Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Оренбургский государственный университет»

Кафедра вычислительной техники

Р.Р. Галимов

АППАРАТНЫЕ СРЕДСТВА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Рекомендовано к изданию Редакционно-издательским советом
федерального государственного бюджетного образовательного учреждения
высшего профессионального образования
«Оренбургский государственный университет»
в качестве методических указаний для студентов, обучающихся по программам
высшего профессионального образования по направлению подготовки
090900.62 Информационная безопасность

Оренбург
2012

УДК 004.3(076)

ББК 32.973.26-04я7
Г 15

Рецензент - кандидат технических наук, доцент А.В. Хлуденев

Галимов Р.Р.

Г 15 Аппаратные средства вычислительной техники: методические указания / Р.Р. Галимов; Оренбургский гос. ун-т. - Оренбург: ОГУ, 2012. – 57 с.

Методические указания содержат 8 лабораторных работ. Каждая работа включает теоретическое изложение материала, постановку задачи, порядок выполнения и контрольные вопросы для самоподготовки.

Методические указания рекомендованы преподавателям как вспомогательный материал в организации и проведении занятий, а также студентам по профилю подготовки - «Комплексная защита объектов информатизации» - для аудиторного и самостоятельного освоения лабораторного курса дисциплины «Аппаратные средства вычислительной техники».

УДК 004.3(076)

ББК 32.973.26-04я7

Содержание

Введение

Настоящие методические указания предназначены для получения практических навыков студентами по профилю подготовки - «Комплексная защита объектов информатизации» при изучении дисциплины «Аппаратные средства вычислительной техники» (АСВТ).

Методические указания содержат 8 работ, рассчитанных на 16 часов аудиторных занятий. Предлагаемые задания охватывают основные разделы рабочей программы, связанные с изучением структуры и функционирования микропроцессора 8086.

Общие методические рекомендации по использованию лабораторных работ и методических указаний:

- к выполнению лабораторной работы следует приступать после ознакомления с теоретической частью соответствующего раздела и рекомендациями, приведенными в конкретной работе;

- лабораторные работы рекомендуется выполнять в порядке их нумерации;

- рекомендуется для экономии времени отчеты о лабораторных работах оформлять в виде протоколов работы с обязательным указанием номера, темы, цели работы и выводов с краткой характеристикой результата;

- дополнительные сведения по лабораторным работам содержатся в прилагаемом списке литературы.

Лабораторный курс может быть освоен на индивидуальном компьютере со средними техническими характеристиками, оснащенной ОС семейства Windows и эмулятором микропроцессора 8086 Emu8086.

Методические указания рекомендовано преподавателям как вспомогательный материал в организации и проведении занятий, а также студентам - для аудиторного и самостоятельного освоения лабораторной части дисциплины «Аппаратные средства вычислительной техники».

1 Лабораторная работа №1. Регистр флагов микропроцессора

Цель работы: изучить назначение регистра флагов микропроцессора 8086.

1.1 Теоретическая часть

Современные микропроцессоры являются сложными устройствами, реализованные на одном кристалле и содержащие большое количество функциональных блоков, таких как АЛУ, буферные регистры, регистры общего назначения, указатель стека, регистр результата (аккумулятор), регистр флагов (состояния).

Регистр флагов предназначен для хранения состояния и управления режимами работы микропроцессора. Регистр состояния МП 8086 содержит 16 бит, так называемых флажков, семь из которых зарезервированы на будущее (рисунок 1). *Флажки* микропроцессора 8086 разделяются на *условные,* отражающие результат предыдущей операции АЛУ, и *управляющие,* от которых зависит выполнение специальных функций.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Рисунок 1.1 - Регистр флагов МП 8086

К флажкам условий относятся ***SF***, ***ZF***, ***AF***, ***PF*** и ***CF***.

Флажок знака ***SF***. После арифметических и логических операций флаг знака принимает значение старшего (7 или 15) бита результата. Для знаковых двоичных чисел флаг знака принимает значение 0 при положительном результате и 1 при отрицательном (если только не возникло переполнение).

Флажок нуля ***ZF***. Устанавливается в 1 при получении нулевого результата и сбрасывается в 0, если результат отличается от нуля.

Флажок паритета ***PF***. Устанавливается в 1, если младшие 8 бит результата содержат четное число единиц; в противном случае он сбрасывается в 0. Флаг четности может использоваться для проверки правильности принятого кода при передаче

данных по линиям связи.

Флажок переноса ***CF***. При сложении (вычитании) устанавливается в 1, если возникает перенос (заем) из старшего бита.

Флажок вспомогательного переноса ***AF***. Устанавливается в 1, если при сложении (вычитании) возникает перенос (заем) из бита 3. Флажок предназначен только для двоично-десятичной арифметики.

Флажок переполнения ***OF***. Устанавливается в 1, если возникает переполнение (получение результата вне допустимого диапазона). При сложении этот флажок устанавливается, если имеется перенос в старший бит и нет переноса из старшего бита или наоборот. При вычитании он устанавливается, когда возникает заем из старшего бита, но заем в старший бит отсутствует, или наоборот.

К флажкам управления микропроцессора 8086 относятся: ***DF***, ***IF***, ***TF***.

Флажок направления ***DF***. Применяется в строковых командах. Если он сброшен, то строка обрабатывается с первого элемента, имеющего наименьший адрес, в противном случае - от наибольшего адреса к наименьшему.

Флажок разрешения прерываний ***IF***. Когда установлен этот флажок, МП распознает маскируемые прерывания, в противном случае - игнорирует.

Флажок прослеживания (трассировки) ***TF***. Когда этот флажок установлен, после выполнения каждой команды генерируется внутреннее прерывание.

К регистру состояния нельзя обратиться как к обычному регистру, для этого нужно использовать специальные команды. Флажки условий изменяются автоматически после выполнения арифметических операций и анализируются командами условного перехода.

Команда перехода нарушает естественный порядок выполнения программы посредством загрузки в программный счетчик ***IP*** адреса команды, к которой осуществляется переход. В командах условного перехода замена содержимого программного счетчика адресом зависят от значения флажков регистра состояний.

Команда условного перехода имеет следующий формат:

J_CODE D8

где ***CODE*** – код, определяющий условие перехода;

D8 – 8-битное знаковое смещение (в дополнительном коде) относительно следующей команды в программе.

В таблице 1.1 представлено описание команд условных переходов МП 8086.

Таблица 1.1 –Команды условных переходов

Код команды	Реальное условие	Условие для перехода (если)
JA ***JBE***	***CF*** = 0 и ***ZF*** = 0	Выше Не ниже и не равно
JAE ***JNB*** ***JNC***	***CF*** = 0	Выше или равно Не ниже Нет переноса
JB ***JNAE*** ***JC***	***CF***=1	Ниже Не выше и не равно Перенос
JBE ***JNA***	***CF*** = 1 или ***ZF*** = 1	Ниже или равно Если не выше
JE ***JZ***	***ZF*** = 1	Равно Ноль
JG ***JNLE***	***ZF*** = 0 и ***SF*** = ***OF***	Больше Не меньше и не равно
JGE ***JNL***	***SF*** = ***OF***	Больше или равно Не меньше
JL ***JNGE***	***SF*** <> ***OF***	Меньше Не больше и не равно
JLE ***JNG***	***ZF*** = 1 или (***SF xor OF***)=1	Меньше или равно Не больше
JNE ***JNZ***	***ZF*** = 0	Не равно Не ноль
JNO	***OF*** = 0	Нет переполнения
JO	***OF***=1	Есть переполнение
JNP ***JPO***	***PF*** = 0	Нет четности Нечетное
JP ***JPE***	***PF***=1	Есть четность Четное
JNS	***SF*** = 0	Нет знака
JS	***SF*** = 1	Есть знак

Слова «выше и ниже» в таблице относят к сравнению чисел без знака; слова «больше» и «меньше» учитывают знак.

При программировании в командах перехода обычно не используют явные

значение смещения ***D***8, а применяют метки. Метка – это идентификатор, присваиваемый первому байту команды, у которой она появляется. Наличие метки в команде не обязательно, но если она есть, то она становится символическим именем адреса данной команды.

Команды условного перехода используются для программирования выполнения различных действий в зависимости от заданного условия (условные операторы языков высокого уровня, например, оператору if .. then.. else фактически реализуются на командах условного перехода).

Пример кода, реализующего два варианта действий в зависимости от условия, приведено ниже:

```
        CMP   AL,100        ; устанавливаем флаги
        JA      NEXT        ; если AL>100, то переходим на адрес, указанный меткой next
        MOV   AH,2          ; иначе умножаем содержимое регистра AL на 2
        MUL   AH
        JMP   exit          ; переход на метку exit
next:   ADD   AL,5          ; прибавляем к содержимому AL число 5
exit:
        ----------------------  ;другие команды
        ----------------------
        ----------------------
```

Данный код выполняет следующие действия: если значение регистра ***AL***>100, то к нему прибавляется 5, иначе его значение умножается на 2 . Результат сохраняется в регистре ***AL***. В коде используются две метки: ***next*** и ***exit***.

Для сравнения чисел используется команда ***CMP***, которая осуществляет операцию вычитания. Результат никуда не записывается, но при этом устанавливаются флаги регистра состояния.

1.2 Постановка задачи

1. Изучить назначение регистра флагов микропроцессора (МП) 8086;

2. Используя эмулятор Emu8086, разработать программный код, позволяющий изменить определенные флаги регистра состояния (в соответствии с вариантом);

3. Сравнить команды сложения чисел ***ADD*** и ***ADC***.

4. Разработать программный код для МП 8086, осуществляющий сложение

содержимого регистров ***R***1 и ***R***2 при выполнении заданного условия ***COND***, иначе – их вычитание;

5. Составить отчет по пунктам 2-4;

6. Ответить на контрольные вопросы для самопроверки.

1.3 Порядок выполнения работы

1. Изучить теоретическую часть данной работы и получить представление о назначении регистра состояния и его флагах;

2. В эмуляторе Emu8086 создать новый проект, выбрав команду «New» на панели инструментов и шаблон «Bin template»;

3. В открывшемся окне после строки «add your code here» добавьте код на языке ассемблера, который будет устанавливать в «1» флаги, указанные в таблице 2;

4. Выполнить переход в режим эмуляции работы процессора, выбрав команду «Emulate» на панели инструментов. Отобразите на экране содержимое флагов регистра состояния, нажав на кнопку «flags» в окне «emulator:»;

5. Осуществить пошаговое выполнение разработанного кода, нажимая на кнопку «single step». Убедитесь, что в результате выполнения программы, заданные флаги устанавливаются в «1»;

6. Создайте проект, как в пункте 2, и наберите нижеследующий код:

```
mov     AL,2Dh
add     AL,0D3h
add     AL,5
```

Выполните программу в пошаговом режиме и запишите конечное значение регистра ***AL***.

7. Создайте проект, как в пункте 2, и наберите нижеследующий код:

```
mov     AL,2Dh
add     AL,0D3h
adc     AL,5
```

Выполните программу в пошаговом режиме и запишите конечное значение регистра ***AL***. Сравните полученные значения регистра ***AL*** в пунктах 6 и 7 и сделайте вывод.

8. Создайте новый проект, как в пункте 2, и разработайте код, осуществляющий сложение содержимого регистров ***R***1 и ***R***2 при выполнении условия ***COND***, иначе – вычитание. Обозначение регистров ***R***1 и ***R***2 и вид условия ***COND*** представлены в таблице 1.2. Осуществите проверку работы программы в пошаговом режиме;

9. Оформите отчет по работе. В отчете должно содержаться:

- цель работы;

- задание;

- листинги разработанных программных кодов;

- вывод по работе.

Таблица 1.2. –Варианты заданий

Номер варианта	Флаги	***R***1	***R***2	***R***3	***COND***
1	***CF,ZF***	***AX***	***BX***	***CX***	***R***3=1
2	***AF,OF***	***AL***	***AH***	***CH***	***R***3>12
3	***SF,PF***	***CL***	***DL***	***AL***	***R***3<230
4	***CF,OF***	***BL***	***AL***	***DH***	***R***3≥123
5	***ZF,SF***	***DH***	***BH***	***AL***	***R***3<>112
6	***AF,PF***	***DH***	***DL***	***CH***	***R***3≤21
7	***OF,PF***	***AH***	***CH***	***DL***	***R***3>15
8	***AF,ZF***	***AH***	***BH***	***BL***	***R***3<35
9	***SF,CF***	***AH***	***DH***	***DL***	***R***3=55
10	***SF,AF***	***AH***	***BL***	***DH***	***R***3≤124
11	***SF,ZF***	***CH***	***BH***	***DX***	***R***3≥300
12	***SF,PF***	***CL***	***BL***	***AX***	***R***3<>450
13	***AF,CF***	***CH***	***CL***	***BX***	***R***3=500
14	***AF,ZF***	***AH***	***BL***	***DH***	***R***3>22
15	***AF,SF***	***BH***	***CL***	***AL***	***R***3<11
16	***OF,CF***	***BL***	***DL***	***CX***	***R***3<>600
17	***OF,PF***	***DX***	***BX***	***AL***	***R***3≤144
18	***OF, ZF***	***AX***	***BX***	***DH***	***R***3≥33
19	***CF,ZF***	***BX***	***CX***	***AH***	***R***3>43
20	***CF,OF***	***CX***	***DX***	***AL***	***R***3<22

1.4 Контрольные вопросы

1. Назначение регистра флагов?

2. Какой флаг устанавливается в единицу, если результат последней операции равен нулю?

3. Какой флаг используется при двоично-десятичной арифметике?

4. В каком случае устанавливается флаг ***OF***?

5. Какой флаг используется для проверки правильности полученных данных по линиям связи?

6. После выполнения каких операций изменяются флаги регистра состояния?

7. В зависимости от чего осуществляется загрузка нового адреса в регистр ***IP*** в командах условного перехода?

8. Что представляет собой метка?

9. Чем отличается команда ***ADD*** от ***ADC***?

10. Какое арифметическое действие выполняет команда ***CMP***?

2 Лабораторная работа №2. Стек

Цель работы: изучить назначение стека микропроцессора 8086.

1.1 Теоретическая часть

Стек является удобной структурой данных для решения различных вычислительных задач, особенность которой в том, что она безадресная. Под термином «безадресная» подразумевается, что в команде не указывается — ни прямо, ни косвенно — адрес ячейки стека. В большинстве современных процессоров реализован аппаратный стек, который представляет из себя специально организованное оперативное запоминающее устройство. В МП 8086 под стек отводится область в ОЗУ и используется в основном для следующих целей:

- для хранения временных данных. Программист может разместить любые данные, не задумываясь, в какую ячейку памяти они будут размещены;

- для хранения адреса возврата из подпрограммы. Вызов подпрограммы из основной программы приводит к прыжку на другой адрес памяти. Чтобы осуществить возврат из подпрограммы в основную программу микропроцессор сохраняет в стек содержимое регистров ***CS*** и ***IP,*** которые определяют адрес текущей исполняемой команды. Подпрограмма должна заканчиваться специальной командой, которая восстанавливает из стека

начальные значения регистров ***CS*** и ***IP***. Подобная схема работает и при обработке прерываний;

- для доступа к регистру флагов;

- для изменения содержимого сегментных регистров;

-для передачи параметров между программами.

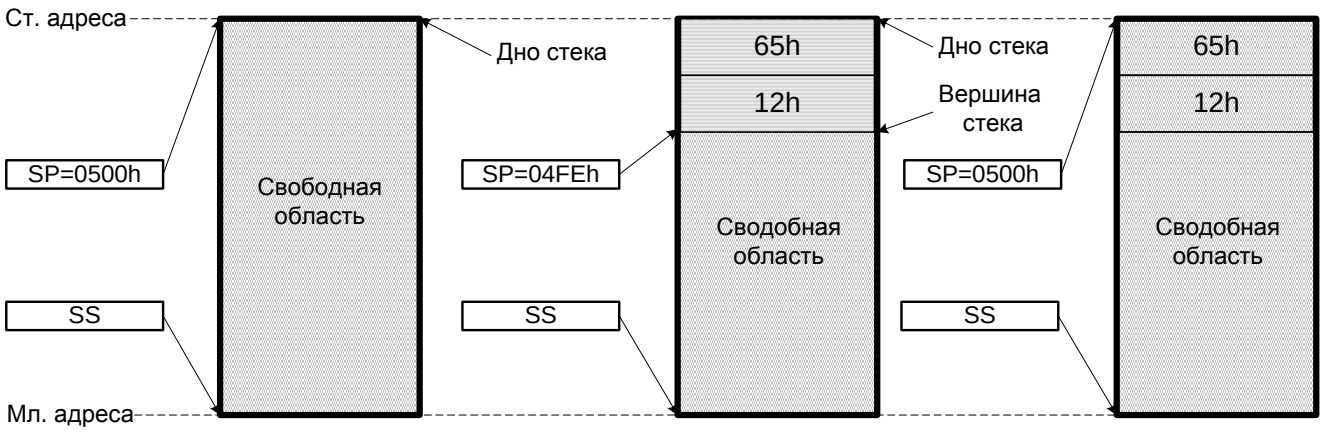
Стек относится к памяти типа ***LIFO*** (Last Input First Output, последним пришел - первым вышел), что означает, что последние загруженные данные будут выгружены в первую очередь. Здесь существует аналогия со стопкой тарелок: последнюю размещенную тарелку в стопке берем в первую очередь.

В МП 8086 каждый элемент стека занимает 2 байта, причем старший байт расположен в ОЗУ по старшему адресу, младший – по младшему. Микропроцессор для обращения к данным в стеке использует два регистра: ***SS*** и ***SP***. Сегментный регистр ***SS*** определяет начало блока памяти, отведенного под стек, а ***SP*** – смещение последний записи от начала сегмента. Стек растет «вниз», т.е. при записи в стек данных значение регистра-счетчика ***SP*** автоматически уменьшается на 2, а при чтении данных – увеличивается на 2. Для работы со стеком используются две основные команды: ***PUSH*** и ***POP***.

Команда **PUSH**. Помещает данные в стек. Формат команды: ***PUSH*** источник. Источником может быть регистр, сегментный регистр, непосредственный операнд или память. Фактически эта команда уменьшает ***SP*** на 2 и копирует содержимое источника в память по адресу ***SS***:***SP***.

Команда **POP**. Считывает данные из стека. Формат команды: ***POP*** приемник. Помещает в приемник слово, находящееся в вершине стека, увеличивая ***SP*** на 2. Приемником может быть регистр общего назначения, сегментный регистр, кроме ***CS***, переменная.

На рисунке 2.1 представлена схема стека при выполнении команды ***PUSH*** 6512h. Первоначально стек не содержит информации и вершина стека и его дно указывают на один и тот же адрес SS:0500h. После выполнения команды ***PUSH*** значение регистра ***SP*** уменьшается на 2 и указывает на свободный участок памяти.



а) б) в)

Рисунок 2.1 – Схема стека при выполнении команды ***PUSH*** и ***POP***

а) – до выполнения команд; б) – после выполнения команды ***PUSH***;

в) –после выполнения команды ***POP***

После выполнения команды ***POP*** данные из стека считываются и помещаются в приемник. При этом данные в ОЗУ не исчезают, просто вершина стека указывает на другие ячейки памяти.

Команды ***PUSF*** и ***POPF*** позволяют программисту получить доступ к регистру флагов, проанализировать его и при необходимости изменить. ***PUSHF*** помещает содержимое регистра флагов в стек, а ***POPF*** возвращает слово из стека в регистр флагов.

Одной из причин необходимости получения доступа к регистру флагов программой является её защита от выполнения под отладчиком. Отладчик – это программа, позволяющая исследовать внутреннее устройство программы. Отладчик обеспечивает пошаговое исполнение программы, просмотр текущих значений переменных, вычисление значения любого выражения программы и другие функции.

Отладчики могут быть использованы специалистами, занимающимися безопасностью вычислительных систем, в частности, для анализа функциональности вирусов. С другой стороны, данными программами пользуются хакеры для модификации программ с целью взлома защиты от нелицензионного использования программ.

Одним из способов реализации пошагового выполнения программы является установка флага ***TF*** регистра флагов. В этом случае процессор после выполнения любой команды генерирует прерывание ***int*** 1h.

Программа-отладчик вначале устанавливает флаг ***TF*** и запускает исследуемую

программу. После выполнения каждой команды исследуемой программы отладчик перехватывает прерывание 1h и может фиксировать текущие параметры вычислительной машины (состояние основных регистров процессора, участков памяти).

Для борьбы с данным механизмом отладки, программа должна при выполнении критически участков, например, проверки лицензионного ключа, выяснить, не установлен ли флаг ***TF***. Если установлен, то прекращается выполнение программы. При этом необходимо учесть, что отладчик может определить попытку проанализировать регистр флагов и снять его перед проверкой, а потом снова включить. Но в процессорах семейства 8086 до 386-го после операций перемещения с сегментными регистрами не вызывается исключительная ситуация даже при установленном флаге ***TF***. Для проверки значения флага ***TF*** можно использовать следующий код:

```
POP SS        ; операция с сегментным регистром, чтобы не вызывалось прерывание
              ; int1h в  случае установки флага TF
PUSHF         ; сохраняем содержимое регистра флагов в стек
POP AX        ; содержимое регистра флагов перемещаем в регистр AX
TEST AX,0100h; выполняем логическую операцию «И».
JNZ tracing  ; переход на блок кода, который обрабатывает факт трассировки
              ; программы, например, осуществляет  аварийное завершение программы
----------------; иначе продолжаем работу программы в штатном режиме
```

2.2 Постановка задачи

1. Изучить назначение стека микропроцессора 8086;
2. Используя эмулятор Emu8086 изучить команды ***PUSH*** и ***POP***;
3. Изменить программный код из лабораторной работы №1 пункта 4 постановки задачи таким образом, чтобы при выполнении условия выполнялось вычитание регистров ***R***1 и ***R***2, а иначе – их сложение за счет изменения флажка в регистре состояния с использованием стека;
4. Разработайте программный код, позволяющий поменять содержимое регистров ***R***3 и ***R***4, используя команды только для работы со стеком;
5. Составить отчет по работе;
6. Ответить на контрольные вопросы для самопроверки.

2.3 Порядок выполнения работы

1. Изучить теоретическую часть данной работы и получить представление о назначении стека микропроцессора 8086;

2. Изучить команды ***PUSH*** и ***POP***. Запустите эмулятор Emu8086 и создайте новый проект, выбрав в меню пункт «New». Выберите шаблон «Bin Template» и после слов «; add your code here», введите следующий код:

```
MOV AX,DI1
MOV BX,DI2
PUSH  AX
PUSH  DS
PUSH word [BX]
POP     ES
POP     DX
POP     CX
```

где ***DI***1 и ***DI***2 значения из таблицы 2.2.

3. Выберите пункт «Emulate» в меню, чтобы перейти в режим эмуляции. Установите значение 2 ячеек памяти ОЗУ (эффективный адрес ячеек определяется содержимым регистра ***BX***) равным ***DI***3. Для этого нажмите в нижней части окна эмуляции кнопку «Aux» и выберите пункт «memory». В результате на экране появится диалоговое окно «Random Access Memory» (рисунок 2.3), которое позволяет отобразить в режиме «table» 128 байт ОЗУ в виде таблицы 8x16. Начало блока памяти определяется двумя адресами: сегментом и смещением. В данном задании адрес сегмента оставляем без изменений, изменяем только смещение на значение ***DI***2 и нажимаем на кнопку «Update».

Необходимо учитывать, что число ***DI***3 двухбайтное, поэтому оно займет две соседние ячейки ОЗУ. Причем по меньшему адресу будет располагаться младший байт числа, а по старшему – старший байт. Для изменения значения ячейки памяти необходимо щелкнуть левой кнопкой (ЛК) мыши на выбранной ячейке и ввести новое значение в шестнадцатеричном коде.

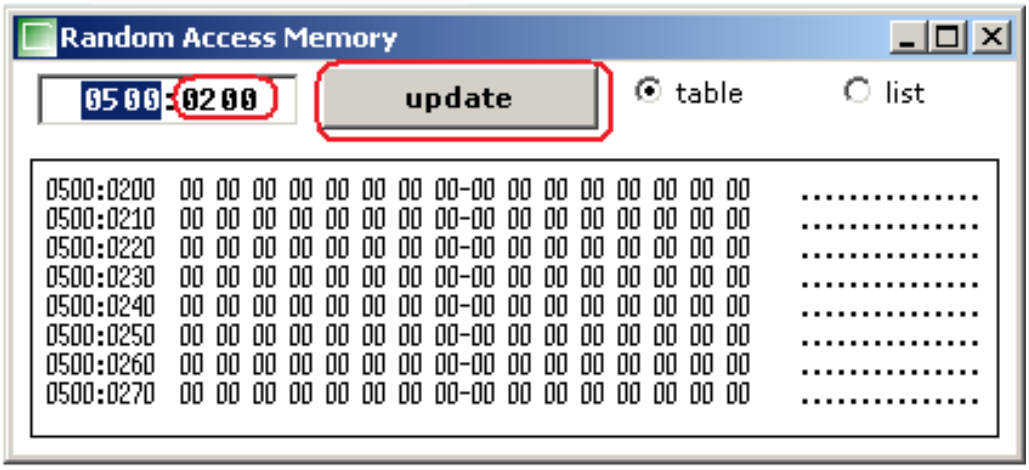


Рисунок 2.3 – Диалоговое окно для изменения содержимого ОЗУ в эмуляторе

4. Выполняем программу в пошаговом режиме, нажимая кнопку «Single Step». На каждом шаге необходимо фиксировать содержимое регистров ***SP***, ***AX***, ***DS***, ***ES***, ***BX***, ***DX***, ***CX***, ***SS*** в таблице следующего вида:

Таблица 2.1 – Изменения содержимого регистров

Номер шага	***SP***	***AX***	***DS***	***BX***	***ES***	***DX***	***CX***	***SS***
1								
2								
3								
4								
5								
6								

Содержимое стека можно просмотреть двумя способами:

- в окне эмуляции в нижней части экрана нажать на кнопку «stack»;

- просмотреть содержимое памяти по адресу ***SS***:***SP*** (как в пункте 3).

5. Сделать выводы по результатам выполнения 4 пункта работы.

6. Изменить программный код, реализующий задание 4 из лабораторной работы №1 таким образом, чтобы вычитание регистров ***R***1 и ***R***2 производилось при выполнении условия ***COND***, иначе – их сложение. При этом изменение работы программы должно реализоваться за счет инвертирования флажков регистра состояния при помощи стека, которые анализирует команда условного перехода.

Например, для инвертирования 15 флага регистра состояния при помощи стека можно использовать следующий алгоритм:

```
PUSHF
POP DX
XOR DX,8000h
PUSH DX
POPF
```

7. Выполните разработанный в пункте 6 программный код в эмуляторе Emu8086 в пошаговом режиме. При этом фиксируйте изменения в регистре флагов на каждом шаге. Сделайте выводы;

8. Создайте новый проект в Emu8086 и разработайте программный код, позволяющий обменять содержимое регистров ***R***3 и ***R***4, используя команды только для работы со стеком;

9. Выполните программный код, разработанный в пункте 8, в пошаговом режиме и убедитесь в его правильности;

10. Оформите отчет по работе. В отчете должно содержаться:

- цель работы;

- задание;

- листинги разработанных программных кодов;

- выводы по работе.

Таблица 2.2 –Варианты заданий

Номер задания	***DI***1	***DI***2	***DI***3	***AX***	***BX***	***R3***	***R***4	***D***1	***D***2	***D***3	***D***4	***D***5
1	234F	235E	236D	10	200	***SI***	***BX***	056A	156A	02B0	0C381	02B4
2	235E	236D	237C	11	201	***SI***	***CX***	056B	256B	02B1	0C181	02B0
3	236D	237C	238B	12	202	***BP***	***DX***	056C	356C	02B2	0C281	02B1
4	237C	238B	239A	13	203	***BP***	***DI***	056D	456D	02B3	0C781	02B2
5	238B	239A	23A9	14	204	***DI***	***SI***	056E	556E	02B4	0C681	02B3
6	239A	23A9	23B8	15	205	***BX***	***CX***	056F	656F	02B5	0C181	02B4
7	23A9	23B8	23C7	16	206	***BX***	***DX***	05A0	15A0	02B0	0C281	02B5
8	23B8	23C7	23D6	17	207	***BX***	***DI***	05A1	25A1	02B7	0C781	02B6
9	23C7	23D6	23E5	18	208	***BX***	***SI***	05A2	35A2	02B8	0C681	02B7
10	23D6	23E5	23F4	19	209	***CX***	***DX***	05A3	45A3	02B9	0C281	02B8
11	23E5	23F4	240A	20	210	***CX***	***DI***	05A4	55A4	02BA	0C781	02B9
12	23F4	240A	241B	21	211	***CX***	***SI***	05A5	15A5	02BB	0C681	02BA
13	240A	241B	242C	22	212	***DX***	***DI***	05A6	25A6	02BC	0C781	02BB
14	241B	242C	243D	23	213	***DX***	***SI***	05A7	35A7	02BD	0C681	02BC
15	242C	243D	0A123	24	214	***DX***	***BP***	05A8	45A8	02BE	0C581	02BD
16	243D	0A123	0B123	25	215	***BX***	***BP***	05A9	55A9	02BF	0C581	02BE

Продолжение таблицы 2.2

Номер задания	***DI***1	***DI***2	***DI***3	***AX***	***BX***	***R3***	***R***4	***D***1	***D***2	***D***3	***D***4	***D***5
17	0A123	0B123	0C2FD	26	216	***CX***	***BP***	05AA	15AA	02C0	0C581	02BF
18	0B123	0C2FD	0A2DF	27	217	***DX***	***BP***	05AB	25AB	02C1	0C581	02C0
19	0C2FD	0A2DF	34AD	28	218	***SI***	***BP***	05AC	35AC	02C2	0C581	02C1
20	0A2DF	34AD	49AA	29	219	***DI***	***BP***	05AD	45AD	02C3	0C581	02C2

2.4 Контрольные вопросы

1. Назначение стека микропроцессора 8086?
2. Какие регистры использует микропроцессор для определения вершины стека?
3. Где размещается стек?
4. Что означает аббревиатура ***LIFO***?
5. Как изменится содержимое регистра ***SP*** после выполнения команды ***PUSH***?
6. Сколько байт занимает одни элемент стека микропроцессора 8086?
7. Назначение программ-отладчиков?

3 Лабораторная работа №3. Подпрограммы и передача параметров через стек

Цель работы: изучить особенности выполнения вызова подпрограмм и передачи им параметров через стек

3.1 Теоретическая часть

В программировании часто встречаются ситуации, когда одинаковые действия необходимо выполнять многократно в разных частях программы. При этом с целью экономии памяти компьютера не следует многократно повторять одну и ту же последовательность команд в программе – достаточно один раз написать подпро-

грамму (в терминах языков высокого уровня - процедуру) и обеспечить правильный вызов этой подпрограммы и возврат в точку вызова по завершению подпрограммы.

Использование подпрограмм позволяет уменьшить время разработки программ и улучшает её структуру с точки зрения её понимания и сопровождения, что особенно важно в языках высокого уровня. Подпрограмма – это последовательность команд, заканчивающаяся командой возврата, выполнение которой может быть вызвано из любого места программы любое количество раз.

На рисунке 3.1 представлена общая схема размещения программы, использующей подпрограмму. Основная программа состоит из обычных команд и оператора вызова подпрограммы Call. Команда Call передает управление заданной подпрограмме, предварительно загрузив в стек адрес возврата (если программа и подпрограмм находятся в одном сегменте, то в стек заносится значение только регистра ***IP***, иначе - регистров ***CS*** и ***IP***). Дальше микропроцессор выполняет команды подпрограммы. Последней командой подпрограммы должна быть команда возврата RET, осуществляющую передачу управления в точку возврата.

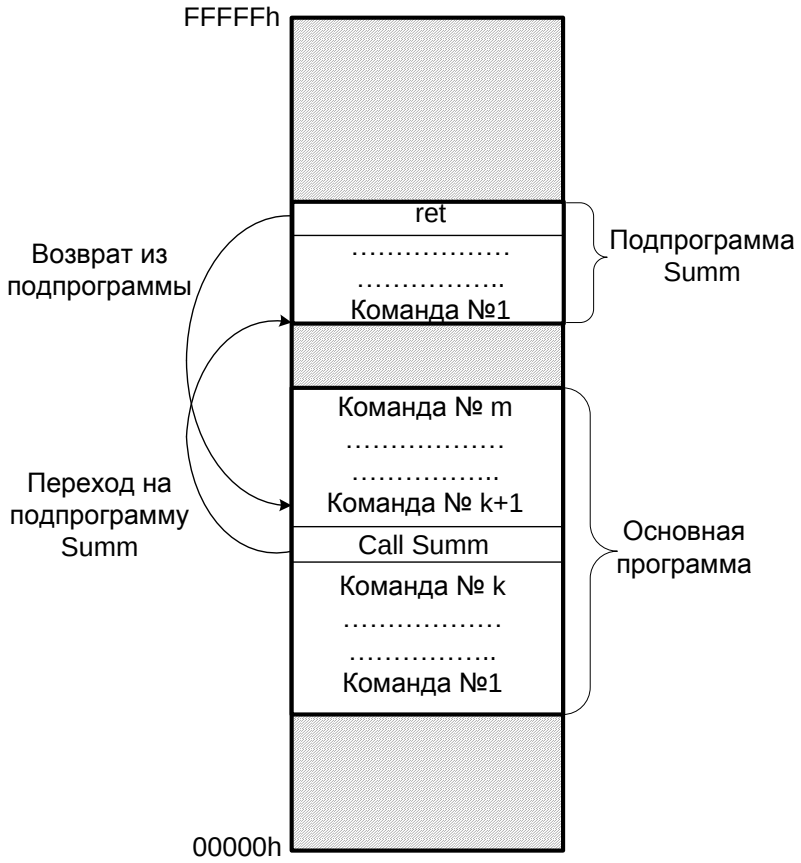
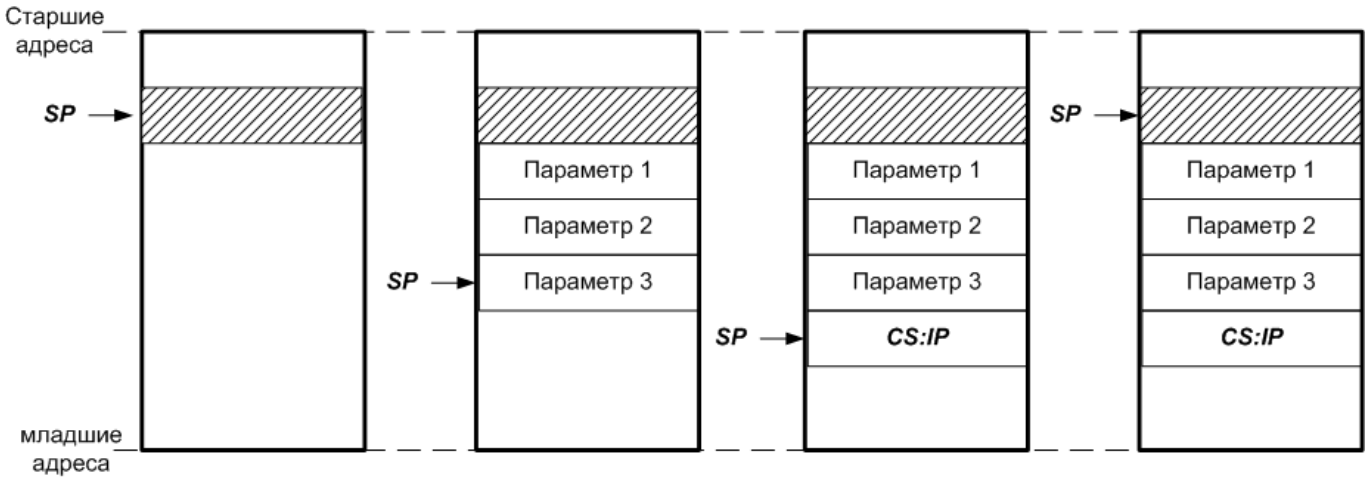


Рисунок 3.1 – Схема размещения в оперативной памяти программы

Часто основная программа должна передать вызываемой подпрограмме определенные данные, с которыми подпрограмма должна работать. Также подпрограмма после завершения работы должна передать результат своего выполнения в вызвавшую её программу. Существует три основных способа передачи параметров: через регистры, через общую область памяти и через стек. В большинстве языков высокого уровня, включая Паскаль и Си, передача параметров обычно осуществляется через стек. Передача параметров заключается в том, что вызывающая программа предварительно помещает в стек в определённом порядке параметры, требуемые для выполнения подпрограммы, а затем вызывает эту подпрограмму[5].

Пример передачи параметров через стек представлен на рисунке 3.2. Перед вызовом подпрограммы в стек размещаются 3 параметра (рисунок 3.2 б), а при выполнении команды Call добавляется адрес возврата из подпрограммы (рисунок 3.2 в).



а) б) в) г)

а — стек до вызова подпрограммы; *б* — в стек поместили три параметра; *в* — стек после выполнения команды call; *г* — стек после возврата из подпрограммы командой ret 6

Рисунок 3.2 – Передача параметров через стек

Команда возврата из подпрограммы ***RET*** считывает из стека адрес возврата, но при этом указатель стека ***SP*** указывал бы на параметр 3. Так как в дальнейшем эти параметры использоваться не будут, то это может привести к переполнению стека. Поэтом используется команда ***RET N***, которая считывает адрес возврата и прибавляет к регистру ***SP*** значение ***N***. На рисунке 3.2 г представлен случай, когда выполняется команда ***RET***

6, что позволяет указателю стека ***SP*** переместится в первоначальное состояние до передачи параметров и вызова подпрограммы.

На языке Ассемблера подпрограммы (процедуры) оформляются в следующем виде:

```
Name Proc
……; команды процедуры
……
Ret ;(RET N)
Name Endp
```

Рассмотрим пример реализации и использования в программе функции, рассчитывающей среднее значение 3 однобайтных целых чисел без знака:

```
; add your code here
push 150            ; передача параметра 1
push 150            ; передача параметра 2
push 30             ; передача параметра 3
call myFunction     ; вызов подпрограммы MyFunction
HLT         ; halt! ; остановка работы процессора
myFunction proc     ;подпрограмма MyFunction
   push bp          ; сохраняем предыдущее значение
   mov bp,sp        ; запоминаем в регистре BP адрес последнего элемента стека
   xor ah,ah        ;обнуляем старший байт регистра AX (AH)
   mov al,[bp+4]    ; записываем в регистр AL значение  параметра 3
   add al,[bp+6]    ; складываем содержимое регистров AL и параметра2
                    ; и сохраняем результат в AL
   jnc l1           ; если есть переполнение регистра AL,
   inc ah           ; то увеличиваем регистр AH на 1
l1:                 ;метка
   add al,[bp+8]    ; прибавляем к регистру AL содержимое параметра 1
   jnc l2:          ; если есть переполнение регистра AL,
   inc ah           ; то увеличиваем регистр AH на 1
                    ; в итоге в регистре AX будет содержать сумму 3 параметров
l2:
   mov bl,3         ; помещаем в регистр BL значение 3
   div bl           ; делим содержимое регистра AX на BL
                    ; в регистре AL – будет целая часть результата операции
                    ;деления, в AH –остаток.
   pop bp           ; восстанавливаем содержимое регистра BP
   ret 6            ; вытаскиваем из стека адрес возврата и устанавливаем
                    ;SP=SP+6
myFunction endp
```

Три входных параметра передаются функции MyFunction через стек. Для обращения к параметрам внутри функции используется регистр ***BP***. Так как данный регистр мог использоваться в основной программе, то его первоначальное значение нужно сохранить.

Поэтому первой командой подпрограммы является сохранение значения регистра ***BP*** в стек и предпоследней – восстановление первоначального значения. В результате, в стеке оказывается 5 двухбайтных элементов: 3 входных параметра, адрес возврата и значение регистра ***BP***.

Следующей командой процедуры в регистр ***BP*** заносим значение указателя стека ***SP***, который содержит адрес последнего элемента стека. Зная адрес последнего элемента и порядок загрузки параметров можно обращаться к нужному параметру, используя регистровую относительную адресацию, по следующей формуле:

$$A=[BP+Sm],\ Sm=4+(n-i)\cdot 2, \qquad (3.1)$$

где ***A*** –адрес параметра;

BP – регистр микропроцессора, хранящий адрес последнего элемента стека;

Sm – смещение параметра в байтах от вершины стека;

n - количество параметров;

i – порядковый номер параметра.

Параметр команды ***RET N*** определяется формулой:

$$N=n\cdot 2. \qquad (3.2)$$

Последней командой процедуры восстанавливаем адрес возврата и устанавливаем значение ***SP*** в первоначальное состояние.

3.2 Постановка задачи

1. Разработать процедуру, вычисляющую значение линейной функции:

$$y=(a+b)/2 \qquad (3.3)$$

где ***y*** - результат, который сохраняется в регистр ***AL***;

a,b - целые однобайтные числа без знака , которые передаются в качестве параметров процедуре;

2. Разработать программу, использующую процедуру из пункта 1 задания, и проверить её в пошаговом режиме в эмуляторе Emu8086;

3. Сделать выводы и составить отчет по работе.

3.3 Порядок выполнения работы

1. Создайте новый проект в среде эмулятора Emu8086, используя шаблон «Bin template»;

2. Разработайте процедуру на языке ассемблера согласно заданию 1 из постановки задачи. Значения параметров ***a*** и ***b*** возьмите из таблицы 3.3 согласно своему варианту.

3. Разработайте программу, использующую процедуру, реализованную во втором пункте. За основу возьмите пример кода, представленного в теоретической части данной работы;

4. Запустите проект на эмуляцию, нажав кнопку меню «Emulate». Нажмите в окне «emulator» кнопку «stack» для отображения содержимого стека;

5. Начните выполнение программы в пошаговом режиме, нажав кнопку «Single step». Перед выполнением команды «Call MyFunction», зафиксируйте значения регистров ***IP*** и ***SP*** соответственно как IP_1 и SP_1 в таблицу 3.1. Вычислите значение смещения до следующей команды (в примере до команды «hlt») по формуле:

$$A_K = IP + 3. \qquad (3.4)$$

Запишите в отчет значение A_K.

Таблица 3.1 – Значения регистров МП и адресов операндов

IP_1	IP_2	IP_3	SP_1	SP_2	SP_3	BP	A_K	A_a	A_b

6. После выполнения команды «mov bp,sp» в процедуре запишите в отчет содержимое стека в таблицу 3.2 и значение регистра ***BP*** в таблицу 3.1.

Таблица 3.2 – Содержимое стека

Смещение	Значение
FFFE	0000
FFFC	FF96
FFFA	FF96
FFF8	001E
FFF6	0009
FFF4	0000

7. По формуле 3.1 рассчитайте адреса A_a и A_b параметров ***a*** и ***b***.

8. Продолжайте выполнять программу в пошаговом режиме. Запишите значение регистра ***IP*** и ***SP*** соответственно как IP_1 и SP_1 в таблицу 3.1 перед выполнением команды «***RET N***» и после - как IP_2 и SP_2.

9. Выполните программу до конца. Убедитесь в корректности программы.

10. Проанализируйте значения таблиц 3.1 и 3.2 и сделайте выводы по работе.

11. Составьте отчет по работе. Отчет должен включать:

- цель работы;
- задания к лабораторной работе;
- листинг программы;
- таблицы 3.1 и 3.2 с результатами, полученными при выполнении работы;
- выводы по работе.

Таблица 3.3 – Варианты заданий

Номер варианта	1	2	3	4	5	6	7	8	9	10
a	15	16	10	120	50	110	39	33	25	89
b	20	30	60	100	39	51	69	79	32	63
Номер варианта	11	12	13	14	15	16	17	18	19	20
a	22	29	68	75	81	36	73	92	74	22
b	167	75	93	134	88	101	99	49	38	150

3.4 Контрольные вопросы

1. Какими преимуществами обладает разработка программ с использованием подпрограмм?

2. Какими способами можно передавать параметры подпрограмме?

3. Какие действия выполняет микропроцессор при выполнении команды «Call»?

4. Что делает команда «RET 3»?

5. Как получить доступ внутри процедуры к параметрам, переданным из основной программы через стек?

6. Где сохраняется адрес возврата из процедуры?

4 Лабораторная работа №4. Команды ввода-вывода микропроцессора

Цель работы: изучить команды ввода-вывода микропроцессора 8086

4.1 Теоретическая часть

Возможности вычислительной техники во многом определяются составом и техническими возможностями внешних устройств. Существует большое количество разнообразных внешних устройств (ВУ), таких как жесткие диски, принтеры, сканеры, клавиатуры и другие. Внешние устройства отличаются по физическим принципам работы, по скорости передачи данных, по используемым протоколам. Важной особенностью внешних устройств является то, что они работают в более медленном темпе, чем возможности вычислительного ядра. В связи с этим возникает задача синхронизации между собой внешнего устройства и процессора.

Процессоры не могут учитывать особенности работы каждого устройства из-за их большого количества и разнообразия. В связи с этим, в систему вводятся специальные устройства - адаптеры, которые являются своеобразными переводчиком с языка микропроцессора на язык ВУ.

Взаимодействие МП с внешними устройствами сводится к записи или чтению данных в специальные регистры, ассоциированных с ВУ. Данные регистры называют портами. Соответственно, управление ВУ, например, принтером, заключается в записи команд и чтении состояния внешнего устройства из соответствующих портов. При этом МП не знает, какие управляющие сигналы и в какой последовательности нужно подавать на исполнительные устройства ВУ. Например, чтобы распечатать в нужной позиции символ, процессор записывает в порты только общие команды: в какой позиции и код символ. Аналогично, сигналы от принтера, например, об отсутствии бумаги или картриджа, поступают в виде числового кода в соответствующие порты, откуда считываются МП.

Для работы с портами используются две команды: ***IN*** и ***OUT***. Команда IN имеет следующий вид:

***IN AL*(*AX*), *Addr*(*DX*),** (4.1)

где ***AL*** или ***AX*** – приемник данных от порта внешнего устройства. Если данные 8-разрядные, то используется регистр ***AL***, если 16-разрядные – то ***AX*;**

Addr - адрес порта. Адрес порта можно задать непосредственно числом (в этом случае адрес может быть числом от 0 до 255) или содержимым регистра ***DX*** (адрес может принимать значения от 0 до 65535).

Вывод данных на порт внешнего устройства осуществляется командой ***OUT***, имеющий следующий формат:

***OUT Addr*(*DX*), *AL*(*AX*).** (4.2)

При выполнении команды «IN» данные с порта внешнего устройства передаются в регистр ***AL*** или ***AX*** микропроцессора. Ниже приведен пример использования команд ввода-вывода:

```
        mov dx,199   ; помещаем в регистр DX адрес порта дисплея
        xor ah,ah            ;выполняем операцию «исключающее ИЛИ» для
                             ;обнуления старшего байта регистра AX
l1:     in al,110            ; считываем значение порта внешнего устройства
                             ;  с адресом 110 в регистр        AL
        out dx,ax            ; выводим в порт с адресом 199  значение регистра AX
        cmp al,20            ; сравниваем значение регистра AL с 20
        jnz l1          ; если не равно, то продолжаем цикл
```

Вышеприведенный программный код в цикле осуществляет чтение одного байта из порта с адресом 110 и отправляет 2 байта на порт 199. В среде эмулятора Emu8086 порту 110 сопоставлено виртуальное устройство Simple. Данное устройство является моделью однобайтного порта. Порту 199 сопоставлен дисплей (Led_display), который отображает двухбайтное число.

4.2 Постановка задачи

1. Разработать программный код в эмуляторе Emu8086, поддерживающий на виртуальном термометре «Thermometer» заданное значение температуры ***T*** при исход-

ном значении температуры окружающей среды ***T***1. Текущее значение температуры можно считать с порта 125, а включить спиртовку, подогревающую термометр, – подав на порт 127 значение 1.

4.3 Порядок выполнения работы

1. Создайте новый проект в среде эмулятора Emu8086, используя шаблон «Bin template»;

2. Разработайте программный код на языке ассемблера согласно заданию 1 из постановки задачи. Значения температур ***T*** и ***T***1 возьмите из таблицы 4.1 согласно варианту;

3. Запустите проект на эмуляцию, нажав кнопку меню «Emulate». В окне эмулятора нажмите на пункт главного меню «virtual devices» и выберите «Thermometer.exe»;

4. Выполните программу в пошаговом или автоматическом режиме, убедитесь, что результаты работы программы в эмуляторе соответствуют постановке задачи;

5. Оформите отчет по работе. В отчете должно содержаться:

- цель работы;
- задание;
- листинги разработанного программного кода;
- выводы по работе.

Таблица 4.1 –Варианты заданий

Номер варианта	1	2	3	4	5	6	7	8	9	10
T	60	61	62	63	64	65	66	67	68	69
$\boldsymbol{T}_1$	25	26	27	28	29	30	31	32	33	34
Номер варианта	11	12	13	14	15	16	17	18	19	20
T	70	71	72	73	74	75	76	77	78	79
$\boldsymbol{T}_1$	35	36	37	38	39	40	41	42	43	44

4.4 Контрольные вопросы

1. Дайте понятие порт.

2. Если порт внешнего устройства является двухбайтным, то куда помещается результат его чтения командой «***IN***»?

3. Что делает команда «***OUT***»?

4. Какой максимальный адрес порта можно указать в команде «IN», если адрес порта задается непосредственно числом?

5. В каком случае в качестве источника данных используется регистр ***AX*** в команда ввода-вывода?

5 Лабораторная работа №5. Адаптер внешнего устройства

Цель работы: изучить принципы работы адаптера внешнего устройства.

5.1. Теоретическая справка

Взаимодействие непосредственно только через порты при помощи команд ввода-вывода реализуется для простых внешних устройств. При этом быстродействие ВУ должна быть сопоставима с производительностью процессора. Иначе, это может привести к потере или дублированию данных. Поэтому для организации взаимодействия внешнего устройства и вычислительного ядра используются адаптеры (контроллеры), позволяющие синхронизировать процесс приема-передачи[3].

На рисунке 5.1 представлена общая схема организации обмена данными между МП и внешним устройством. Любой адаптер содержит по крайне мере порт (регистр) данных ***Pd***. В режиме ввода ВУ записывает данные в порт ***Pd***, где они хранятся до момента пересылки по системной шине в МП. В режиме вывода микропроцессор записывает данные в порт ***Pd***, которые хранятся там до вывода во ВУ адаптером. Кроме того, большинство адаптеров содержат регистр состояния, отражающий параметры работоспособности внешнего устройства. Один из разрядов порта со-

стояния ***Ps*** обычно используется как флаг готовности ***Ready***. Данный флаг устанавливается внешним устройством при готовности осуществить обмен данными с вычислительным ядром. Перед началом цикла ввода-вывода МП процесс должен проверить флаг ***Ready***. При значении флага равного «1» микропроцессор осуществляет обмен данными с внешним устройством, иначе - переходит в режим ожидания.

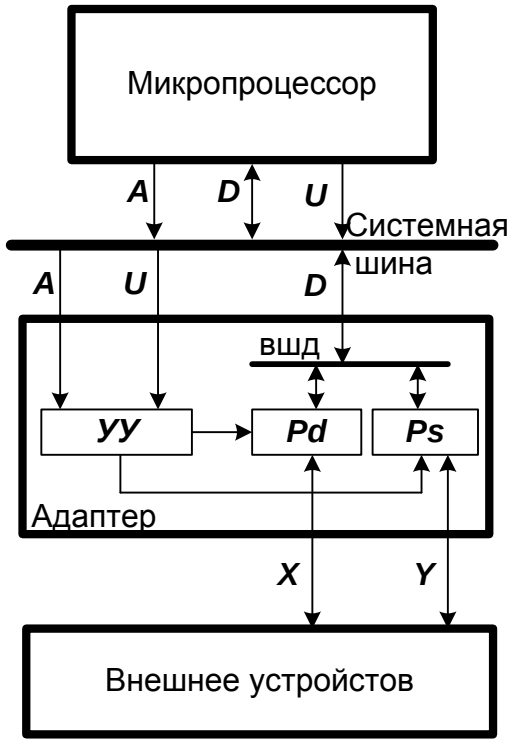


Рисунок 5.1 – Общая схема организации обмена данными МП
с внешним устройством

Каждый порт адаптера имеет уникальный адрес. МП при записи выдает на системную шину (СШ) адрес порта ***A***, затем данные ***D*** и выставляет сигнал записи ***IOWR*** на шину управления ***U***.

Так как работа ВУ часто не привязана к тактовой частоте процессора, то возможна ситуация, когда МП записывает новую порцию данных в порт, а внешнее устройство еще не прочитало предыдущую. В результате данные будут потеряны. Чтобы это предотвратить применяют механизм квитирования, заключающийся в выработке специальных сигналов стробирования ***Stb*** и подтверждения ***Ask***. Сигнал ***Stb*** формируется ведущим устройством при записи данных в порт ***Pd***, чтобы сообщить ведомому, что появились данные. Ведомое устройство подтверждает получе-

ние данных сигналом ***Ask***. Эти сигналы могут быть привязаны к определенным разрядам регистра состояния адаптера.

5.2 Постановка задачи

1. Написать программный код в эмуляторе Emu8086, отображающий в цикле данные с внешнего устройства (ВУ) «Simple» на виртуальный дисплей «Led display». ВУ подключено к адаптеру, который содержит два 8-разрядных порта: порт данных ***Pd*** и порт статуса ***Ps***. Внешнее устройство записывает данные в ***Pd*** только при условии, что бит №0 порта статуса равен «0». Запись данных внешним устройством в порт ***Pd*** приводит к установке нулевого бита порта ***Ps*** в единицу. Дисплей отображает значение, которое содержится в порте 199 (2 байта). Выход из цикла осуществляется при подаче ВУ на порт ***Pd*** значения ***D***;

5.3 Порядок выполнения работы

1. Создайте новый проект в среде эмулятора Emu8086, используя шаблон «Bin template»;
2. Разработайте программный код на языке ассемблера, позволяющий отображать на виртуальном дисплее «Led_display» данные, полученные от ВУ. Прежде чем считать данные с порта ***Pd***, необходимо проверить, есть ли в буфере адаптера данные. Наличие данных определяется по нулевому разряду порта ***Ps***. Адреса портов ***Pd*** и ***Ps*** определяются из таблицы 5.1 согласно варианту задания;
3. Запустите режим эмуляции работы процессора. Нажмите на кнопку «virtual devices» в окне эмулятора и выберите «Led_display» для отображения дисплея.
4. Запустите программу «Simple». В открывшемся окне нажмите кнопку «Настройки» и укажите соответствующие адреса портов ***Pd*** и ***Ps*** из таблицы 5.1;
5. Протестировать разработанный код в различных режимах передачи данных ВУ. Программа предусматривает 2 режима передачи данных МП:

- передача одиночных данных. Для этого необходимо в текстовое поле ввести передаваемые данные и нажать на кнопку «Записать». Состояние флага наличия данных в буфере ***Ps***0 отображается флажком «Порт пуст»;

- циклическая передача данных. Передаваемое значение увеличивается (уменьшается) на единицу со значением интервала времени, указанного в настройках программы;

6. Оформите отчет по работе. В отчете должно содержаться:

- цель работы;

- задание;

- листинги разработанных программных кодов;

- выводы по работе.

Таблица 5.1 –Варианты заданий

Номер варианта	1	2	3	4	5	6	7	8	9	10
Pd	145	146	147	148	149	100	101	102	103	104
Ps	150	151	152	153	154	155	160	120	121	122
Номер варианта	11	12	13	14	15	16	17	18	19	20
Pd	148	123	124	125	126	127	130	131	132	133
Ps	160	161	162	163	164	165	166	167	168	169

5.4 Контрольные вопросы

1. Назначение адаптера внешнего устройства?
2. Каково назначение регистра состояния адаптера?
3. Какие основные порты включает в себя адаптера?
4. В чем заключается механизм квитирования?
5. Что сообщает сигнал ***ASK*** адпатера?

6 Лабораторная работа №6. Контроль передачи данных

Цель работы: изучить механизм контроля по четности/ нечетности при передаче данных между внешним устройством и МП

6.1 Теоретическая часть

Передача данных между вычислительным ядром и внешним устройством осуществляется по линиям связи, на которые могут воздействовать помехи. Это может привести к искажению значения передаваемых данных. Одним из простых способов определения искажения данных является механизм контроля по четности/нечетности.

При контроле по четности передаваемые данные представляют собой структуру , в которой кроме информационных бит присутствует и бит четности ***PF***. На рисунке 6.1 представлена структура пакета, в котором для передачи данных используются 7 бит.

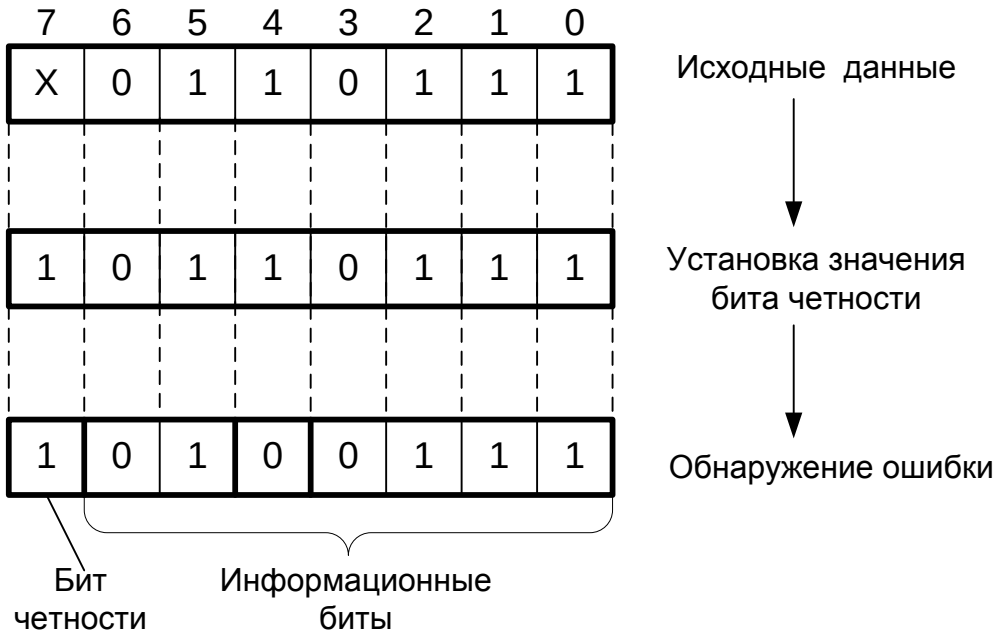


Рисунок 6.1 –Контроль по четности

Так как количество «1» в информационных разрядах нечетно, то бит четности устанавливается в единицу. При передаче происходит искажение 4-го бита. Получатель обнаруживает искажение пакета, так как количество единиц во всем пакете нечетно.

Контроль по нечетности, аналогичен контролю по четности, только здесь в бит нечетности помещают единицу, чтобы их количество было нечетно. При этом

приемник фиксирует ошибку, если количество единиц в пакете четно.

На рисунке 6.2 представлена схема организации обмена данными с внешним устройством, которая моделируется в лабораторной работе.

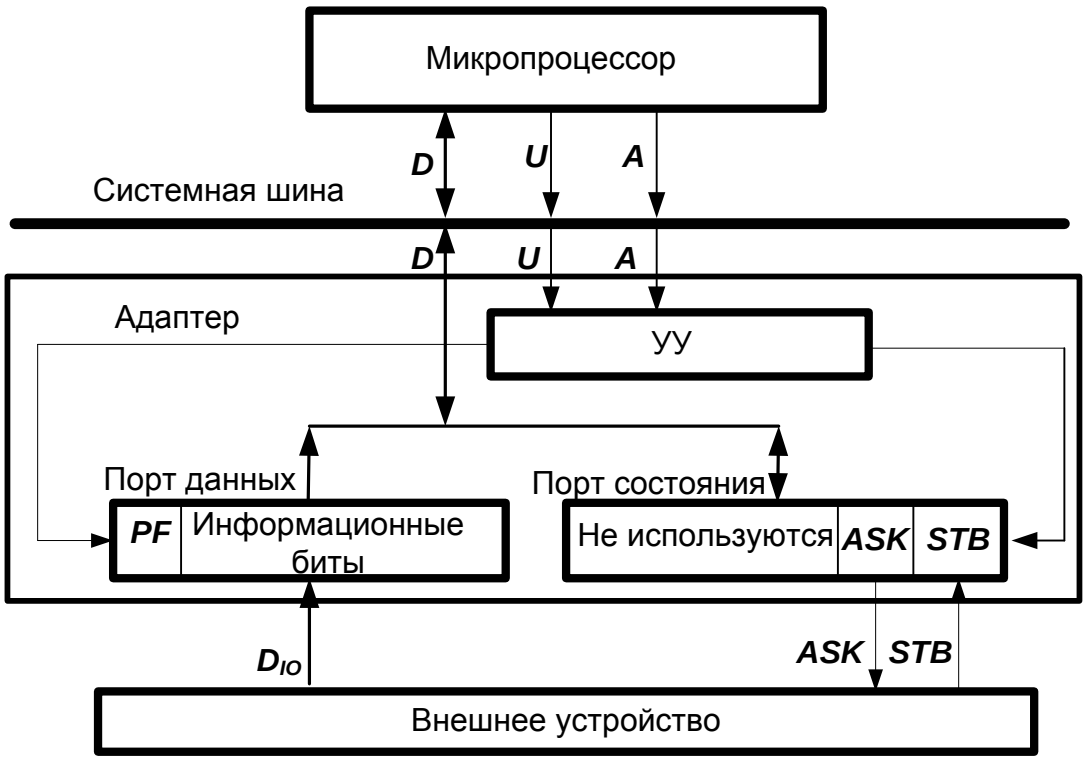


Рисунок 6.2 – Схема организации обмена данными между МП и ВУ с использованием контроля по четности

Внешнее устройство формирует пакет с семью информационными битами и битом четности/нечетности, который передает в порт данных адаптера. Передача данных сопровождается сигналом стробирования ***STB*** от ВУ, который фиксируется в нулевом разряде порта состояния. Микропроцессор, в свою очередь, в цикле опрашивает порт состояния, ожидая появление «1» в поле ***STB***. Как только МП обнаруживает появление данных, считывает пакет из порта данных и подсчитывает количество «1». Если количество единиц в пакете четно, то МП устанавливается бит ***ASK*** в «1» , иначе в ноль. Одновременно сбрасывается бит ***STB***, тем самым сообщая ВУ о получении данных. При искажении данных внешнее устройство повторно передает пакет данных.

Ниже приведен программный код, который определяет четность количества единиц в регистре ***AL***:

```
OR AL,AL     ; логическая операция для установки битов регистра флагов МП
```

```
        JP  paritet     ;переход на метку paritet, если количество «1»  в младшем байте
                        ;результата четно
        ……..            ;команды, выполняющиеся при нечетном количестве «1»
        jmp next        ; переходим на метку next, чтобы не выполнять команды из блока
                        ; paritet
paritet:
        ……..            ;команды, выполняющиеся при нечетном количестве «1»
```

Для контроля по четности/нечетности используется флаг ***PF*** регистра состояния МП, который устанавливается в «1», если количество 1 в младшем байте результата арифметической или логической операции четно. Поэтому первой командой является логическая операция «ИЛИ», которая не изменяет результата, а только устанавливает флаги.

6.2 Постановка задачи

Разработать программный код в эмуляторе Emu8086, отображающий в цикле данные с внешнего устройства (ВУ) «Paritety» на виртуальный дисплей «Led display». ВУ подключено к адаптеру, содержащему два 8-разрядных порта: порт данных ***Pd*** и порт статуса ***Ps***. Старший разряд ***Pd***7 используется для организации контроля по четности или нечетности. Разряд №0 порта статуса ***PS*** устанавливается в «1», когда ВУ записывает данные в порт данных. МП при чтении данных с адаптера должен сбрасывать бит ***Ps***0. Записав 1 в разряд №1 порта ***Ps***, МП подтверждает правильность полученного байта сигналом ***ASK*** (проверка корректности полученных определяется по контролю четности/нечетности), иначе внешнее устройство должно повторить передачу данных. При выводе полученного байта на дисплей старший разряд ***Pd***7 не должен учитываться.

6.3 Порядок выполнения работы

1. Создайте новый проект в среде эмулятора Emu8086, используя шаблон «Bin template»;

2. Разработайте программный код на языке ассемблера, отображающий данные от ВУ на виртуальном дисплее с контролем по четности (нечетности). Программа перед чтением должна удостовериться, что ВУ записало данные в порт данных (бит ***Ps***0 установлен в единицу), и проверить их на возможные искажения. Об получении корректного пакета данных МП сообщает внешнему устройству установкой бита ***Ps***1 порта статуса в единицу, иначе внешнее устройство повторяет передачу текущего байта;

3. Запустите режим эмуляции работы процессора. Нажмите на кнопку «virtual devices» в окне эмулятора и выберите «Led_display» для отображения дисплея.

4. Запустите программу «Paritety». В открывшемся окне нажмите кнопку «Настройки» и укажите адреса портов ***Pd*** и ***Ps*** из таблицы 6.1 согласно своим вариантам;

5. Запустите программный код на выполнение в эмуляторе Emu8086. В программе «Paritety» выставьте вид контроля согласно варианту из таблицы 6.1, уберите галочку «Ошибка в бите», введите в текстовое поле число от 0 до 127 и нажмите кнопку записать. Проверьте корректность разработанного программного кода на языке ассемблера;

6. Повторите пункт 14 с моделирование одиночной ошибки при передаче данных. Для этого установите галочку «Ошибка в бите». Номер бита, в котором произойдет ошибка, выберите из таблицы 6.1 согласно варианту;

7. Запустите передачу данных от ВУ «Paritety» в автоматическом режиме. Для этого выберите режим изменения передаваемых данных внешним устройством (инкремент/декремент) из таблицы 6.1 и нажмите кнопку старт. Проверьте корректность разработанного программного кода на языке ассемблера;

8. Оформите отчет по работе. В отчете должно содержаться:

- цель работы;
- задание;
- листинг разработанной программы;
- выводы по работе.

Таблица 6.1 –Варианты заданий

Номер варианта	1	2	3	4	5	6	7	8	9	10
Pd	145	146	147	148	149	100	101	102	103	104
Ps	150	151	152	153	154	155	160	120	121	122
Код контроля	0	1	0	1	0	1	0	1	0	1
Код режима передачи данных	1	0	1	0	1	0	1	0	1	0
Номер варианта	11	12	13	14	15	16	17	18	19	20
Pd	148	123	124	125	126	127	130	131	132	133
Ps	160	161	162	163	164	165	166	167	168	169
Код контроля	0	1	0	1	0	1	0	1	0	1
Код режима передачи данных	1	0	1	0	1	0	1	0	1	0

Таблица 6.2 – Коды задания

Код	Вид контроля	Режим передачи данных
1	По четности	Инкремент
2	По нечетности	Декремент

6.4 Контрольные вопросы

1. Как реализуется контроль по четности?
2. Оцените избыточность в процентах при использовании контроля по четности для пакета размеров 8 бит.
3. Определяет ли контроль по нечетности ошибку в двух разрядах?
4. Какой флаг регистра состояния микропроцессора используется для определения четного количества единиц в байте?

7 Лабораторная работа №7. Машинный формат команд микропроцессора

Цель работы: изучить машинный формат команд микропроцессора 8086 с помощью отладчика ***Debug***.

7.1. Теоретическая справка

По функциональному признаку система команд микропроцессора 8086 разби-

вается на 6 групп: пересылка данных, арифметические операции, логические операции и сдвиги, передача управления, обработка цепочек и управления микропроцессором. Каждой команде соответствует определенный двоичный код, который определяет тип команды и адреса операндов.

Способ представления команды в виде чисел в двоичном коде называется машинным форматом команды. Микропроцессор понимает команды, представленные только в машинном формате. Так как вся цифровая информация, обрабатываемая МП, в том числе и команды, хранятся в элементах памяти в виде набора логических «1» и «0». Но человеку запомнить соответствие последовательности до нескольких десятков единиц и нулей (например, до 48) определенной команде очень сложно. Поэтому были разработаны языки программирования (Ассемблер, Pascal, Ci++), использующие символьные коды для представления команд. Для перевода команд в символьном представлении в машинный формат используются специальные программы: компиляторы и трансляторы.

Машинные форматы команд МП 8086 в зависимости от типа команды и способа адресации содержат от 1 до 6 байтов, общая структура которого представлена на рисунке 7.1.

Ключевыми являются первые два байта: ***COP*** и ***RegAdr***. Первый байт в основном определяет код операции, а второй – способ адресации операндов. Следующие 2 байта команды используются как адрес одного из операндов, причем адрес может быть как 8-битным, так и 16-битным.

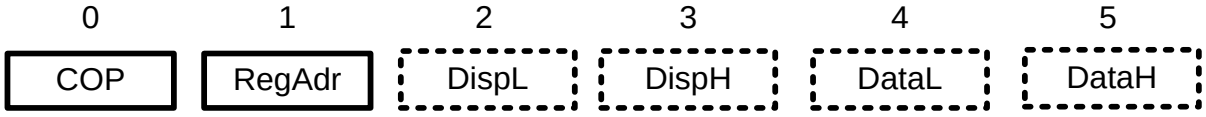


Рисунок 7.1 – Структура машинного формата команды МП 8086

Последние два байта используются, если в команде непосредственно указывается один из операндов виде числа.

В данной работе изучение машинного формата команды производится на примере команды ***MOV***. Команда ***MOV*** предназначена для пересылки данных от источника (***src***) к приемнику (***dst***). Источником могут быть регистр, ячейка памяти и

или непосредственный, а в качестве приемника – регистр или ячейка памяти. Команда ***MOV*** состоит из кода операции (***COP***, первый байт), способа адресации (второй байт) и, если того требует ***COP***, адреса ячейки памяти (***dispL*** и ***dispH***), непосредственно операнда (***dataL*** и ***dataH***).

По ***COP*** различают 2 вида команды пересылки:

- в качестве источника и приемника используются регистры или ячейки памяти (рисунок 7.2 а));

- в роли источника выступают непосредственные данные, а в качестве приемника – либо регистр, либо память (рисунок 4.2 б).

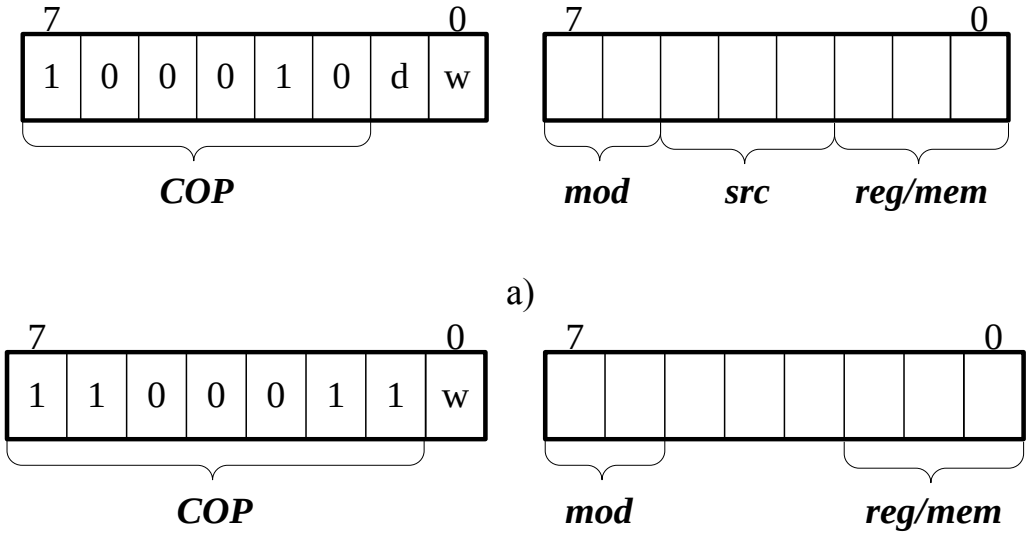


Рисунок 7.2 – Первые 2 байты команды ***MOV***

Поле ***COP*** определяет тип команды, а поле ***src*** - адрес регистра источника. Адреса регистров представлены в таблице 7.1.

В МП 8086 есть ряд регистров имеющих одинаковые адреса регистров. Например, регистр ***DI*** имеет адрес 111_2, и регистр ***BH*** имеет тот же адрес. Единственная разница в том, что ***DI*** —16-разрядный регистр, а ***BH*** - 8-разрядный. Значит, в ***COP*** должен быть предусмотрел отличительный признак байта или слова. Это разряд ***W*** (word — слово). Для него возможны следующие случаи:

- 0 - значит, что все адреса касаются однобайтовых данных;

- 1 — значит, что все адреса имеют отношение к словам.

Для кодирования адреса приемника отведено 5 разрядов: ***mod*** и ***reg***/***mem***. Следовательно, существует 32 (2^5) различных способов адресации. Основой адресации приемника является поле ***mod*** (***D***7,***D***6), который определяет два варианта: приемник - это регистр микропроцессора (***reg***), или элемент ЗУ (***mem-MEMORY***).

Приемником является регистр если ***mod***=11. Поле ***reg***/***mem*** в этом случае определяет адрес регистра-приемника данных в соответствии с таблицей 4.1.

Второй случай - приемником является элемент ЗУ (ОЗУ или ПЗУ). Здесь возможны 24 варианта (2^5-8). Их можно объединить в 7 глобальных групп, которые определяются полем ***mod***: 00, 01, 10.В таблице 7.2 представлены способы формирования эффективного адреса ***EA*** соответствующие данным вариантам.

Таблица 7.1 – Адреса регистров

Регистр	Адрес (двоичный код)	Регистр	Адрес (двоичный код)
AX	000	DL	010
AH	100	BX	011
AL	000	BH	111
CX	001	BL	011
CH	101	DI	111
CL	001	SI	110
DX	010	BP	101
DH	110	SP	100

Таблица 7.2 –Способы адреса приемника данных

Reg/***mem***	***mod***(***D***7***D***0)		
$D_2D_1D_0$	00	01	10
000	***EA***=<***BX***>+<***SI***>	***EA***=<***BX***>+<***SI***>+***D***1	***EA***=<***BX***>+<***SI***>+ ***D***2
001	***EA***=<***BX***>+<***DI***>	***EA***=<***BX***>+<***DI***>+ ***D***1	***EA***=<***BX***>+<***DI***>+ ***D***2
010	***EA***=<***BP***>+<***SI***>	***EA***=<***BP***>+<***SI***>+ ***D***1	***EA***=<***BP***>+<***SI***>+ ***D***2
011	***EA***=<***BP***>+<***DI***>	***EA***=<***BP***>+<***DI***>+ ***D***1	***EA***=<***BP***>+<***DI***>+ ***D***2
100	***EA***=<***SI***>	***EA***=<***SI***>+ ***D***1	***EA***=<***SI***>+ ***D***2
101	***EA***=<***DI***>	***EA***=<***DI***>+ ***D***1	***EA***=<***DI***>+ ***D***2
110	Прямая адресация, ***EA***=адрес(2 байта)	***EA***=<***BP***>+ ***D***1	***EA***=<***BP***>+ ***D***2
111	***EA***=<***BX***>	***EA***=<***BX***>+ ***D***1	***EA***=<***BX***>+ ***D***2

Непосредственная адресация: адрес «запрятан» в коде операции. Очевидный пример: ***mod***=11, значит приемнкик - это регистр, номер которого указывается в ***COP***.

Прямая адресация. Следом за ***COP*** должен следовать адрес ячейки памяти (сначала младший байт, затем - старший). Таким образом, в машинном коде сразу виден адрес приемника. В таблице 4.2: ***mod***=00, ***reg***/***mem***=110.

Косвенная адресация. В коде операции указывается источник (хранилище) адреса ячейки памяти. Пример: ***mod***=00, ***reg***/***mem***=100, 101, 111. Во всех этих случаях адрес приемника автоматически будет извлекаться из регистра ***SI***, ***DI*** или ***BX***. Прежде чем обратиться к ячейке памяти, адрес этой ячейки надо загрузить в соответствующий регистр.

Относительная адресация. В этом случае вместо прямого адреса следом за кодом операции располагают смещение (***disp***). Это смещение (в таблице 4.2 однобайтное смещение обозначено как ***D***1, а двухбайтное - как ***D***2) складывается с текущим адресом, который может храниться в ***BP***, ***BX***, ***SI***, ***DI***. Причем, это смещение со знаком (старший разряд выступает в качестве знакового бита), поскольку смещение может быть как вперед, так и назад, и в дополнительном коде.

Все эти четыре способа адресации являются основными, далее возможны комбинации из этих способов со смещениями, что и следует из таблицы 7.2.

Также конечный адрес в самом сложном случае может формироваться как сумма двух регистров и смещения (например, ***mod***=01, ***mod***=10, и ***reg***/***mem***=000…011).

В команде ***MOV***, если приемником может быть и регистр МП, и ячейка ЗУ, то источником — только регистр. Чтобы сделать источником ячейку памяти в ***COP*** отводится один разряд ***D***9 для изменения направления пересылки (***d*** - ***down*** - вниз). Если ***d***=0, то направление не меняется: ***dst*** остается приемником, а ***src*** — источником. Если же ***d***=l, то направление переворачивается в обратную сторону, то есть ***src*** — становится приемником, а ***dst*** —источником.

Рассмотрим пример по синтезу машинной команды, которая осуществляет перемещение содержимого регистра ***BL*** в регистр ***AL*** (кратко: <***AL***>:=<***BL***>):

- ***w***=0(1 байт) - пересылается информация размером в 1 байт;

- ***d***=0 (направление не меняется);

- ***src***=011 (адрес регистра ***BL*** — из таблицы 7.1);

- ***dst***=l 1 000 (***mod*** =11— из таблицы 7.2; ***reg***/***mem***=***reg***=000 — адрес регистра ***AL*** —из таблицы 4.1).

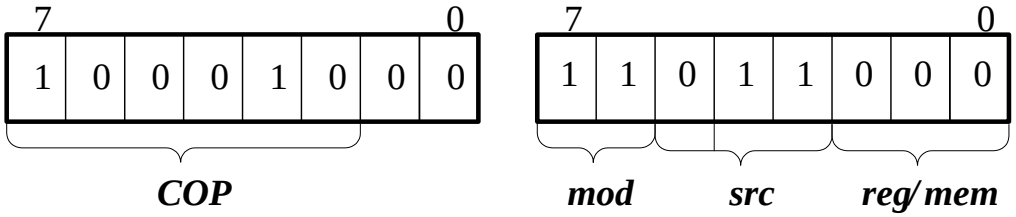


Рисунок 7.3 – Машинный формат команды ***MOV AL,BL***

Разбиваем команду в машинном формате на тетрады и переводим в шестнадцатеричный код. Получается, что вся команда состоит из двух байт: 88 D8.

7.2 Задание к выполнению лабораторной работы

1. Переслать содержимое регистра ***R***1 в регистр ***R***2.

Кратко: <***R***2>:=< ***R***1>.

Ассемблер: MOV ***R***2, ***R***1;

2. Отправить в ячейку памяти с адресом ***A***1 содержимое регистра ***R***3.

Кратко: M_{A1}: =<***R***3> .

Ассемблер: MOV [***A***1],***R***3.

3. Отправить в ячейку памяти с адресом ***A***2 значение ***D***1.

Кратко: M_{A2}: =***D***1.

Ассемблер:MOV BYTE [***A***2],***D***1;

4. Отправить в 2 ячейки памяти по адресу ***A***3 два байта ***D***2.

Кратко: M_{A3}: =***D***2

Ассемблер: MOV WORD [***A***3],***D***2.

Таблица 7.3 –Варианты заданий

Номер варианта	***R***1	***R***2	***A***1	***R***3	***A***2	***D***1	***A***3	***D***2
1	***DX***	***AX***	23CE	***AH***	12BD	1D	BC21	A12C
2	***BX***	***CX***	10FF	***DL***	3D12	2A	7AD2	B231
3	***AL***	***DH***	3FD1	***CX***	89A1	2C	20C1	1FDA
4	***BL***	***CH***	71DD	***BX***	34DA	48	2398	1233
5	***CX***	***AX***	3DFF	***BH***	FF12	1F	44CA	3DA1
6	***DH***	***CL***	F123	***DX***	A23C	3F	23DE	12AA
7	***SI***	***AX***	F1F2	***DI***	3FF1	E2	E123	D145
8	***BP***	***CX***	4ED1	***SI***	C123	91	616D	C2D1
9	***SP***	***BX***	3FF1	***DI***	DD12	67	2A12	34F1
10	***DX***	***SP***	34AF	***BP***	45DF	AD	2367	3D1A
11	***AL***	***AH***	0A121	***BX***	22DF	2B	0C431	45AA
12	***DH***	***CL***	12DE	***BH***	8A01	0FF	32CE	4AAA
13	***DI***	***CX***	0E561	***SI***	71AF	77	1AFE	391F
14	***CX***	***SI***	2367	***BH***	9A5D	EE	34A1	0AA33
15	***DL***	***CL***	0B399	***AX***	8A9B	DF	56FD	7171
16	***BL***	***CH***	925D	***AH***	7227	D1	4671	8DAE
17	***SI***	***BX***	7DF4	***BX***	8D40	28	9667	5523
18	***BL***	***DL***	449D	***DH***	3912	9A	0F717	6912
19	***CL***	***BH***	1198	***BL***	0F211	81	0DDD1	6743
20	***CX***	***SI***	7AF5	***DH***	8812	11	6CB1	8ADF

7.3 Порядок выполнения работы

1 Согласно варианту задания сформировать машинный код.

2. Запустите программу DEBUG.exe. Для ОС Windows XP и Vista выполните следующие действия:

- в меню «Пуск» выберите команду «Выполнить»;

- наберите в командной строке «cmd» для запуска командного интерпретатора;

- наберите команду ***debug*** в командном интерпретаторе, для запуска отладчика.

3.Занесите в память по смещению 100 машинный код при помощи следующей команды:

- ***e*** 100 <***COP***>,

где <***COP***> - это машинный код команды в шестнадцатеричной системе счисления;

100 - относительный адрес.

Пример команды: ***-e*** 100 88 ***D***8. Байты команды необходимо отделять друг от друга пробелами.

4. Значения большинства регистров в среде ***debug*** при первоначальном запуске равны нулю. Чтобы увидеть перемещение данных из одного регистра в другой командой ***-r*** устанавливаем значение регистров. Команда ***–r*** без параметров показывает содержимое всех регистров (в том числе и регистра флагов). Команда ***–r*** <***reg***> показывает содержимое регистра «***reg***» и позволяет установить новое значение в шестнадцатеричном коде (новое значение вводится после двоеточия). Причем обращение к 8- разрядным регистрам, например, к ***AL*** и ***BH*** напрямую невозможно. Можно изменить только 16-разрядные регистры, такие как ***AX*** и ***BX***.

5. Осуществите пошаговое выполнение команды в основной памяти по смещению 100 при помощи команды ***-t*** =100. Не пропустите знак «=», иначе отладчик выполнит не одну команду, которая размещена по адресу 100, а 100 команд, начиная с текущего адреса.

6. Просмотрите содержимое регистров при помощи команды ***r***. Убедитесь, что закодированная в машинный формат команда, выполняет действие в пункте 1.

7. Выполните пункты 1-5 для второго задания из раздела 7.2. Для просмотра содержимого памяти по определенному адресу выполните команду:

-d <***addr***>,

где ***d*** –команда отладчика debug для просмотра содержимого участка памяти;

addr- адрес начала участка памяти. Можно указывать как полный адрес, указав сегмент и смещение, или указывать только смещение от начала сегмента. В данной работе достаточно использовать только смещение, так как все сегменты сопоставлены на один и тот же блок памяти.

При выполнении данной команды на экран выдается 8 строк по 16 байт. Убедитесь, что команда перемещает содержимое регистра ***R***3 в ячейки ОЗУ с адресом ***A***1.

6. Выполните аналогичные действия для заданий 3 и 4 раздела 7.2. Для занесения исходных данных в память, которые будут перемещаться при выполнении команды в машинном формате, используйте команду ***–e*** в отладчике.

7. Оформите отчет по работе. В отчете должно содержаться:

- цель работы;
- задание;
- машинные форматы команд для каждого пункта задания;
- экранные формы отладчика ***debug*** с результатами после выполнения команд;
- выводы по работе.

7.4 Контрольные вопросы

1. Чем отличается представление команд в машинном формате от символьного представления?
2. Сколько байт максимально может содержать команда в машинном формате МП 8086? Сколько минимально?
3. Каково назначение основных полей команды ***MOV*** МП8086?
4. За что отвечает поле ***d*** в команде ***MOV***?
5. Какое будет значение поля ***w*** («0» или «1»)в машинном формате команды перемещения, если в качестве источника используется регистр ***AL***?
6. Какие функции выполняет поле ***mod***?
7. Какая комбинация полей ***mod*** и ***reg***/***mem*** определяет косвенную адресацию?
8. Что делает команда -***e*** в среде ***debug***?
9. Как просмотреть содержимое памяти по смещению 500 в среде ***debug***?

8 Лабораторная работа № 8. Способы адресации МП

Цель работы: изучить основные виды способов адресации МП 8086

8.1 Теоретическая часть

8.1.1 Способы адресации

Для выполнения какой-либо операции в команде должно содержаться указание вида операции, а также, откуда берутся в операции операнды и куда помещается результат (т. е указание на источник и приемник операндов).

Под способом адресации называют способы указания источников и приемников операндов.

Типичные режимы адресации микропроцессора 8086 разделяются на два класса — режимы адресации данных и режимы адресации переходов. Первая группа предназначена для определения размещения обрабатываемых данных, вторая группа – адреса следующей выполняемой команды. В данной работе будут рассмотрены режимы адресации данных.

Способ адресации данных МП 8086 определяется обычно во втором байте команды в машинном формате (рисунок 7.1) и возможные варианты представлены в предыдущей лабораторной работе в таблице 7.2. В данной работе будут рассмотрены способы адресации с точки зрения особенности использования их при разработке программ.

Рассмотрим основные способы адресации данных.

Непосредственный. Операнд длиной 8 или 16 бит является частью команды. Пример на языке ассемблера:

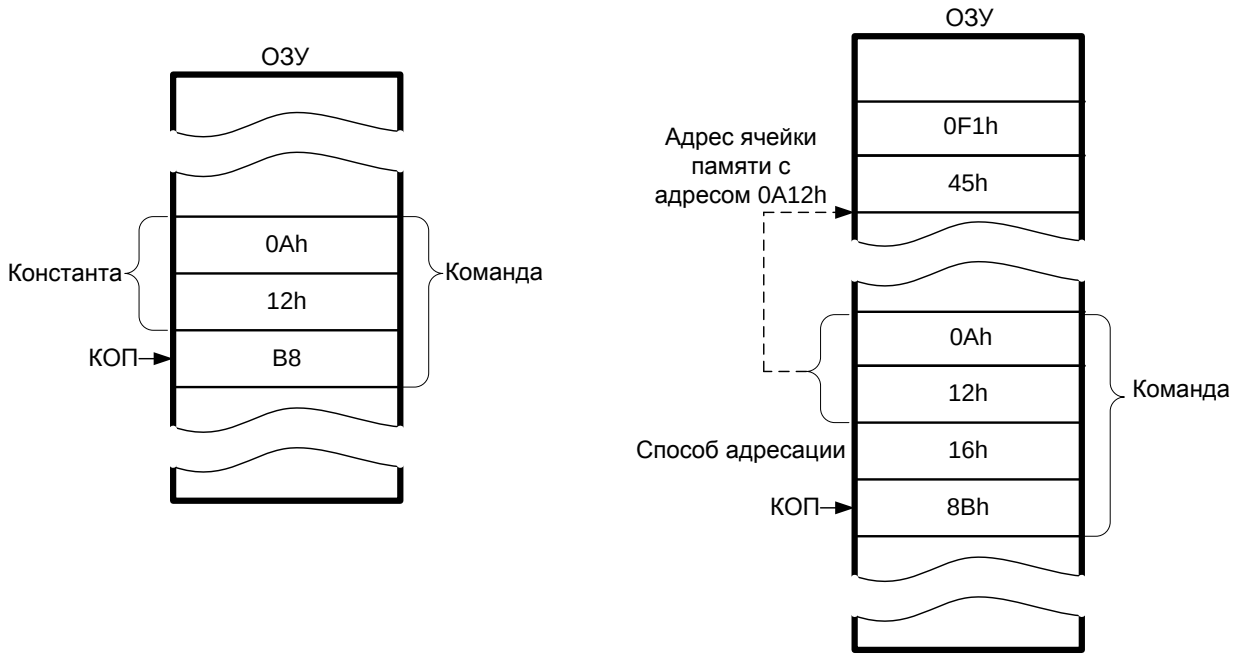
```
MOV AX,0A12h.
```

Как видно из рисунка 8.1 а) команда занимает 3 байта: код операции, определяющий перемещение константы в регистр ***AX***, и два байта - непосредственно константа.

Прямой. 16-битный эффективный адрес операнда является частью команды. Пример на языке ассемблера:

```
MOV DX, [0A12h].
```

Команда состоит из 4 байт (рисунок 8.1 б): код операции, способ адресации, эффективный адрес (2 байта). В результате в регистре ***DX*** окажется значение F145h, которое хранилось в ячейке по адресу 0A12h.



а) - непосредственный б) - прямой

Рисунок 8.1 – Способы адресации

Регистровый. Операнд содержится в определяемом командой регистре. 16-битный операнд может находиться в регистрах ***AX***, ***BX***, ***CX***, ***DX***, ***SI***, ***DI***, ***SP*** или ***BP***, а 8-битный - в регистрах ***AL***, ***AH***, ***BL***, ***BH***, ***CL***, ***CH***, ***DL*** или ***DH***.

Пример на языке ассемблера:

```
ADD AX,CX.
```

Команда выполняет сложение значений регистров ***AX*** и ***CX*** и помещает результат в регистр ***AX*** (AX:=AX+CX). Достоинством данного способа адресации является небольшой размер команды (требуется всего 2 байта: КОП и способ адресации) и минимальное время выполнения команды (данные находятся внутри МП и нет необходимости выполнения команд по извлечению данных из ОЗУ).

Регистровый косвенный. Эффективный адрес (***EA***) операнда находится в ба-

зовом регистре ***BX*** или индексном регистре:

$$EA = \begin{Bmatrix} (BX) \\ (SI) \\ (DI) \end{Bmatrix}. \qquad (8.1)$$

Пример на языке ассемблера:

ADD DL,[*BX*]; сложить содержимое регистра ***DL*** и ячейки памяти, эффективный адрес которого хранится в регистре ***BX*** (рисунок 8.2).

К достоинствам данного способа адресации относится небольшая длина команды в сравнении с прямой адресацией.

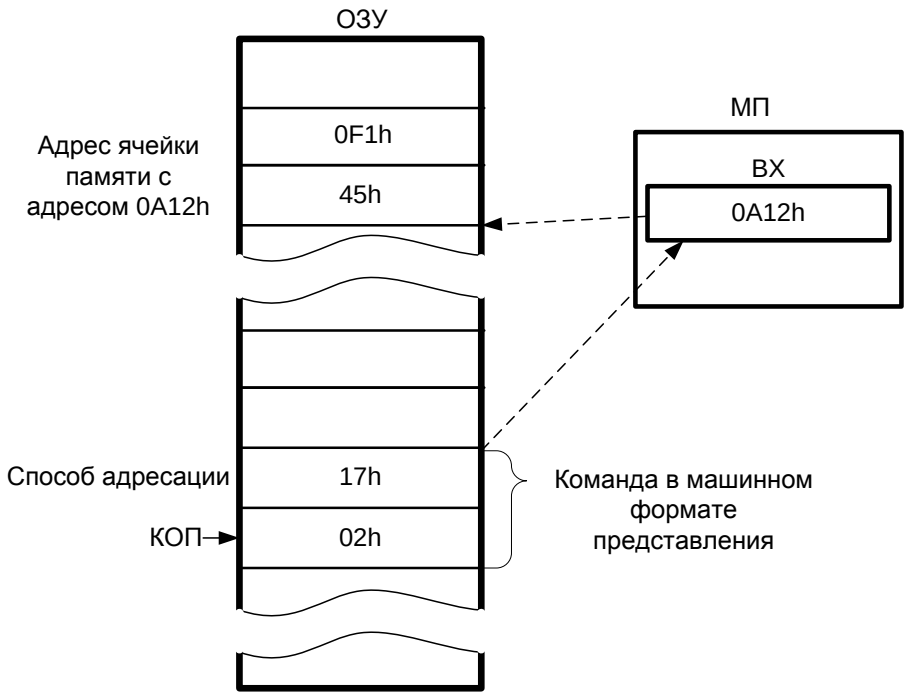


Рисунок 8.2 – Регистровый косвенный способ адресации

Данный способ адресации удобно использовать при работе с массивами, например, при расчете среднего значения. В этом случае в начале цикла в регистр (базовый или индексный) заносится эффективный адрес первого элемента массива:

LEA BX,mas_b;

где ***LEA*** – команда вычисления эффективного адреса переменной ***mas_b***, который помещается в регистр ***BX***.

Далее в каждой итерации значение регистра увеличивается на ***n***, где ***n*** – размер одного элемента в байтах:

ADD BX,2; увеличение значения регистра ***BX*** на 2

Регистровый относительный. Эффективный адрес равен сумме 8- или 16-битного смещения и содержимого базового или индексного регистров.

$$EA = \begin{Bmatrix}(BX)\\(BP)\\(DI)\\(SI)\end{Bmatrix} + \begin{Bmatrix}8-\text{битное смещение}\\16-\text{битное смещение}\end{Bmatrix}. \qquad (8.2)$$

Пример на языке ассемблера:

ADD AL,[BX+3]

Данный способ можно использовать при обращении к сложным структурам данных, например, к записям. Базовый регистр адресует начало записи, а конкретный элемент записи определяется смещением.

Базовый индексный. Эффективный адрес равен сумме содержимого базового и индексного регистров, определяемых командой:

$$EA = \begin{Bmatrix}(BX)\\(BP)\end{Bmatrix} + \begin{Bmatrix}(SI)\\(DI)\end{Bmatrix}. \qquad (8.3)$$

Пример на языке ассемблера:

ADD AL,[BX+SI]

Удобно использовать базовый индексный способ при работе с матрицами.

Относительный базовый индексный. Эффективный адрес равен сумме 8- или 16-битного смещения и базово-индексного адреса:

$$EA = \begin{Bmatrix}(BX)\\(BP)\end{Bmatrix} + \begin{Bmatrix}(SI)\\(DI)\end{Bmatrix} + \begin{Bmatrix}8-\text{битное смещение}\\16-\text{битное смещение}\end{Bmatrix}. \qquad (8.4)$$

Пример на языке ассемблера:

ADD AL,[BX+SI+3]

8.1.2 Определение массива

Массив - это структурированный тип данных, состоящий из некоторого числа элементов одного типа.

При необходимости представления данных в виде массива в программе можно

воспользоваться следующими способами:

- перечислением элементов массива в поле операндов одной из директив описания данных;

- использование оператора повторения ***dup***.

Ниже приведены примеры представления массива данных на языке ассемблера:

```
mas_b     db    0,1,2,3,4,5
mas_w     dw    6     dup (0).
```

В первом случае выделяется блок памяти размером в 6 байт со значениями 0,1,2,3,4,5, во втором – 12 байт со значениями равными 0.

Для получения доступа к элементам массива обычно используют следующие способы адресации: регистровый косвенный, регистровый относительный, базовый. Во всех этих вариантах сначала нужно определить адрес первого элемента массива. Для этого можно воспользоваться командой ***LEA*** (Load effective address). Команда имеет следующий синтаксис:

LEA приемник, источник

Пример:

```
LEA BX,mas_b
```

На языке ассемблера нет специальных средств для работы с двумерными массивами. Двумерный массив нужно моделировать. Программист при разработке алгоритма, определяет некоторую область памяти как двумерный массив. При этом блок памяти фактически представляет собой одномерный массив и программист сам определяет расположение элемента по адресу (***i***,***j***), где ***i*** и ***j*** –номер строки и столбца матрицы.

Пример кода, показывающий работу с двумерным массивом:

```
Matr db 25 dup(0); определяем блок памяти под матрицу 5x5
-----------------------------------
-----------------------------------
Lea bx,matr; загружаем в BX эффективный адрес первого элемента матрицы
Mov DI,2      ;помещаем в регистр Di значение 2
Add AX,[BX+DI] ; Ax:=AX+matr[0,2]
Add BX,5
Add BX,5
INC DI
Add ax,[bx+di] ; Ax:=AX+matr[2,3]
```

В примере регистр ***BX*** определяет номер строки, а ***DI*** – номер столбца. Соответственно, при изменении номера строки значение регистра ***BX*** изменяется на ***M***:

$$M=N \cdot L, \qquad (8.5)$$

где ***N*** –количество столбцов матрицы;

L – размер одного элемента в байтах.

Регистр ***DI*** изменяет свое значение на ***L***.

8.1.3 Организация циклов

Обработка данных массива обычно имеет циклический характер. Цикл можно организовать при помощи команды:

LOOP метка

Данная команда проверяет содержимое регистра ***CX***, если оно не равно нулю, то уменьшает содержимое ***CX*** и осуществляет переход на адрес соответствующей метке. Иначе выполняется следующая команда.

Пример:

```
Mov ax,0
mov cx,5      ; в CX помещаем количество шагов цикла
loop1:        ; устанавливаем метку LOOP1
inc ax        ; увеличиваем AX на 1
loop loop1    ; переход на метку LOOP1,если не выполнено5 операций.
; в результате AX=5
```

Также циклы организуются при помощи команд условного перехода:

```
Mov ax,0          ; обнуляем содержимое AX
Mov dx,5          ; DX=5
Loop1:            ; метка
Inc ax            ; AX=AX+1
Dec dx            ; dx=dx-1
Jnz loop1         ; если DX<>0 не равен нулю, то переход на метку
                  ;LOOP1
```

8.2 Задание к выполнению лабораторной работы

1. Разработать программу, которая вычисляет среднее значение массива однобайтных чисел размерности ***K***. Заполняется массив первоначально данными от виртуального устройства «Thermometer». Результат выводится на устройство «Display».

2. Разработать программу, которая заполняет массив размерности ***K*** данными от виртуального устройства «Thermometer». Каждый элемент массива состоит из ***L*** байт. Необходимо найти минимальное значение ***i***-го байта среди всех элементов массива. Результат выводится на устройство «Display».

3. Разработать программу, которая определяет максимальные значение для каждой строки матрицы размерности ***K***x***N*** и из них выбирает минимальное. Каждый элемент матрицы занимает в памяти 1 байт. Матрица заполняется данными от устройства «Thermometer». Результат выводится на устройство «Display».

Таблица 8.1 – Варианты заданий

Номер варианта	***K***	***A***1	***A***2	***L***	***I***	***A***3	***A***4	***A***5	***A***6
1	15	0	11	3	2	10	9	8	11
2	16	1	0	3	2	11	10	9	10
3	17	2	1	3	3	0	11	10	9
4	18	3	2	3	3	1	0	11	8
5	15	4	3	3	2	2	1	0	10
6	16	5	4	4	2	3	2	1	9
7	17	6	5	4	2	4	3	2	8
8	18	7	6	4	3	5	4	3	11
9	15	8	7	4	3	6	5	4	9
10	16	9	8	4	4	7	6	5	8
11	17	10	9	5	2	8	7	6	11
12	18	11	10	5	4	9	8	7	10
13	15	0	11	5	3	10	9	8	11
14	16	1	0	5	3	11	10	9	10
15	17	2	1	5	5	0	11	10	9
16	18	3	2	6	3	1	0	11	8
17	15	4	3	6	4	2	1	0	10
18	16	5	4	6	5	3	2	1	9
19	17	6	5	6	2	4	3	2	8
20	18	7	6	6	6	5	4	3	11

Таблица 8.2 –Коды способов адресации

Код	Способ адресации
0	Косвенная (***SI***)
1	Косвенная (***DI***)
2	Косвенная (***BX***)
3	Косвенная (***BP***)
4	Базовая адресация (***BP***+смещение)
5	Базовая адресация (***BX***+смещение)
6	Индексная адресация (***SI***+смещение)
7	Индексная адресация (***DI***+смещение)
8	Базово-индексная (***BX***+***SI***)
9	Базово-индексная (***BX***+***DI***)
10	Базово-индексная (***BP***+***SI***)
11	Базово-индексная (***BP***+***DI***)

8.3 Порядок выполнения работы

1. Создайте новый проект в среде «Emu8086». В программе определите массив однобайтных чисел размерностью ***K masB***:

```
Jmp start              ;пропускаем блок памяти, выделенный под массив,
                       ;чтобы не выполнить их как команды
Masb db 10 dup(0)      ; выделяем блок памяти под массив
Start:                 ;начало программы
```

2. Разработайте алгоритм, который заполните массив данными, полученными от термометра. При заполнении массива используйте способ адресации согласно своему варианту (код ***A***1 в таблице 8.1). Расшифровка кода способа адресации представлена в таблице 8.2.

3. Разработайте алгоритм, позволяющий определить среднее значение массива ***masB*** и выводящий результат на дисплей. Для доступа к элементам массива используйте способ адресации, заданный кодом ***A***2 в таблице 8.1

4. Запустите программу в режим эмуляции, выбрав пункт меню «Emulate». Выберите устройства «Thermometer.exe» и «Led_Display» из пункта меню «virtual devices». Включите спиртовку, нажав на окне устройства кнопку «***On***», чтобы значения элементов в массиве были различными (или выключите, если термометр по-

казывает максимальную температуру).

5. Выполните программу в пошаговом режиме, нажав кнопку «Step» в окне emulator. Убедитесь в корректности разработанного программного кода.

6. Создайте новый проект в среде «Emu8086», используя шаблон «Bin template».

7. В коде программы объявите массив ***masB*** размерности ***K***, каждый элемент которого состоит из ***L*** байтов. Ниже приведен пример, объявления массива из 10 элементов по 5 байт:

```
Jmp start
masB db 10 dup(0,1,2,3,4)
Start:
```

8. Разработайте алгоритм, который заполнит массив данными, полученными от термометра (если массив состоит из 10 элементов, то необходимо 50 раз получить данные от термометра). При заполнении массива используйте способ адресации согласно своему варианту (код ***A***3).

9. Разработайте программный код на языке ассемблера, позволяющий определить минимальное значение ***i***-го байта среди всех элементов массива ***masB*** и выводящий результат на дисплей. Для доступа к элементам массива используйте способ адресации заданный кодом ***A***4 в таблице 8.1

10. Выполните пункты 4 и 5. Убедитесь в корректности разработанной программы.

11. Создайте новый проект в среде «Emu8086», используя шаблон «Bin template».

12. В коде программы объявите матрицу ***matr*** размерности ***K***x***N.*** Ниже приведен пример, объявления матрицы 5x5:

```
Jmp start
matr db 25 dup(0)
Start:
```

13. Разработайте алгоритм, который заполнит матрицу данными, полученными от термометра. При заполнении массива используйте способ адресации согласно своему варианту (код ***A***5).

14. Разработайте алгоритм, позволяющий определить минимальное значение

из максимальных, определенных для каждой строки. Для доступа к элементам матрицы используйте способ адресации заданный кодом ***A***6 в таблице 8.1

15. Сделайте выводы по работе и составьте отчет.

8.4 Контрольные вопросы

1. Что понимается под способом адресацией?

2. Что означает директива ***DUP***?

3. Команда с каким способом адресации занимает больше места в ОЗУ: прямым или регистровым косвенным?

4. В чем особенность регистрового относительного способа адресации?

5. Какой способ адресации лучше использовать при обработке данных, представленных одномерным массивом?

6. Как организуется двумерный массив данных средствами языка ассемблера?

7. Что выполняет команда ***LEA***?

8. Как организовать цикл на языке ассемблера?

9. Как микропроцессор определяет, где находится операнд при непосредственной адресации?

Список использованных источников

1 Булатов, В.Н. Основы микропроцессорной техники : учеб. пособие / В. Н. Булатов . - Оренбург : ГОУ ОГУ, 2008. - 268 с.

2 Калабеков, Б.А. Цифровые устройства и микропроцессорные системы / Б.А. Калабеков - М.: Горячая линия – Телеком, 2002.-336 с.

3 Мелехин, В.Ф. Вычислительные машины, системы и сети: учебник для студ. высш. учеб. заведений/В.Ф. Мелехин, Е.Г. Павловский. -2-е издание., стер. –М.: Издательский центр «Академия», 2007. -560 с.

4 Юров, В.И. Assembler: учебное пособие для вузов/ В.И. Юров. -2-е изд. – СПб.: Питер, 2007.-637 с.

5 Цифровая и микропроцессорная техника: методические указания [Электронный ресурс]. Режим доступа: http://opprib.ru/main/labor/cpu_my.php. Проверено 06.01.2012.

Приложение А

Описание используемых в работе команд МП 8086

ADD – сложение. Команда выполняет арифметическое сложение приемника и источника, помещает сумму в приемник, не изменяя содержимое источника. Приемник может быть регистром или переменной, источник - числом, регистром или переменной, но нельзя использовать переменную одновременно и для источника, и для приемника. Команда ***ADD*** никак не различает числа со знаком и без знака, но, употребляя значения флагов ***CF, OF*** и ***SF***, разрешается применять ее и для тех, и для других.

ADC – сложение с переносом. Эта команда аналогична ***ADD***, но при этом выполняет арифметическое сложение приемника, источника и флага ***CF.***

CMP –сравнение. Сравнивает источник и приемник и устанавливает флаги. Действие осуществляется вычитанием источника (число, регистр или переменная) из приемника (регистр или переменная). Результат никуда не записывается.

SUB – вычитание. Вычитает источник из приемника и помещает разность в приемник. Приемник - может быть регистром или переменной, источник - числом, регистром или переменной, но нельзя использовать переменную одновременно и для источника, и для приемника. Точно так же, как и команда ***ADD, SUB*** не делает различий между числами со знаком и без знака, но флаги позволяют использовать ее и для тех и для других.

SBB – вычитание с заемом. Эта команда аналогична ***SUB***, но она вычитает из приемника значение источника и дополнительно вычитает значение флага ***CF.***

JMP – команда безусловного перехода. Осуществляет переход по адресу указанному в команде. Операндом может быть непосредственный адрес, регистр или переменная.

PUSH. Помещает данные в стек. Формат команды:

PUSH источник

Источником может быть регистр, сегментный регистр, непосредственный опе-

ранд или память. Фактически эта команда уменьшает ***SP*** на 2 и копирует содержимое источника в память по адресу ***SS***:***SP***. Команда PUSH почти всегда используется в паре с командой ***POP.***

POP. Считывает данные из стека. Формат команды:

POP приемник

Помещает в приемник слово, находящееся в вершине стека, увеличивая ***SP*** на 2. ***POP*** выполняет действия обратные ***PUSH***. Приемником может быть регистр общего назначения, сегментный регистр, кроме ***CS***, переменная.

PUSHF. Помещает содержимое регистра флагов в стек. Формат команды:

PUSF

Копирует содержимое регистра флагов в стек.

POPF. Загружает регистр флагов из стека. Формат команды:

POPF

Считывает из вершины стека слово и помещает в регистр флагов.

LOOP. Используется для организации циклов, в которых регистр ***CX*** играет роль счетчика.

Команда имеет следующий формат:

LOOP метка

Уменьшает регистр ***CX*** на 1 и выполняет переход на метку, если ***CX*** не 0.

IN приемник, источник.

Копирует число из порта ввода-вывода, номер которого указан в источнике, в приемник. Приемником может быть только ***AL, AX***. Источник или непосредственный адрес порта, или регистр ***DX***, причем во время использования непосредственного операнда можно указывать лишь номера портов не больше 255.

OUT приемник, источник.

Копирует число из источника (***AL, AX***) в порт ввода/вывода, номер которого указан в приемнике. Приемником может либо непосредственно операнд, либо регистр ***DX***.

XOR приемник, источник. Команда выполняет побитовое «исключающее ИЛИ» над приемником и источником и помещает результат в приемник.