультиплексированные.

Рис. 4.1. Мультиплексирование шин адреса и данных.

В некоторых мультиплексированных магистралях после одного кода адреса передается несколько кодов данных (массив данных). Это позволяет существенно повысить быстродействие магистрали. Иногда в магистралях применяется частичное мультиплексирование, то есть часть разрядов данных передается по немультиплексированным линиям, а другая часть – по мультиплексированным с адресом линиям.

Шина управления – это вспомогательная шина, управляющие сигналы на которой определяют тип текущего цикла и фиксируют моменты времени, соответствующие разным частям или стадиям цикла. Кроме того, управляющие сигналы обеспечивают согласование работы процессора (или другого хозяина магистрали, задатчика, master) с работой памяти или устройства ввода/вывода (устройства-исполнителя, slave). Управляющие сигналы также обслуживают запрос и предоставление прерываний, запрос и предоставление прямого доступа.

Сигналы шины управления могут передаваться как в положительной логике (реже), так и в отрицательной логике (чаще). Линии шины управления могут быть как однонаправленными, так и двунаправленными. Типы выходных каскадов могут быть самыми разными: с двумя состояниями (для однонаправленных линий), с тремя состояниями (для двунаправленных линий), с открытым коллектором (для двунаправленных и мультиплексированных линий).

Самые главные управляющие сигналы – это стробы обмена, то есть сигналы, формируемые процессором и определяющие моменты времени, в которые производится пересылка данных по шине данных, обмен данными. Чаще всего в магистрали используются два различных строба обмена:

- ✓ Строб записи (вывода), который определяет момент времени, когда устройство-исполнитель может принимать данные, выставленные процессором на шину данных;
- ✓ Строб чтения (ввода), который определяет момент времени, когда устройство-исполнитель должно выдать на шину данных код данных, который буде прочитан процессором.

При этом большое значение имеет то, как процессор заканчивает обмен в пределах цикла, в какой момент он снимает свой строб обмена. Возможны два пути решения (рис. 4.2):

- ✓ При ***синхронном*** **обмене** процессор заканчивает обмен данными самостоятельно, через раз и навсегда установленный временной интервал выдержки ($t_{выд}$), то есть без учета интересов устройства-исполнителя;
- ✓ При ***асинхронном*** **обмене** процессор заканчивает обмен только тогда, когда устройство-исполнитель подтверждает выполнение операции специальным сигналом (так называемый режим handshake – рукопожатие).

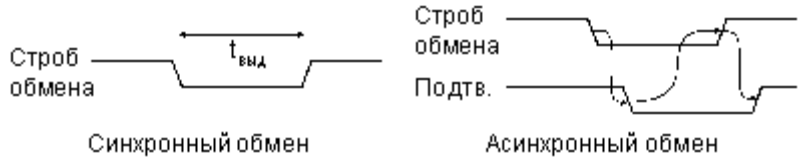


Рис. 2.2. Синхронный обмен и асинхронный обмен.

Достоинства синхронного обмена – более простой протокол обмена, меньшее количество управляющих сигналов. Недостатки – отсутствие гарантии, что исполнитель выполнил требуемую операцию, а также высокие требования к быстродействию исполнителя.

Достоинства асинхронного обмена – более надежная пересылка данных, возможность работы с самыми разными по быстродействию исполнителями. Недостаток – необходимость формирования сигнала подтверждения всеми исполнителями, то есть дополнительные аппаратурные затраты.

Какой тип обмена быстрее, синхронный или асинхронный? Ответ на этот вопрос неоднозначен. С одной стороны, при асинхронном обмене требуется какое-то время на выработку, передачу дополнительного сигнала и на его обработку процессором. С другой стороны, при синхронном обмене приходится искусственно увеличивать длительность строба обмена для соответствия требованиям большего числа исполнителей, чтобы они успевали обмениваться ин-

формацией в темпе процессора. Поэтому иногда в магистрали предусматривают возможность как синхронного, так и асинхронного обмена, причем синхронный обмен является основным и довольно быстрым, а асинхронный применяется только для медленных исполнителей.

По используемому типу обмена магистрали микропроцессорных систем также делятся на синхронные и асинхронные.

Практическая работа № 5

ПРОГРАММНЫЙ РЕЖИМ РАБОТЫ МПУ.

Цель работы: Изучить принципы обмена данными в микропроцессорном устройстве в программном режиме работы.

Циклы программного обмена

Рассмотрим для примера два довольно типичных случая программного обмена по магистрали микропроцессорной системы.

Первый пример – это обмен по мультиплексированной асинхронной магистрали Q-bus, предложенной фирмой DEC и широко применявшейся в микрокомпьютерах и промышленных контроллерах. Упрощенные временные диаграммы циклов чтения (ввода) и записи (вывода) по этой магистрали приведены на рис. 5.1 и 5.2.

Отметим, что в дальнейшем тексте знак «минус» перед названием сигнала говорит о том, что активный уровень сигнала низкий, пассивный – высокий, то есть сигнал отрицательный. Если минуса перед названием сигнала нет, то сигнал положительный, его низкий уровень пассивный, а высокий – активный.

На шине адреса/данных (AD) в начале цикла обмена (в фазе адреса) процессор (задатчик) выставляет код адреса. На этой шине используется отрицательная логика. Средний уровень сигналов на шине AD обозначает, что состояния сигналов на шине в данные временные интервалы не важны. Для стробирования адреса используется отрицательный синхросигнал -SYNC, выставляемый также процессором. Его передний (отрицательный) фронт соответствует действительности кода адреса на шине AD. Фаза адреса одинакова в обоих циклах записи и чтения.

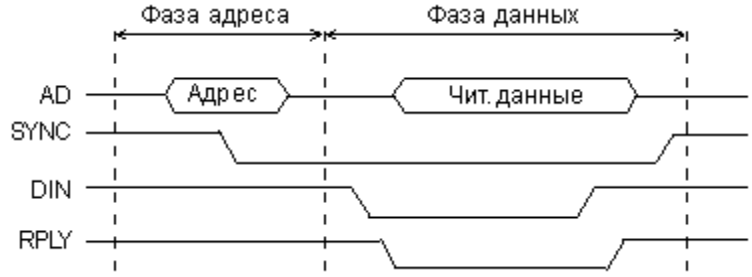


Рис. 5.1. Цикл чтения на магистрали Q-bus.

Получив (распознав) свой код адреса, устройство ввода/вывода или память (исполнитель) готовится к проведению обмена. Через некоторое время после начала (отрицательного фронта) сигнала -SYNC процессор снимает адрес и начинает фазу данных.

Рис. 5.2. Цикл записи на магистрали Q-bus.

В фазе данных цикла чтения (рис. 5.2) процессор выставляет сигнал строба чтения данных -DIN, в ответ на который устройство, к которому обращается процессор (исполнитель), должно выставить свой код данных (читаемые дан-

ные). Одновременно это устройство должно подтвердить выполнение операции сигналом подтверждения обмена -RPLY.

Для сигнала -RPLY используется тип выходного каскада ОК, чтобы не было конфликтов между устройствами-исполнителями. Процессор, получив сигнал -RPLY, заканчивает цикл обмена. Для этого он снимает сигнал -DIN и сигнал -SYNC. Устройство-исполнитель в ответ на снятие сигнала -DOUT должно снять код данных с шины AD и закончить сигнал подтверждения -RPLY. После этого процессор снимает сигнал -SYNC.

В фазе данных цикла записи (рис. 5.2) процессор выставляет на шину AD код записываемых данных и сопровождает его отрицательным сигналом строба записи данных -DOUT. Устройство-исполнитель должно по этому сигналу принять данные от процессора и сформировать сигнал подтверждения обмена -RPLY. Процессор, получив сигнал -RPLY, заканчивает цикл обмена. Для этого он снимает код данных с шины AD и сигнал -DOUT. Устройство-исполнитель в ответ на снятие сигнала -DIN должно закончить сигнал подтверждения -RPLY. После этого процессор снимает сигнал -SYNC.

То есть на данной магистрали адрес передается синхронно (без подтверждения его получения исполнителем), а данные передаются асинхронно, с обязательным подтверждением их выдачи или приема исполнителем. Отсутствие сигнала подтверждения -RPLY в течение заданного времени воспринимается процессором как аварийная ситуация. В принципе возможна и асинхронная передача адреса, что увеличивает надежность обмена, хотя может снижать его скорость.

Помимо циклов чтения и записи на магистрали Q-bus используются также и циклы типа «ввод-пауза-вывод» («чтение-модификация-запись»). Упрощенная временная диаграмма этого цикла представлена на рис. 5.3.

Рис. 5.3. Цикл «ввод-пауза-вывод» на магистрали Q-bus.

В этом цикле адресная фаза производится точно так же, как и в циклах чтения (ввода) и записи (вывода). Но в фазе данных процессор производит сначала чтение из заданного в адресной фазе адреса, а потом запись в тот же самый адрес. Для чтения используется строб чтения -DIN, а для записи – строб записи -DOUT. В ответ на сигнал -DIN устройство-исполнитель выдает свои данные на шину AD, а по сигналу -DOUT – принимает данные с шины AD. Как и в циклах чтения и записи, устройство-исполнитель подтверждает выполнение каждой операции сигналом подтверждения -RPLY. Понятно, что цикл «ввод-пауза-вывод» требует больше времени, чем каждый из циклов чтения или записи, но меньше времени, чем два последовательно произведенных цикла чтения и записи (так как для него нужна только одна адресная фаза). Сигнал -SYNC вырабатывается процессором в начале цикла «ввод-пауза-вывод» и держится до окончания всего цикла.

В качестве второго примера рассмотрим циклы обмена на синхронной немультиплексированной магистрали ISA (Industrial Standard Architecture), предложенной фирмой IBM и широко используемой в персональных компьютерах.

Упрощенные циклы записи в устройство ввода/вывода и чтения из устройства ввода/вывода приведены на рис. 5.4 и 5.5.

Оба цикла начинаются с выставления процессором (задатчиком) кода адреса на шину адреса SA (логика на этой шине положительная). Адрес остается на шине SA до конца цикла. Фаза адреса, одинаковая для обоих циклов, заканчивается с началом строба обмена данными -IOR или -IOW. В течение фазы адреса устройство-исполнитель должно принять код адреса и распознать или не распознать его. Если адрес распознан, исполнитель готовится к обмену.

В фазе данных цикла чтения (рис. 5.4) процессор выставляет отрицательный сигнал чтения данных из устройства ввода/вывода -IOR. В ответ на него устройство-исполнитель должно выдать на шину данных SD свой код данных (читаемые данные). Логика на шине данных положительная. Через установленное время строб обмена -IOR снимается процессором, после чего снимается также и код адреса с шины SA. Цикл заканчивается без учета быстродействия исполнителя.

Рис. 5.4. Цикл чтения из УВВ на магистрали ISA.

Рис. 5.5. Цикл записи в УВВ на магистрали ISA.

Но так происходит только в случае основного, синхронного обмена. Кроме него на магистрали ISA также предусмотрена возможность асинхронного обмена. Для этого применяется сигнал готовности канала (магистрали) I/O CH RDY. Тип выходного каскада для данного сигнала – ОК, для предотвращения конфликтов между устройствами-исполнителями. При синхронном обмене сигнал I/O CH RDY всегда положительный. Но медленное устройство-исполнитель, не успевающее работать в темпе процессора, может этот сигнал снять, то есть сделать нулевым сразу после начала строба обмена. Тогда процессор до того момента, пока сигнал I/O CH RDY не станет снова положительным, приостанавливает завершение цикла, продлевает строб обмена. Конечно, слишком большая длительность этого сигнала рассматривается как аварийная ситуация. Для простоты понимания можно считать, что устройство-исполнитель формирует в данном случае отрицательный сигнал неготовности завершить обмен. На время этого сигнала обмен на магистрали приостанавливается.

Принципиальное отличие асинхронного обмена по магистрали ISA от асинхронного обмена по магистрали Q-bus состоит в следующем. Если в случае Q-bus сигнал подтверждения обязателен, и его должен формировать каждый исполнитель, то в случае ISA сигнал о неготовности исполнитель может не формировать, если он успевает работать в темпе процессора. Зато в случае Q-bus к концу цикла обмена процессор всегда уверен, что устройство-

исполнитель выполнило требуемую операцию, а в случае ISA такой уверенности нет.

В фазе данных цикла записи по магистрали ISA (рис. 5.5) процессор выставляет на шину данных SD код записываемых данных и сопровождает их стробом записи данных в устройство ввода/вывода -IOW. Получив этот сигнал, устройство-исполнитель должно принять с шины SD код записываемых данных. Если оно не успевает сделать это в темпе процессора, то оно может снять на нужное время сигнал I/O CH RDY после получения переднего фронта сигнала -IOW. Тогда процессор приостановит окончание цикла записи.

Практическая работа № 6

РЕЖИМ ОБРАБОТКИ ПРЕРЫВАНИЙ.

Цель работы: Изучить принципы обработки прерываний и организацию обработки прерываний при помощи специализированных команд языка ассемблер для процессора 8085.

Обмен по прерываниям используется тогда, когда необходима реакция микропроцессорной системы на какое-то внешнее событие, на приход внешнего сигнала. В случае компьютера внешним событием может быть, например, нажатие на клавишу клавиатуры или приход по локальной сети пакета данных. Компьютер должен реагировать на это, соответственно, выводом символа на экран или же чтением и обработкой принятого по сети пакета.

В общем случае организовать реакцию на внешнее событие можно тремя различными путями:

- ✓ с помощью постоянного программного контроля факта наступления события (так называемый метод опроса флага или polling);
- ✓ с помощью прерывания, то есть насильственного перевода процессора с выполнения текущей программы на выполнение экстренно необходимой программы;
- ✓ с помощью прямого доступа к памяти, то есть без участия процессора при его отключении от системной магистрали.

Команды прерываний относятся к командам перехода с возвратом. Эти команды в качестве входного операнда требуют номер прерывания (адрес вектора). Обслуживание таких переходов осуществляется точно так же, как и аппаратных прерываний. То есть для выполнения данного перехода процессор обращается к таблице векторов прерываний и получает из нее по номеру прерывания адрес памяти, в который ему необходимо перейти. Адрес вызова прерывания и содержимое регистра состояния процессора (PSW) сохраняются в стеке. Сохранение PSW – важное отличие команд прерывания от команд переходов с возвратом.

Команды прерываний во многих случаях оказываются удобнее, чем обычные команды переходов с возвратом. Сформировать таблицу векторов прерываний можно один раз, а потом уже обращаться к ней по мере необходимости. Номер прерывания соответствует номеру подпрограммы, то есть номеру функции, выполняемой подпрограммой. Поэтому команды прерывания гораздо чаще включаются в системы команд процессоров, чем обычные команды переходов с возвратом.

Для возврата из подпрограммы, вызванной командой прерывания, используется команда возврата из прерывания. Эта команда извлекает из стека сохраненное там значение счетчика команд и регистра состояния процессора (PSW).

Отметим, что у некоторых процессоров предусмотрены также команды условных прерываний, например, команда прерывания при переполнении.

Вызов обработки прерывания можно организовать тремя способами:

1. Командами вызова.
    - ♦ RST 0 – Запуск программы с адреса 0h
    - ♦ RST 1 – Запуск программы с адреса 8h
    - ♦ RST 2 – Запуск программы с адреса 10h
    - ♦ RST 3 – Запуск программы с адреса 18h
    - ♦ RST 4 – Запуск программы с адреса 20h

- ♦ RST 5 – Запуск программы с адреса 28h
- ♦ RST 6 – Запуск программы с адреса 30h
- ♦ RST 7 – Запуск программы с адреса 38h

2. Нажатием кнопки INTR. После нажатия кнопки появляется окно с запросом номера прерывания.
   - ♦ #1 – RST 0
   - ♦ #2 – RST 1
   - ♦ #3 – RST 2
   - ♦ #4 – RST 3
   - ♦ #5 – RST 4
   - ♦ #6 – RST 5
   - ♦ #7 – RST 6
   - ♦ #8 – RST 7
3. Нажатием кнопок RST5.5, RST6.5, RST7.5.
   - ♦ Кнопка RST5.5 – Запуск программы с адреса 2Ch
   - ♦ Кнопка RST6.5 – Запуск программы с адреса 34h
   - ♦ Кнопка RST7.5 – Запуск программы с адреса 3Ch

Пример.

Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 01H и вывод этого символа в порт 02H. Окончанием ввода служит ввод символа **Enter.** Программа обработки располагается по адресу 0034H. Запрос на обработку должен осуществляться нажатием кнопки **RST6.5.**

PC	Мнемоника	Примечание
0000	JMP 0100H	// Переход к основной программе
Программа обработки прерываний		
0003	.ORG 0034H	// Установка регистра **PC** равное 0034H
0034	L1: IN 01H	// Прием кода ASCII из порта 01H.
0036	CPI 0DH	// Сравнение принятого кода с кодом символа **Enter**
0038	OUT 02H	// Пересылка приятого кода ASCII в порт 02
003A	JNZ L1	// Если принятый код равен коду символа **Enter**
003D	EI	// Разрешение прерываний
003E	RET	// Возврат из команды обработки прерываний
Основная программа		
003F	.ORG 0100H	// Установка регистра **PC** равное 0034H
0100	EI	// Разрешение прерываний
0101	L2: JMP L2	// Бесконечный цикл. Ожидание запроса на прерывание.
0104	HLT	// Остановка программы.

Задания для выполнения.

1. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 02H и вывод этого символа в порт 03H. Окончанием ввода служит ввод символа **Space.** Программа обработки располагается по адресу 0008H. Запрос на обработку должен осуществляться специальной командой в коде программы.

2. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 05H и вывод этого символа в порт 06H. Окончанием ввода служит ввод символа **Tab.** Программа обработки располагается по адресу 0010H. Запрос на обработку должен осуществляться нажатием кнопки **INTR** и вводом номера прерывания.

3. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 01H и вывод этого символа в порт 03H. Окончанием ввода служит ввод символа **A.** Программа обработки располагается по адресу 003CH. Запрос на обработку должен осуществляться нажатием кнопки **RST7.5** и вводом номера прерывания.

4. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 01H и вывод этого символа в порт 03H. Окончанием ввода служит ввод символа **b.** Программа обработки располагается по адресу 0008H. Запрос на обработку должен осуществляться специальной командой в коде программы.

5. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 02H и вывод этого символа в порт 06H. Окончанием ввода служит ввод символа **C.** Программа обработки располагается по адресу 0010H. Запрос на обработку должен осуществляться нажатием кнопки **INTR** и вводом номера прерывания.

6. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 05H и вывод этого символа в порт 02H. Окончанием ввода служит ввод символа **d.** Программа обработки располагается по адресу 002CH. Запрос на обработку должен осуществляться нажатием кнопки **RST5.5** и вводом номера прерывания.

7. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 05H и вывод этого символа в порт 03H. Окончанием ввода служит ввод символа **e.** Программа обработки располагается по адресу 0008H. Запрос на обработку должен осуществляться специальной командой в коде программы.

8. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 01H и вывод этого символа в порт 06H. Окончанием ввода служит ввод символа **f.** Программа обработки располагается по адресу 0010H. Запрос на обработку должен осуществляться нажатием кнопки **INTR** и вводом номера прерывания.

9. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 03H и вывод этого символа в порт 06H. Окончанием ввода служит ввод символа **g.** Программа обработки располагается по адресу 0034H. Запрос на обработку должен осуществляться нажатием кнопки **RST6.5** и вводом номера прерывания.

10. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 02H и вывод этого символа в порт 05H. Окончанием ввода служит ввод символа **h.** Программа обработки располагается по адресу 0008H. Запрос на обработку должен осуществляться специальной командой в коде программы.

11. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 05H и вывод этого символа в порт 01H. Окончанием ввода служит ввод символа **i.** Программа обработки располагается по адресу 0010H. Запрос на обработку должен осуществляться нажатием кнопки **INTR** и вводом номера прерывания.

12. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 05H и вывод этого сим-

вола в порт 04H. Окончанием ввода служит ввод символа **j.** Программа обработки располагается по адресу 003CH. Запрос на обработку должен осуществляться нажатием кнопки **RST7.5** и вводом номера прерывания.

13. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 02H и вывод этого символа в порт 04H. Окончанием ввода служит ввод символа **k.** Программа обработки располагается по адресу 0008H. Запрос на обработку должен осуществляться специальной командой в коде программы.

14. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 02H и вывод этого символа в порт 01H. Окончанием ввода служит ввод символа **l.** Программа обработки располагается по адресу 0010H. Запрос на обработку должен осуществляться нажатием кнопки **INTR** и вводом номера прерывания.

15. Организовать обработку прерывания, где в качестве программы прерывания выступает прием символа кода ASCII из порта 03H и вывод этого символа в порт 02H. Окончанием ввода служит ввод символа **m.** Программа обработки располагается по адресу 0034H. Запрос на обработку должен осуществляться нажатием кнопки **RST6.5** и вводом номера прерывания.

Практическая работа № 7

ПРЯМАЯ АДРЕСАЦИЯ

Цель работы: Изучить прямую адресацию операндов

Основная функция любого процессора, ради которой он и создается, – это выполнение команд. Система команд, выполняемых процессором, представляет собой нечто подобное таблице истинности логических элементов или таблице режимов работы более сложных логических микросхем. То есть она определяет логику работу процессора и его реакцию на те или иные комбинации внешних событий.

Написание программ для микропроцессорной системы – важнейший и часто наиболее трудоемкий этап разработки такой системы. А для создания эффективных программ необходимо иметь хотя бы самое общее представление о системе команд используемого процессора. Самые компактные и быстрые программы и подпрограммы создаются на языке Ассемблер, использование которого без знания системы команд абсолютно невозможно, ведь язык Ассемблер представляет собой символьную запись цифровых кодов машинного языка, кодов команд процессора. Конечно, для разработки программного обеспечения существуют всевозможные программные средства. Пользоваться ими обычно можно и без знания системы команд процессора. Чаще всего применяются языки программирования высокого уровня, такие как Паскаль и Си. Однако знание системы команд и языка Ассемблер позволяет в несколько раз повысить эффективность некоторых наиболее важных частей программного обеспечения любой микропроцессорной системы – от микроконтроллера до персонального компьютера.

Каждая команда, выбираемая (читаемая) из памяти процессором, определяет алгоритм поведения процессора на ближайшие несколько тактов. Код команды говорит о том, какую операцию предстоит выполнить процессору и с какими ***операндами*** (то есть кодами данных), где взять исходную информацию для выполнения команды и куда поместить результат (если необходимо). Код команды может занимать от одного до нескольких байт, причем процессор узнает о том, сколько байт команды ему надо читать, из первого прочитанного им байта или слова. В процессоре код команды расшифровывается и преобразуется в набор микроопераций, выполняемых отдельными узлами процессора. Но разработчику микропроцессорных систем это знание не слишком важно, ему важен только результат выполнения той или иной команды.

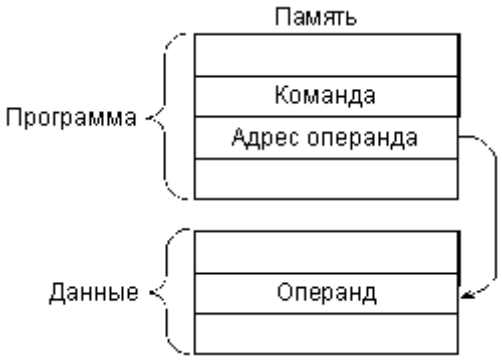
Адресация операндов

Большая часть команд процессора работает с кодами данных (операндами). Одни команды требуют входных операндов (одного или двух), другие выдают выходные операнды (чаще один операнд). Входные операнды называются еще операндами-источниками, а выходные называются операндами-приемниками. Все эти коды операндов (входные и выходные) должны где-то располагаться. Они могут находиться во внутренних регистрах процессора (наиболее удобный и быстрый вариант). Они могут располагаться в системной памяти (самый распространенный вариант). Наконец, они могут находиться в устройствах ввода/вывода (наиболее редкий случай). Определение места положения операндов производится кодом команды. Причем существуют разные методы, с помощью которых код команды может определить, откуда брать входной операнд и куда помещать выходной операнд. Эти методы называются **методами адресации**. Эффективность выбранных методов адресации во многом определяет эффективность работы всего процессора в целом.

Методы адресации

Количество методов адресации в различных процессорах может быть от 4 до 16. Рассмотрим несколько типичных методов адресации операндов, используемых сейчас в большинстве микропроцессоров.

Прямая (она же абсолютная) адресация (рис. 8.2) предполагает, что операнд (входной или выходной) находится в памяти по адресу, код которого находится внутри программы сразу же за кодом команды. Например, команда может состоять в том, чтобы очистить (сделать нулевым) содержимое ячейки памяти с адресом 1000000. Код этого адреса 1000000 будет располагаться в памяти, внутри программы в следующем адресе за кодом данной команды очистки.



Прямая адресация.

Практическая работа № 8

РЕГИСТРОВАЯ АДРЕСАЦИЯ

Цель работы: Изучить регистровую адресацию операндов.

Основная функция любого процессора, ради которой он и создается, – это выполнение команд. Система команд, выполняемых процессором, представляет собой нечто подобное таблице истинности логических элементов или таблице режимов работы более сложных логических микросхем. То есть она определяет логику работу процессора и его реакцию на те или иные комбинации внешних событий.

Написание программ для микропроцессорной системы – важнейший и часто наиболее трудоемкий этап разработки такой системы. А для создания эффективных программ необходимо иметь хотя бы самое общее представление о системе команд используемого процессора. Самые компактные и быстрые программы и подпрограммы создаются на языке Ассемблер, использование которого без знания системы команд абсолютно невозможно, ведь язык Ассемблер представляет собой символьную запись цифровых кодов машинного языка, кодов команд процессора. Конечно, для разработки программного обеспечения существуют всевозможные программные средства. Пользоваться ими обычно можно и без знания системы команд процессора. Чаще всего применяются языки программирования высокого уровня, такие как Паскаль и Си. Однако знание системы команд и языка Ассемблер позволяет в несколько раз повысить эффективность некоторых наиболее важных частей программного обеспечения любой микропроцессорной системы – от микроконтроллера до персонального компьютера.

Каждая команда, выбираемая (читаемая) из памяти процессором, определяет алгоритм поведения процессора на ближайшие несколько тактов. Код команды говорит о том, какую операцию предстоит выполнить процессору и с какими ***операндами*** (то есть кодами данных), где взять исходную информацию для выполнения команды и куда поместить результат (если необходимо). Код команды может занимать от одного до нескольких байт, причем процессор узнает о том, сколько байт команды ему надо читать, из первого прочитанного им байта или слова. В процессоре код команды расшифровывается и преобразуется в набор микроопераций, выполняемых отдельными узлами процессора. Но разработчику микропроцессорных систем это знание не слишком важно, ему важен только результат выполнения той или иной команды.

Адресация операндов

Большая часть команд процессора работает с кодами данных (операндами). Одни команды требуют входных операндов (одного или двух), другие выдают выходные операнды (чаще один операнд). Входные операнды называются еще операндами-источниками, а выходные называются операндами-приемниками. Все эти коды операндов (входные и выходные) должны где-то располагаться. Они могут находиться во внутренних регистрах процессора (наиболее удобный и быстрый вариант). Они могут располагаться в системной памяти (самый распространенный вариант). Наконец, они могут находиться в устройствах ввода/вывода (наиболее редкий случай). Определение места положения операндов производится кодом команды. Причем существуют разные методы, с помощью которых код команды может определить, откуда брать входной операнд и куда помещать выходной операнд. Эти методы называются **методами адресации**. Эффективность выбранных методов адресации во многом определяет эффективность работы всего процессора в целом.

Регистровая адресация (рис. 8.3) предполагает, что операнд (входной или выходной) находится во внутреннем регистре процессора. Например, команда может состоять в том, чтобы переслать число из нулевого регистра в первый. Номера обоих регистров (0 и 1) будут определяться кодом команды пересылки.

Регистровая адресация.

Практическая работа № 9

КОСВЕННАЯ АДРЕСАЦИЯ

Цель работы: Изучить укосвенную адресацию операндов.

Основная функция любого процессора, ради которой он и создается, – это выполнение команд. Система команд, выполняемых процессором, представляет собой нечто подобное таблице истинности логических элементов или таблице режимов работы более сложных логических микросхем. То есть она определяет логику работу процессора и его реакцию на те или иные комбинации внешних событий.

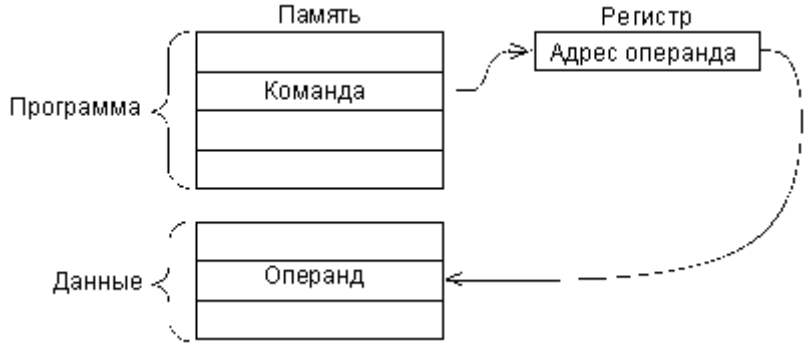
Написание программ для микропроцессорной системы – важнейший и часто наиболее трудоемкий этап разработки такой системы. А для создания эффективных программ необходимо иметь хотя бы самое общее представление о системе команд используемого процессора. Самые компактные и быстрые программы и подпрограммы создаются на языке Ассемблер, использование которого без знания системы команд абсолютно невозможно, ведь язык Ассемблер представляет собой символьную запись цифровых кодов машинного языка, кодов команд процессора. Конечно, для разработки программного обеспечения существуют всевозможные программные средства. Пользоваться ими обычно можно и без знания системы команд процессора. Чаще всего применяются языки программирования высокого уровня, такие как Паскаль и Си. Однако знание системы команд и языка Ассемблер позволяет в несколько раз повысить эффективность некоторых наиболее важных частей программного обеспечения любой микропроцессорной системы – от микроконтроллера до персонального компьютера.

Каждая команда, выбираемая (читаемая) из памяти процессором, определяет алгоритм поведения процессора на ближайшие несколько тактов. Код команды говорит о том, какую операцию предстоит выполнить процессору и с какими ***операндами*** (то есть кодами данных), где взять исходную информацию для выполнения команды и куда поместить результат (если необходимо). Код команды может занимать от одного до нескольких байт, причем процессор узнает о том, сколько байт команды ему надо читать, из первого прочитанного им байта или слова. В процессоре код команды расшифровывается и преобразуется в набор микроопераций, выполняемых отдельными узлами процессора. Но разработчику микропроцессорных систем это знание не слишком важно, ему важен только результат выполнения той или иной команды.

Адресация операндов

Большая часть команд процессора работает с кодами данных (операндами). Одни команды требуют входных операндов (одного или двух), другие выдают выходные операнды (чаще один операнд). Входные операнды называются еще операндами-источниками, а выходные называются операндами-приемниками. Все эти коды операндов (входные и выходные) должны где-то располагаться. Они могут находиться во внутренних регистрах процессора (наиболее удобный и быстрый вариант). Они могут располагаться в системной памяти (самый распространенный вариант). Наконец, они могут находиться в устройствах ввода/вывода (наиболее редкий случай). Определение места положения операндов производится кодом команды. Причем существуют разные методы, с помощью которых код команды может определить, откуда брать входной операнд и куда помещать выходной операнд. Эти методы называются **методами адресации**. Эффективность выбранных методов адресации во многом определяет эффективность работы всего процессора в целом.

Косвенно-регистровая (она же косвенная) адресация предполагает, что во внутреннем регистре процессора находится не сам операнд, а его адрес в памяти (рис. 8.4). Например, команда может состоять в том, чтобы очистить ячейку памяти с адресом, находящимся в нулевом регистре. Номер этого регистра (0) будет определяться кодом команды очистки.



Косвенная адресация.

Практическая работа № 10

СТЕК

Цель работы: Ознакомиться с понятиями стек и очередь.

Стек – это последовательный список переменной длины, включение и исключение элементов из которого выполняются только с одной стороны списка, называемого **вершиной** стека. Другие названия стека – «магазин» и очередь, функционирующая по принципу LIFO (Last – In – First – Out – "последним пришел – первым исключается"). Примеры стека: винтовочный патронный магазин, тупиковый железнодорожный разъезд для сортировки вагонов.

Основные операции над стеком – включение нового элемента (англ. push – заталкивать) и исключение элемента из стека (англ. pop – выскакивать).

Полезными могут быть также вспомогательные операции:

- определение текущего числа элементов в стеке;
- очистка стека;
- неразрушающее чтение элемента из вершины стека, которое может быть реализовано как комбинация основных операций:

x=pop(stack0); push(stack0,x).

Для наглядности рассмотрим небольшой пример, демонстрирующий принцип включения элементов в стек и исключения элементов из стека. На рис. 14.1 изображены следующие состояния стека:

- а) пустого;
- б–г) после последовательного включения в него элементов с именами 'A', 'B', 'C';
- д, е) после последовательного удаления из стека элементов 'C' и 'B';
- ж) после включения в стек элемента 'D'.

Рис. 14.1. Включение и исключение элементов из стека

Стек можно представить, например, в виде стопки книг (элементов), лежащей на столе. Присвоим каждой книге свое название, например A, B, C, D... Тогда в момент времени, когда на столе книги отсутствуют, про стек по аналогии можно сказать, что он пуст, т.е. не содержит ни одного элемента. Если же начинать последовательно класть книги одну на другую, то получим стопку книг (допустим из n книг), или получим стек, в котором содержится n элементов, причем вершиной его будет являться элемент n+1. Удаление элементов из стека осуществляется аналогичным образом, т. е. удаляется последовательно по одному элементу, начиная с вершины, или по одной книге из стопки.

Реализация стека может быть выполнена на основе массива. Кроме собственно массива необходимо дополнительно иметь переменную (назовем ее «указатель стека»), адресующую вершину стека. Под вершиной стека можно понимать либо первый свободный элемент стека, либо последний записанный. Все равно, какой из этих двух вариантов выбрать, важно впоследствии при обработке стека строго его придерживаться.

При занесении элемента в стек элемент записывается на место, определяемое указателем стека, затем этот указатель модифицируется таким образом, чтобы он указывал на следующий свободный элемент.

Операция выталкивания элемента состоит в модификации указателя стека (в направлении обратном модификации при включении) и выборке значения, на которое указывает указатель стека. После выборки элемент, в котором размещалось выбранное значение, считается свободным.

Библиографический список рекомендуемых источников

1. Гуров В.В. Архитектура микропроцессоров [Электронный ресурс] / В.В. Гуров. – Электрон. текстовые данные. – М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. – 115 с. – 978-5-9963-0267-3. – Режим доступа: http://www.iprbookshop.ru/56313.html

2. Электроника и микропроцессорная техника : учебник / В.Г. Гусев, Ю.М. Гусев. – Москва : КноРус, 2018. – 798 с. – Для бакалавров. – ISBN 978-5-406-06106-0. – Режим доступа: https://www.book.ru/book/926521

3. Вычислительные системы, сети и телекоммуникации (для бакалавров). Учебное пособие : учебное пособие / Л.П. Гудыно. – Москва : КноРус, 2019. – 372 с. – ISBN 978-5-406-06790-1. – Режим доступа: https://www.book.ru/book/930419