МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Государственное образовательное учреждение высшего профессионального образования
Оренбургский государственный университет

Кафедра систем автоматизации производства

А.П.Карягин

АРХИТЕКТУРА МИКРОПРОЦЕССОРОВ И ИХ ПРОГРАММИРОВАНИЕ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
К ЛАБОРАТОРНЫМ И САМОСТОЯТЕЛЬНЫМ РАБОТАМ

Рекомендовано к изданию Редакционно-издательским советом
государственного образовательного учреждения
высшего профессионального образования
↔Оренбургский государственный университет≈

Оренбург 2004

ББК 32.973я7
К 27
УДК 681.3+681.5](075)

Рецензент
кандидат технических наук, доцент Черноусова А.М.

Карягин А.П.
К 27 **Архитектура микропроцессоров и их программирование: Методические указания к лабораторным и самостоятельным работам. - Оренбург: ГОУ ОГУ, 2004.-58 с.**

Методические указания предназначены для выполнения лабораторных и самостоятельных работ по курсу "Вычислительные машины, системы и сети" для студентов третьего курса по специальности 210200. Могут использоваться студентами других специальностей при изучении микропроцессорных вычислительных систем и их программирования.

ББК 32.973я7

Введение

В настоящее время во всех областях человеческой деятельности широко используются электронно-вычислительные машины, системы и сети, что прежде всего обусловлено бурным развитием интегральных технологий изговления микросхем, когда появилась возможность на одном кристалле размещать схемы многофункциональных устройств (процессоров, контроллеров, логических устройств и т.п.), включающих в себя миллионы транзисторных структур. На современном этапе развития вычислительной техники использование сверхбольших интегральных схем (СБИС) стало неотъемлемой частью построения электронно-вычислительных (ЭВМ) и вычислительных систем, микро-ЭВМ и персональных компьютеров.

Первые персональные компьютеры IBM, появившиеся в 1981 г. и получившие название IBM PC, использовали в качестве центрального вычислительного узла 16-разрядный микропроцессор Intel 8088, который в дальнейшем был заменён полностью 16-разрядным процессором Intel 8086.

В 1983 году корпорацией Intel был предложен микропроцессор 80286, в котором был реализован новый режим работы, получивший название защищённого. В дальнейшем на смену процессору 80286 пришли модели 80386, 80486 и различные варианты процессора Pentium, которые могут работать и в реальном, и в защищённом режимах.

Наиболее полное представление об архитектурных и структурных особенностях микропроцессоров (МП) и микропроцессорных вычислительных систем (МПС), в частности, корпорации Intel позволяет получить понимание и умение составлять программы на языке ассемблера для этих процессоров и систем. Кроме того, язык ассемблера используется как инструмент отладки как в системах программирования на языках высокого уровня, так и для отладки программ, не имеющих исходного текста.

Хотя каждая модель была совершеннее предыдущей (в частности, почти на два порядка возросла скорость работы процессора, начиная с модели 80386 процессор стал 32-разрядным, а в процессорах Pentiuum реализован даже 64-разрядный обмен данными с системной шиной), однако с точки зрения программиста все эти процессоры весьма схожи.

Существенным моментом в освоении "компьютерной среды" является знание системы прерываний и системы ввода-вывода, что наиболее важно при проектировании автоматизированных систем управления.

В лабораторной работе 1 студенты получают представление об архитектуре МП и МПС на уровне машинного языка и мнемонического представления команд с помощью программы-отладчика DEBUG. В остальных работах продолжается изучение архитектуры и логических основ ЭВМ и приобретаются навыки программирования на языке ассемблера.

1 Изучение архитектуры микропроцессоров

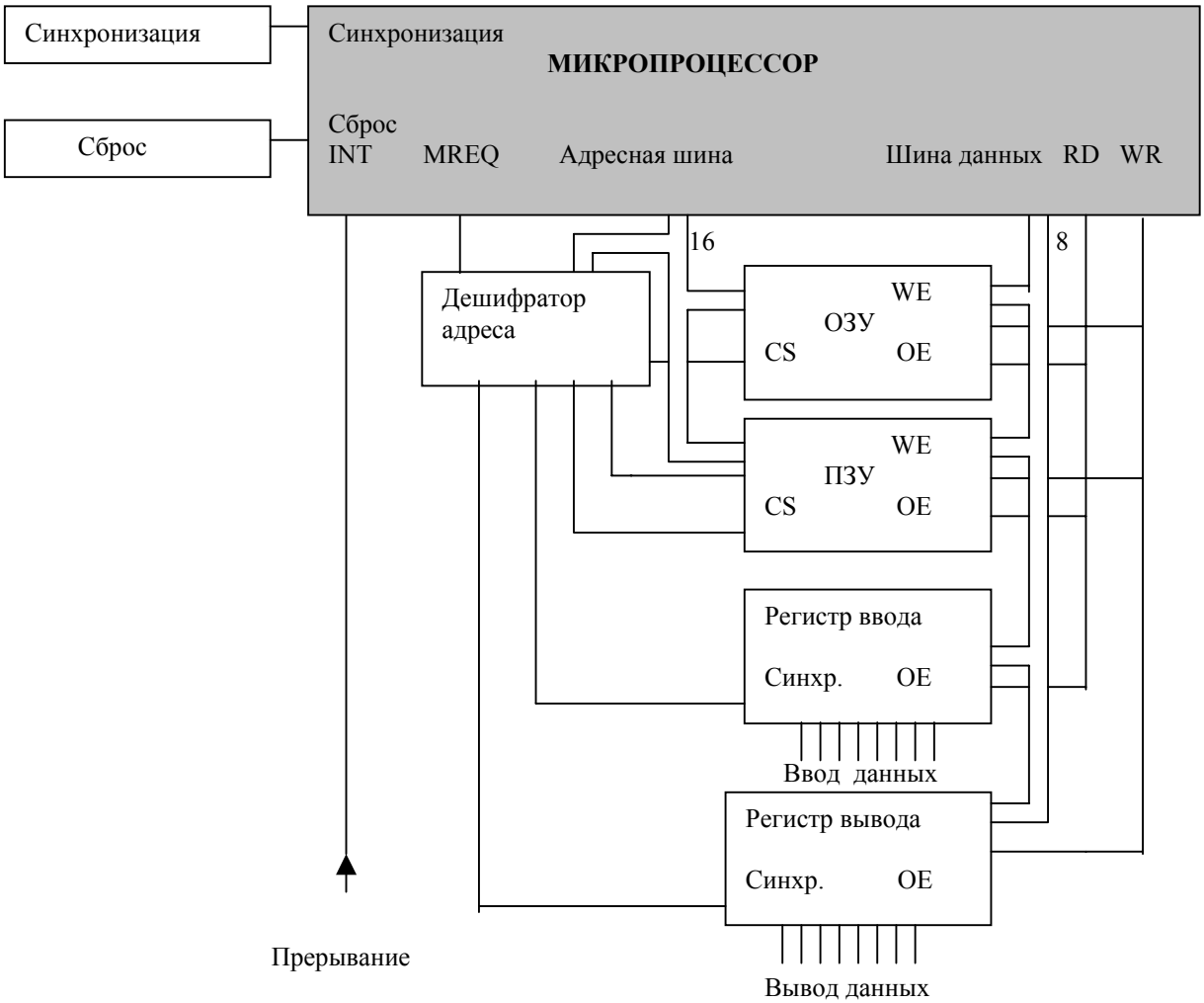
1.1 Цель работы

Целью работы является исследование архитектурных характеристик отдельных микропроцессоров и приобретение навыков записи программ в мнемокодах и машинных кодах.

1.2 Общие положения

1.2.1 Архитектура микропроцессорных систем

Центром вычислительной системы является ее процессор. Это основное звено, или "мозг" компьютера.



Обозначения: INT - прерывание; MREQ - запрос памяти; RD - управление чтением; WR - управление записью; OE - разрешение по выходу; CS - выбор микросхемы; WE - разрешение записи

Рисунок 1.1- Микропроцессорная вычислительная система

Именно процессор обладает способностью выполнять команды, составляющие компьютерную программу. Персональные компьютеры строятся на базе микропроцессоров, выполняемых в настоящее время на одном кристалле или "чипе".

На рисунке 1.1 представлена структура микропроцессорного устройства (МПУ), выполненного на базе 8-разрядного микропроцессора с архитектурой типа CISC, например, отечественного МП КР580ВМ80А или МП Intel 8086.

Для программиста, разрабатывающего программное обеспечение микропроцессорной системы (МПС) на языке ассемблера, микропроцессор (МП) представляется в виде следующих элементов: памяти, счетчика команд, рабочих регистров процессора, регистров признаков, стека и указателя стека, портов ввода-вывода, системы прерывания, набора команд.

При программировании МПС наиболее важное значение имеет знание *архитектуры* МП. В общем случае под архитектурой МП понимают совокупность следующих компонентов и характеристик:

- разрядность адресов и данных;
- состав, имена и назначение программно-доступных регистров;
- форматы системы команд;
- режимы адресации памяти;
- способы машинного представления данных разного типа;
- способ адресации внешних устройств и средства выполнения операций ввода-вывода;
- особенности инициирования и обработки прерываний.

Программы, управляющие функционированием МПС, могут быть расположены в различных типах памяти. Наиболее часто используют постоянное (ПЗУ) и оперативное (ОЗУ) запоминающие устройства. Из ПЗУ МП может лишь считывать команды и данные, но не может модифицировать его содержимого в отличие от ОЗУ, для которого возможно как считывание, так и запись любой информации.

1.2.2 Архитектура микропроцессора

Архитектура МП Intel 8086 в упрощённом виде представлена на рисунке 1.2. С функциональной точки зрения МП можно разделить на две части: операционное устройство и шинный интерфейс. Все компоненты МП взаимодействуют между собой посредством систем шин: адресных (16 разрядов), данных (8 разрядов) и управления. В процессоре 8086 имеется несколько быстрых элементов памяти на интегральных схемах, которые называются регистрами. Каждый из регистров предоставляет определенные возможности, которые другими регистрами или ячейками памяти не поддерживаются.

Регистры разбиваются на четыре категории (см. рисунок 1.2):

- регистр флагов;
- регистры общего назначения AX, BX, CX, DX, BP, SP,DI и SI;
- указатель инструкций IP;
- сегментные регистры CS, SS, DS и ES.

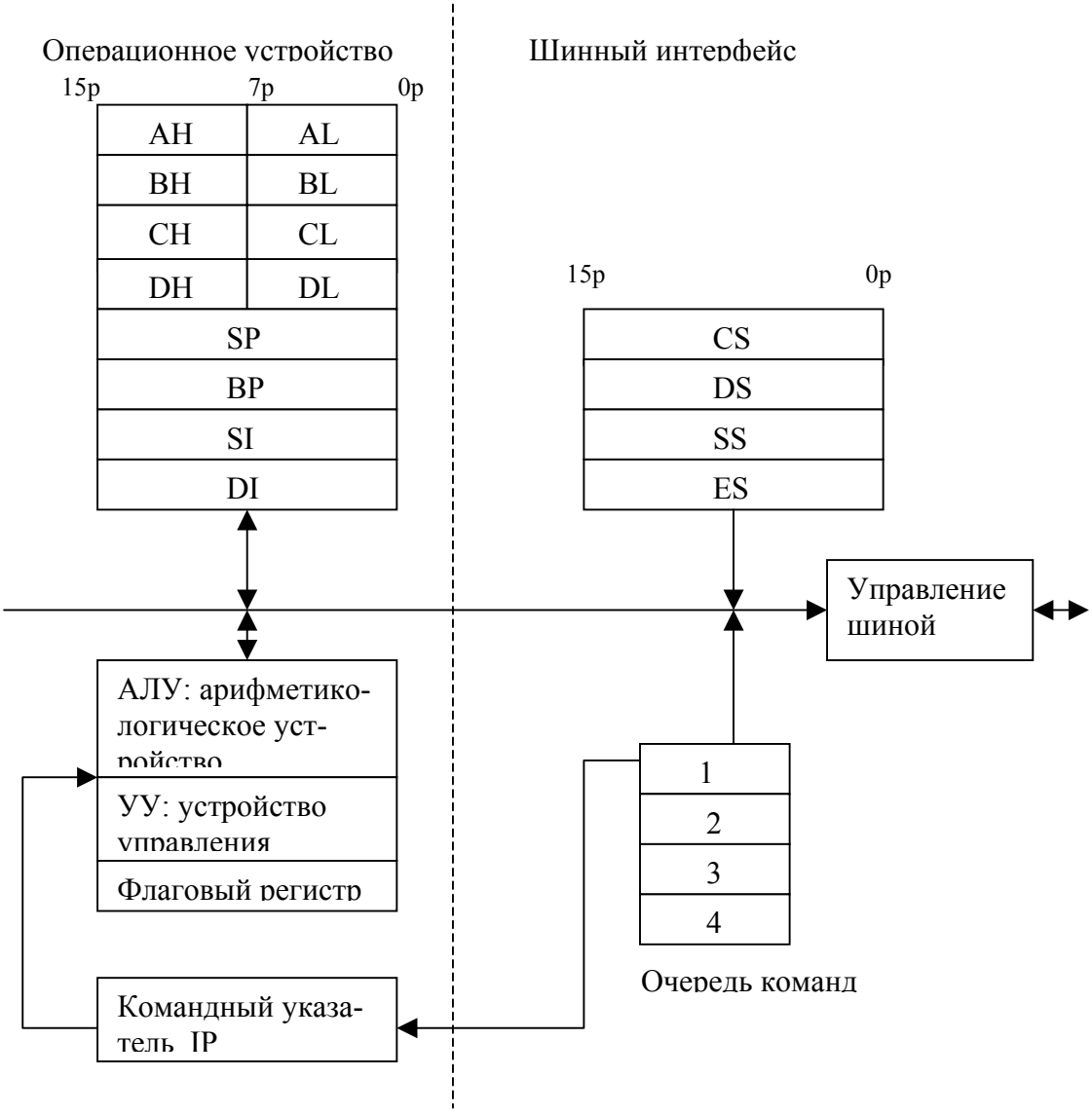


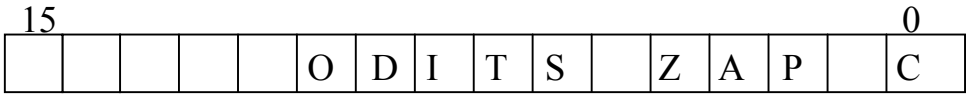
Рисунок 1.2- Операционное устройство и шинный интерфейс

1.2.3 Регистр флагов

Регистр флагов - это 16-разрядный (16-битовый) регистр содержит всю необходимую информацию о состоянии процессора 8086 и результатах последних инструкций (рисунок 1.3).

Например, если вы хотите знать, получен ли при вычитании нулевой результат, непосредственно после этой инструкции вам следует проверить флаг нуля (бит Z в регистре флагов). Если он установлен (то есть имеет ненулевое значение), это будет говорить о том, что результат нулевой. Другие флаги, такие, как флаги переноса и переполнения аналогичным образом сообщают о результатах арифметических и логических операций.

Другие флаги управляют режимом операций процессора 8086. Флаг направления управляет направлением, в котором строковые инструкции выполняют перемещение, а флаг прерывания управляет тем, будет ли разрешено внешним аппаратным средствам.



Битовые флаги:

O - флаг переполнения;
D - флаг направления;
I - флаг прерывания;
T - флаг перехвата;
S - флаг знака;
Z - флаг нуля;
A - флаг дополнительного переноса;
P - флаг четности;
C - флаг переноса.

Рисунок 1.3 - Регистр флагов процессора 8086.

Так, например, клавиатура или модем, могут временно приостанавливать текущий код для выполнения функций, требующих немедленного обслуживания. Флаг перехвата используется только программным обеспечением, которое служит для отладки другого программного обеспечения (отладчики).

Регистр флагов не считывается и не модифицируется непосредственно. Вместо этого регистр флагов управляется в общем случае с помощью специальных инструкций (таких, как CLD, STI и CMC), а также с помощью арифметических и логических инструкций, модифицирующих отдельные флаги. И наоборот, содержимое отдельных разрядов регистра флагов влияет на выполнение инструкций (например, JZ, RCR и MOVSB). Регистр флагов не используется на самом деле, как ячейка памяти, вместо этого он служит для контроля за состоянием и управления процессором 8086.

1.2.4 Регистры общего назначения

Восемь регистров общего назначения (или общих регистров) процессора 8086 (каждый размером 16 бит) используются в операциях большинства инструкций в качестве источника или приемника при перемещении данных и вычислениях, указателей на ячейки памяти и счетчиков. Каждый регистр общего назначения может использоваться для хранения 16-битового значения, в арифметических и логических операциях, может выполняться обмен между регистром и памятью (запись из регистра в память и наоборот). Например, в данном фрагменте программы:

```
mov   ax,5
mov   dx,9
add   ax,dx
  :
```

значение 5 загружается в регистр AX, значение 9 - в DX, и эти два значения складываются вместе.

При этом результат (14) сохраняется в регистре AX. Вместо регистров AX и DX здесь можно использовать регистр CX, SI или любой другой регистр общего назначения.

Кроме такого общего свойства регистров, как использования их для хранения значений или в качестве источника и приемника при работе в инструкциях с данными, каждый регистр общего назначения имеет свою особенность.

Регистр AX называют также накопителем (аккумулятором). Этот регистр всегда используется в операциях умножения или деления и является также одним из тех регистров, который можно использовать для наиболее эффективных операций (арифметических, логических или операций перемещения данных).

Младшие 8 бит регистра AX называются также регистром AL, а старшие 8 бит - регистром AH. Это может оказаться удобным при работе с данными размером в байт. Таким образом, регистр AX можно использовать, как два отдельных регистра. В следующем фрагменте программы регистр AH устанавливается в значение 0, это значение копируется в AL и затем в регистр AL добавляется 1:

```
.
mov   ah,0
mov   al,ah
inc   al
.
```

В результате в регистре AX будет записано значение 1. Регистры BX, CX и DX могут аналогичным образом использоваться либо как один 16-разрядный регистр, либо как два 8-разрядных.

Регистр BX может использоваться для ссылки на ячейку памяти (указатель). Если говорить кратко, то 16-битовое значение, записанное в BX, может использоваться в качестве части адреса ячейки памяти, к которой производится доступ. Например, следующий код загружает в AL содержимое адреса памяти 9:

```
:
mov   ax,0
mov   ds,ax
mov   bx,9
mov   al,[bx]
:
```

Как можно заметить, перед обращением к ячейке памяти, на которую указывает BX, мы загрузили в DS значение 0 (через регистр AX). Это результат сегментной организации памяти процессора 8086. По умолчанию, когда BX используется в качестве указателя на ячейку памяти, он ссылается на нее относительно сегментного регистра DS. Регистр BX может интерпретироваться, как два восьмибитовых (8-разрядных) регистра - BH и BL.

Специализация *регистра CX* - использование в качестве счетчика. Предположим, мы хотим 10 раз повторить выполнение блока инструкций. Это можно сделать следующим образом:

```
        mov   cx,10
Begin:
      <блок инструкций, который нужно повторить>
        sub   cx,1
        jnz   Begin
        :
```

Инструкции между меткой Begin и инструкцией JNZ будут повторяться до тех пор, пока содержимое регистра CX не станет равным 0. Заметим, что чтобы уменьшить содержимое CX и перейти на начало цикла Begin, если регистр CX еще не равен 0, здесь используются две инструкции - SUB CX,1 и JNZ.

Уменьшение значения счетчика и цикл - это часто используемый элемент программы, поэтому в процессоре 8086 используется специальная инструкция для того, чтобы циклы выполнялись быстрее и были более компактными. Эта инструкция называется LOOP. Инструкция LOOP (инструкция цикла) вычитает 1 из значения регистра CX и выполняет переход, если содержимое регистра CX не равно 0 (все это в одной инструкции). Для приведенного выше примера можно записать такой эквивалент:

```
        :
        mov   cx,10
Begin:
      <блок инструкций, который нужно повторить>
        :
        loop Begin
        :
```

Регистр CX можно интерпретировать, как два 8-разрядных регистра - CH и CL.

Регистр DX - это единственный регистр, которые может использоваться в качестве указателя адреса ввода-вывода в инструкциях IN и OUT. Фактически, кроме использования регистра DX нет другого способа адресоваться к портам ввода-вывода с 256 по 65535. Например, в следующем фрагменте программы в порт 878h (LPT) записывается значение 62:

```
        :
        mov   al,62
        mov   dx,878h
        out   dx,al
        :
```

Другие уникальные качества регистра DX относятся к операциям деления и умножения. Когда вы делите 32- или 16-битовый делитель, старшие 16 бит делимого должны быть помещены в регистр DX. После выполнения деления остаток также сохраняется в DX. (Младшие 16 бит делимого должны быть помещены в AX. Частное от деления также будет записано в AX.) Ана-

логично, когда вы перемножаете два 16-битовых сомножителя, старшие 16 бит произведения сохраняются в DX (младшие 16 бит записываются в регистр AX). Регистр DX можно интерпретировать, как два 8-разрядных регистра - DH и DL.

Регистр SI может использоваться, как указатель на ячейку памяти. Например:

```
:
mov   ax,0
mov   ds,ax
mov   si,20
mov   al,[si]
:
```

Здесь 8-битовое значение, содержащееся по адресу 20, записывается в регистр AL. Особенно полезно использовать регистр SI для ссылки на память в строковых инструкциях процессора 8086. Например:

```
:
mov   ax,0
mov   ds,ax
mov   si,20
mov   al,[si]
lodsb
:
```

Здесь не только содержимое по адресу памяти, на который указывает SI, сохраняется в регистре AX, но к SI также добавляется 1. Это может оказаться очень эффективным при организации доступа к последовательным ячейкам памяти (например, к строке текста). Кроме того, можно сделать так, что строковые инструкции будут автоматически определенное число раз повторять свои действия, так что отдельная инструкция может выполнить сотни, а иногда и тысячи действий.

Регистр DI очень похож на регистр SI в том плане, что его можно использовать в качестве указателя ячейки памяти. При использовании его в строковых инструкциях он имеет также особые свойства. Например:

```
:
mov   ax,0
mov   ds,ax
mov   di,1024
add   bl,[di]
lodsb
:
```

Здесь 8-битовое значение, расположенное по адресу 1024, записывается в регистр BL. При использовании его в строковых инструкциях регистр DI несколько отличается от регистра SI. В то время как SI всегда используется в строковый инструкциях, как указатель на исходную ячейку памяти (источник), DI всегда служит указателем на целевую ячейку памяти (приемник). Кроме того, в строковых инструкциях регистр SI обычно адресуется к памяти относи-

тельно сегментного регистра DS, тогда как DI всегда адресуется к памяти относительно сегментного регистра ES. Когда регистры SI и DI используются в качестве указателей на ячейки памяти в других инструкциях (не строковых), то они всегда адресуются к памяти относительно регистра DS. Например:

```
:
cld
mov   dx,0
mov   es,dx
mov   di,2048
stosb
:
```

Строковая инструкция STOSB используется здесь и для сохранения значения в регистре AL (по адресу памяти, на который указывает регистр DI), и для добавления к содержимому регистра DI 1.

Регистр BP также может использоваться в качестве указателя на ячейку памяти, но здесь есть некоторые отличия. Регистры BX, SI и DI обычно ссылаются на память относительно сегментного регистра DS (или, в случае использования в строковых инструкциях регистра DI, относительно сегментного регистра ES), а регистр BP адресуется к памяти относительно регистра SS (сегментный регистр стека).

Стек находится в сегменте, на который указывает регистр SS. Например:

```
:
push  bp
mov   bp,sp
mov   ax,[bp+4]
:
```

Здесь выполняется обращение к сегменту стека для загрузки в AX первого параметра, передаваемого при вызове Турбо Си подпрограммы на Ассемблере. Если говорить кратко, то регистр BP создан для обеспечения работы с параметрами, локальными переменными другой адресации к памяти с использованием стека.

Регистр SP называется также указателем стека. Это "наименее общий" из регистров общего назначения, поскольку он практически всегда используется для специальной цели - обеспечения стека. Стек - это область памяти, в которой можно сохранять значения и из которой они могут затем извлекаться по дисциплине "последний – пришел – первый - ушел" (LIFO). То есть последнее сохраненное в стеке значение будет первым значением, которое вы получите при чтении из стека.

Регистр SP в каждый момент времени указывает на вершину стека. Действие, состоящее в занесении значений в стек, называют также "заталкиванием" (pushing) в стек. В самом деле, инструкция PUSH используется для занесения значений в стек. Аналогично, действие, состоящее в извлечении (выборке) значений из стека, называют также "выталкиванием" (popping) из стека (для этого используется инструкция POP).

На рисунке 1.4 показывается, как изменяются регистры SP, AX и BP по мере выполнения следующего кода (при этом подразумевается, что начальное значение SP равно 1000, а круглые скобки означают содержимое регистров и ячеек памяти):

```
:
mov   ax,1
push  ax
mov   bx,2
push  bx
pop   ax
pop   bx
:
```

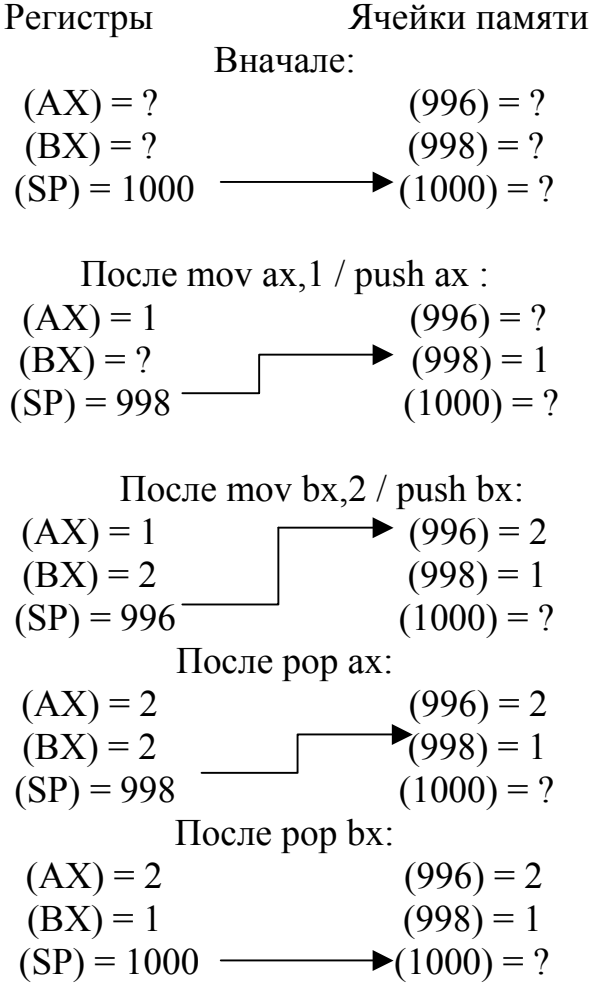


Рисунок 1.4 - Регистры AX, BX, SP и стек.

Внимание! Стек используется всякий раз, когда вы вызываете подпрограмму. Кроме того, стек используют некоторые системные ресурсы, когда они прерывают процессор, чтобы выполнить свои функции. Если вы измените SP, даже на несколько инструкций, то правильное значение стека может оказаться недоступным, когда он потребуется системным ресурсам.

1.2.5 Указатель инструкций

Указатель инструкций (регистр IP) всегда содержит смещение в памяти, по которому хранится следующая выполняемая инструкция. Когда выполняется одна инструкция, указатель инструкций перемещается таким образом, чтобы указывать на адрес памяти, где хранится следующая инструкция. Обычно следующей выполняемой инструкцией является инструкция, хранимая по следующему адресу памяти, но некоторые инструкции, такие, как вызовы или переходы, могут привести к тому, что в указатель инструкций будет загружено новое значение. Таким образом, будет выполнен переход на другой участок программы.

Значение счетчика инструкций нельзя прочитать или записать непосредственно. Загрузить в указатель инструкций новое значение может только специальная инструкция перехода (аналогичная только что описанным).

Указатель инструкций сам по себе не определяет адрес, по которому находится следующая выполняемая инструкция. Картину здесь опять усложняет сегментная организация памяти процессора 8086. Для извлечения инструкции предусмотрен регистр CS, где хранится базовый адрес, при этом указатель инструкций задает смещение относительно этого базового адреса.

1.2.6 Сегментация памяти процессора 8086 и сегментные регистры

Основной предпосылкой сегментации является следующее: процессор 8086 может адресоваться к 1 мегабайту памяти. Для адресации ко всем ячейкам адресного пространства в 1 мегабайт необходимы 20-разрядные сегментные регистры. Однако процессор 8086 использует только 16-разрядные указатели на ячейки памяти. Вспомним, например, что для ссылки на память используется 16-разрядный регистр BX. Как же тогда согласовать 16-разрядные указатели процессора 8086 и 20-разрядные адреса?

Ответ состоит в том, что процессор 8086 использует двухступенчатую схему адресации. Да, используются 16-разрядные указатели, но эта форма представляет собой только часть полной схемы адресации. Каждый 16-разрядный указатель памяти или смещение комбинируется с содержимым 16-разрядного сегментного регистра для формирования 20-разрядного адреса памяти.

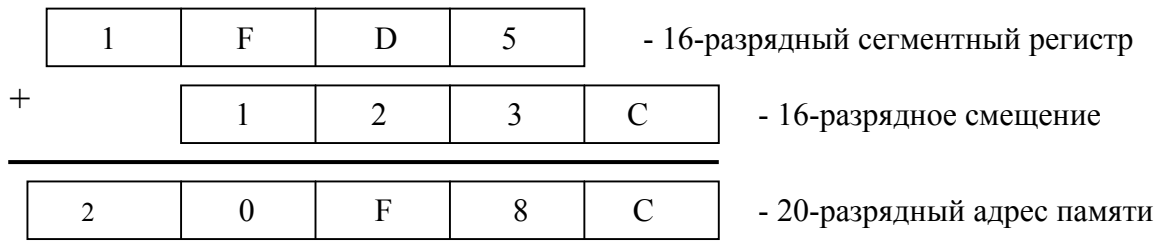


Рисунок 1.5 - 20-разрядные адреса памяти.

Сегменты и смещения комбинируются следующим образом: значение сегмента сдвигается влево на 4 разряда (то есть умножается на 16), а затем складывается со смещением, как показано на рисунке 1.5.

Рассмотрим, например, следующий фрагмент программы:

```
    :
    mov   ax,1000h
    mov   ds,ax
    mov   si,201h
    mov   dl,[si]
    :
```

Здесь для сегментного регистра DS устанавливается значение 1000h, SI устанавливается в значение 201h. Мы можем представить их в виде "сегмент: смещение" - 1000:201h. Адрес, из которого загружаются данные в DL, представляет собой:

((DS * 16) + SI) или ((1000h * 16) + 201h)

Итак, программа получает доступ к полному адресному пространству в 1 мегабайт с помощью использования только пары "сегмент: смещение". Все инструкции и режимы адресации процессора 8086 по умолчанию работают относительно того или иного сегментного регистра, хотя в некоторых инструкциях можно явно указать, что нужно использовать желаемый сегментный регистр. Обычно редко требуется загружать значение непосредственно в сегментный регистр, а вместо этого сегментные регистры в программе имеют символические имена сегментов, которые в ходе ассемблирования, компоновки и выполнения превращаются в числа. Это необходимо, поскольку нет способа сказать заранее, где в памяти будет находиться данный сегмент: это зависит от версии DOS, числа и размера резидентных в памяти программ, а также потребности в памяти остальной части программы. Использование имен сегментов позволяет Турбо Ассемблеру и операционной системе DOS выполнять подобные вычисления.

Наиболее общим именем сегмента для размещения данных является @Data, которое в *упрощенных директивах* определения сегментов используется для ссылки на используемый по умолчанию сегмент данных. Например:

```
        dosseg
       .model  small
       .data
var1    dw      0
       :
       .code
       mov     ax,@data
       mov     ds,ax
       :
       end
```

Здесь регистр DS загружается таким образом, что он будет указывать на используемый по умолчанию сегмент данных, в котором находится Var1.

Так как 64К - это максимальный объем памяти, к которой можно адресоваться с помощью 16-битового смещения, то при работе с большим (более 64К) объемом памяти значение сегментного регистра и смещение придется часто изменять.

Сегментные регистры не могут использоваться в качестве источников или приемников в арифметических и логических инструкциях и единственная операция, которую можно выполнять с сегментными регистрами, состоит в копировании значений между сегментными регистрами и другими общими регистрами или памятью. Например, чтобы добавить значение 100 к регистру ES, потребуется следующее:

```
:
mov   ax,es
add   ax,100
mov   es,ax
:
```

Из всего этого можно сделать заключение, что процессор 8086 лучше подходит для работы с памятью в блоках, не превышающих 64К.

Ещё одна особенность использования сегментов состоит в том, что каждая ячейка памяти адресуется через многие возможные сочетания "сегмент: смещение". Например, адрес памяти 100h адресуется с помощью следующих значений "сегмент: смещение": 0:100h, 1:F0h, 2:E0h и т.д., так как при вычислении всех этих пар "сегмент: смещение" получается значение адреса 100h.

Аналогично регистрам общего назначения каждый сегментный регистр играет свою, конкретную роль. Регистр CS указывает на код программы, DS указывает на данные, SS - на стек, сегмент (сегментный регистр) ES - это "трафаретный" (дополнительный) сегмент, который может использоваться так, как это необходимо.

Регистр CS указывает на начало блока памяти объемом 64К, или сегмент кода, в котором находится следующая выполняемая инструкция. Следующая инструкция, которую нужно выполнить, находится по смещению, определяемому в сегменте кода регистром IP, то есть на нее указывает адрес (в форме "сегмент:смещение") CS: IP. Никакие другие режимы адресации или указатели памяти, отличные от IP, не могут нормально работать относительно регистра CS.

Регистр DS указывает на начало сегмента данных, которые представляет собой блок памяти объемом 64К, в котором находится большинство размещенных в памяти операндов. Обычно для ссылки на адреса памяти используются смещения, предполагающие использование регистров BX, SI или DI.

Регистр ES указывает на начало блока памяти объемом 64К, который называется дополнительным сегментом. Как и подразумевает его название, дополнительный сегмент не служит для какой-то конкретной цели, но доступен тогда, когда в нем возникает необходимость. Иногда дополнительный сегмент используется для выделения дополнительного блока памяти объемом 64К для данных. Однако доступ к памяти в дополнительном сегменте менее эффективен, чем доступ к памяти в сегменте данных. Особенно полезен до-

полнительный сегмент, когда используются строковые инструкции. Все строковые инструкции, которые выполняют запись в память, используют в качестве адреса памяти, в которую нужно выполнить запись, пару регистров ES:DI. Это означает, что регистр ES особенно полезен при использовании его в качестве целевого сегмента при копировании блоков, сравнении строк, просмотре памяти и очистке блоков памяти.

Регистр SS указывает на начало сегмента стека, которые представляет собой блок памяти объемом 64К, в котором находится стек. Все инструкции, которые неявно используют регистр SP (включая занесение в стек, извлечение из стека, вызовы и возвраты управления), работают с сегментом стека, так как только регистр SP может использоваться для адресации памяти в сегменте стека. Как мы обсуждали ранее, регистр BP также работает относительно сегмента стека. Это позволяет использовать регистр BP для доступа к параметрам и переменным, которые хранятся в стеке.

Итак, чтобы на практике просмотреть все выше описанные примеры использования регистров и средств управления оперативной памятью, наиболее удобно воспользоваться программой-отладчиком DEBUG.

1.2.7 Программа-отладчик DEBUG

Программа-отладчик DEBUG является очень важным и необходимым инструментом для изучения работы ЭВМ, поставляемым в составе DOS. Программа DEBUG позволяет осуществлять три вида действий в отношении содержимого ПЗУ и ОЗУ:

1) выборку произвольного участка памяти и отображение его содержимого в двух форматах:
   - шестнадцатеричный / ASCII;
   - формат не связанный с деассемблированием;

2) запись программ на машинном языке или на языке ассемблера и их выполнение;

3) исследование и отладку программ, хранящихся на диске или в памяти.

Программы, представленные на языке машинных команд (например, исполнительный модуль), чрезвычайно сложны для восприятия человеком. Процесс деассемблирования, реализуемый программой DEBUG, значительно облегчает процедуру интерпретации машинного языка.

Деассемблирование - это процесс трансляции или преобразования инструкций машинного языка, представленных в абсолютном шестнадцатеричном виде в символическую нотацию языка ассемблера. Так, например, на языке ассемблера можно записать *INC AX* (увеличить содержимое регистра AX на единицу); ассемблер переведет эту конструкцию в команду на машинном языке с кодом 40/16. Функция деассемблера состоит в том, чтобы привести команду машинного языка с кодом 40/16 обратно к виду *INC AX*. Несмотря на то, что деассемблер может выполнить преобразование команд машинного языка к более удобным командам языка ассемблера, целый ряд

весьма существенных компонентов программы, написанный на языке ассемблера реконструкции не поддается. Так, очевидно, не могут быть восстановлены весьма полезные комментарии программиста. Кроме того, не представляется возможным восстановить оригинальные символические имена адресов памяти. Например, вместо оригинальной авторской конструкции типа *JMP FINISH* (перейти к завершающей процедуре) деассемблер сгенерирует строку вида *JMP OE6C*. Пользователь видит перед собой команду перехода, однако ее смысл остается для него неясным.

Подобно остальным командам, эта программа DEBUG запускается путем набора на клавиатуре ее имени:

```
C:\ >  DEBUG
```

В процессе ее выполнения запросам на ввод исполнительных операторов предшествуют лишь "тире". Рассмотрим выполнение команд программой - отладчиком DEBUG на примере поиска в памяти адреса символьной строки.

В качестве образца объекта поиска выберем сообщение об ошибке, выдаваемое MS DOS: "Файл не найден". На рисунке 1.6 показана процедура запуска программы DEBUG и оператор поиска указанного выше сообщения. Предположим, для определенности, что в качестве начального адреса просмотра (поиска) программе DEBUG был передан адрес параграфа 0100, а также размер (длина) просматриваемого участка (L) равный 65535 байтам (`FFFF в шестнадцатеричной системе счисления). Программа DEBUG сообщает, что интересующий нас текстовой объект найден.

```
C:\ >  DEBUG
 -  S 0100:0000  L  FFFF  "Файл не найден"
 0100:0FF9
```

Рисунок 1.6- Запуск программы DEBUG и поиск сообщения

Это сообщение формируется в форме сегментированного адреса 0100:0FF9, означающего, что относительно сегмента с адресом 0100 берется шестнадцатеричное смещение 0FF9. Выполнив действия над адресами, мы получим полный 20-байтовый физический адрес ячейки памяти:

```
  0100   <адрес сегмента>
+ 0FF9  <смещение>
 -------
  01FF9
```

Если в рамках программы DEBUG использовать команду "D", указав при этом начальный адрес и количество просматриваемых байт (например, -D 0100:0100 L 100), то на экране будет отображено содержимое указанных ячеек памяти. Одна из этих особенностей состоит в том, что в левую часть поля

вывода помещается шестнадцатеричная информация, а в правую - информация в коде ASCII. Вторая особенность заключается в том, что кодовая комбинация, не имеющая символьного представления в коде ASCII, изображается в правой части поля вывода с помощью точки.

Команда "U" программы DEBUG - означает "деассемблирование" - осуществляет преобразование произвольных кодов памяти в мнемонические коды языка ассемблера. Деассемблер не делает две вещи.

Первое. Деассемблер не интерпретирует смысл программы и не обучает пользователя. Для понимания листинга, выдаваемого деассемблером, необходимо знать язык ассемблера.

Второе, что не под силу программе деассемблера - это установка так называемой абсолютной синхронизации. Известно, что команды машинного языка для микропроцессора INTEL 8086/8088 имеют переменную длину - от одного до шести байтов. После того как деассемблеру сообщена конкретная позиция памяти он приступает к процедуре прямого декодирования, не отличая кодов команд от данных. Достаточно ошибиться в выборе исходной позиции памяти (например, попасть не на границу между командами или в область данных) и результат окажется неверным. Если начальная точка набора команд известна, то никаких проблем не возникает.

Таблица 1.1- Перечень команд программы-отладчика DEBUG

Наименование команды	Назначение	Формат
1	2	3
Assemble	Ввод исполнительной программы на языке ассемблера	A [адрес]
Compare	Сравнение кодов по указанным адресам	C <диапазон><адрес>
Dump	Отображение содержимого ячеек указанного участка памяти	D [диапазон]
Enter	Ввод программы пользователя на машинном языке	E <адрес> [список]
Fill	Размещение в памяти указанного списка байт	F <диапазон>< список>
Go	Пуск программы с указанного адреса или по умолчанию с текущего	G [=адрес] [адреса]
Hex	Арифметические действия с шестнадцатеричными числами	H <значение1> <значение2>
Input	Вывод в порт	I <порт>
Load	Загрузка в память данных или файла по указанному адресу с диска	L [адрес] [диск] [первый_сектор] [число]
Move	Перемещение указанного диапазона участка памяти по адресу	M <диапазон><адрес>

Продолжение таблицы 1.1

1	2	3
Name	Указание имени файла	N [путь] [список_аргументов]
Output	Вывод в порт	O <порт>< байт>
Proceed	Переход по указанному адресу в программе выполнения	P [=адрес] [число]
Quit	Выход из программы DEBUG	Q
Register	Отображение содержимого регистров	R [регистр]
Search	Поиск указанной информации	S диапазон список
Trace	Выполнение пользовательской программы по шагам	T [=адрес] [значение]
Unassemble	Деассемблирование	U [диапазон]
Write	Запиь на диск	W [адрес] [диск] [первый_сектор] [число]
	Выделение памяти EMS	XA [число страниц]
	Освобождение памяти EMS	XD [дескриптор]
	Сопоставление страниц EMS	XM [L_страница] [P_страница] [дескриптор]
	Вывод состояния памяти EMS	XS

В таблицу 1.1 сведены основные команды программы - отладчика DEBUG.

При использовании таблицы следует учитывать следующее:

- квадратные скобки означают необязательные конструкции, в данном случае параметры, которые могут устанавливаться по умолчанию;
- параметры в угловых скобках указывать обязательно;
- все диапазоны и адреса задаются в шестнадцатеричном виде с указанием сегментного адреса (если необходимо) и смещения.

1.3 Задание на выполнение лабораторной работы

1.3.1 Познакомьтесь с функциями и форматом следующих команд отладчика: дамп (d), ввод (e), шестнадцатеричный (h), выход (q), регистры (r), трассировка(t).

1.3.2 Загрузите DEBUG. После приглашения к диалогу с помощью команды d определите:

а) размер памяти (ячейки 413h и 414h)

-d 0040:0013 <enter>

б) серийный номер

-d fe00:0 <enter>

в) дату ROM BIOS в формате mm/dd/yy по адресу FFFF5

Данные запишите в отчёт.

1.3.3 Наберите программу в машинных кодах (см. таблицу 1.2) с шага CS:100, набирая побайтно через пробел:

-e cs:100 b8 23 …….. cb <enter>

Выполните программу по шагам, используя команды отладчика R и T. При этом наблюдайте как изменяется содержимое регистров микропроцессора:

-r <enter>

-t <enter>

.

.

. до команды RETF

Таблица 1.2 - Пример программы в машинных кодах

Машинный код Команды	Мнемокод (ассемблер)	Назначение Команды
B82301		
052500		
8bd8		
03d8		
8bcb		
90		
Cb		

Заполните графы таблицы 1.2 "мнемокод" и "назначение".

1.3.4 Исследуйте команду отладчика H. Возьмите несколько значений (<значение1> и <значение2>) и объясните результаты.

1.3.5 Изучите команду отладчика A. Выпишите в отчёт все фрагменты ассемблерных программ пп. 1.2.4 и 1.2.6 методического указания к данной лабораторной работе, введите их в DEBUG, выполните по шагам, наблюдая за изменением всех компонент МП, используемых в данном фрагменте, и напишите комментарий к каждой команде по следующему образцу:

-A cs:100 <enter>

1D60:0100 mov ax,5

1D60:0103 mov dx,9

1D60:0106 add ax,dx

1D60:0108

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D60 ES=1D60 SS=1D60 CS=1D60 IP=0100   NV UP EI PL NZ NA PO NC
1D60:0100 B80500        MOV     AX,0005
-t
AX=0005 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D60 ES=1D60 SS=1D60 CS=1D60 IP=0103   NV UP EI PL NZ NA PO NC
1D60:0103 BA0900        MOV     DX,0009
-t
AX=0005 BX=0000 CX=0000 DX=0009 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1D60 ES=1D60 SS=1D60 CS=1D60 IP=0106   NV UP EI PL NZ NA PO NC
1D60:0106 01D0          ADD     AX,DX
-t
AX=000E BX=0000 CX=0000 DX=0009 SP=FFEE BP=0000 SI=0000 DI=000
DS=1D60 ES=1D60 SS=1D60 CS=1D60 IP=0108   NV UP EI PL NZ NA PO NC
```

Запись в отчёте:

mov ax,5 ; загрузить в регистр AX число 5
mov dx,9 ; загрузить в регистр DX число 9
add ax,dx ; сложить содержимое AX и DX, результат=Eh и помещается в регистр AX, значения битов флагового регистра остаются неизменными

1.3.6 Команды U и G отладчика на примере ввода данных с клавиатуры. Введите A cs:100. Наберите программу:

```
100 mov ah,3f
102 mov bx,00
105 mov cx,0c
108 mov dx,10f
10b int 21
10d jmp 100
10f db  ' '
```

Нажмите ENTER. Выполните команду U 100,10f. Выполните программу до шага 10b и введите G 10D, введите данные и проверьте содержимое с адреса 10f. Напишите комментарий к каждой строке.

1.3.7 Определите размер памяти, используя прерывание BIOS int 12h, для чего необходимо ввести команду "-e cs:100 cd 12 cb <enter>" и выполнить программу по шагам до команды IRET(в AX-размер памяти).

1.3.8 Определите адрес подпрограммы обслуживания вектора прерывания n (по указанию преподавателя).

1.3.9 Составьте программу на Ассемблере или на другом языке по указанию преподавателя, получите EXE-модуль и выполните его по шагам с помощью отладчика DEBUG.

1.4 Содержание отчета

1.4.1 Название лабораторной работы.

1.4.2 Цель работы.

1.4.3 Рисунки 1.1 и 1.2, а также таблица 1.1.

1.4.4 Результаты выполнения пунктов 1.3.2, 1.3.3, 1.3.4 и 1.3.8 задания к лабораторной работе.

1.4.4 Фрагменты программ с комментариями пунктов 1.3.5…1.3.7 задания к лабораторной работе.

1.4.5 Листинг разработанной программы к пункту 1.3.9 задания к лабораторной работе.

1.5 Контрольные вопросы

1.5.1 В чём отличие понятий структуры и архитектуры системы (вычислительной, управляющей, микропроцессорной и т.п.)?

1.5.2 Какие основные компоненты входят в состав микропроцессорной системы?

1.5.3 Какие элементы микропроцессора являются определяющими при функционировании процессора?

1.5.4 Какая система счисления является машинным языком? В какой системе счисления обычно записывают команды на машинном языке? Почему?

1.5.5 Какие архитектурные характеристики микропроцессора представляют наибольший интерес при его программировании?

1.5.6 Является ли модель МП Intel 8086 моделью фон Неймана и почему?

1.5.7 К какой разновидности архитектур относится архитектура 8086: RISC, CISC, SIMD, SISD, MIMD?

1.5.8 Перечислите основные группы регистров, входящих в состав МП 8086.

1.5.9 При работе PC в реальном режиме используется сегментный способ адресации памяти. В чём суть? Поясните на примере с 16-битной шиной адреса.

1.5.10 Для чего предназначена программа-отладчик DEBUG?

1.5.11 Что такое стек и где он используется? Приведите пример использования стека.

1.5.12 Назовите основные признаки отличия RISC - архитектуры МП от CISC. Что такое конвейерная обработка данных и какой из архитектур МП она наиболее приемлема: RISC или CISC? Почему?

1.5.13 Какая информация включается в состав слова состояния процессора?

1.5.14 Как организуется в микро-ЭВМ сегментно-страничный способ управления памятью. Почему этот режим называется защищённым?

2 Требования языка Ассемблер. Ассемблирование и выполнение программы

2.1 Цель работы

Показать основные требования к программам на языке Ассемблер и этапы ассемблирования, компоновки и выполнения программы.

2.2 Общие положения

2.2.1 Комментарии в программах на Ассемблере

Комментарий всегда начинается на любой строке исходного модуля с символа ";" , и ассемблер полагает, что все символы, находящиеся справа от ";" , являются комментарием. Комментарий может занимать всю строку или следовать за командой на той же строке:

1. ; Эта строка является комментарием
2. ADD BX,AX ; Комментарий с командой

Комментарии появляются только в листингах ассемблирования исходного модуля и не приводят к генерации машинных кодов.

2.2.2 Формат кодирования команд

Формат кодирования команд Ассемблера имеет следующий вид:

[метка] команда [операнды]

Метка ,команда и операнд разделятся по крайней мере одним пробелом. Максимальная длина строки - 132 символа. Примеры кодирования:

```
Метка    Команда   Операнд
count      db          1       ;имя, команда, один операнд
           mov         ax,0    ;команда, два операнда
```

Максимальная длина метки - 31 символ. Метка может содержать буквы, цифры и специальные символы и начинается с буквы или специального символа. Ассемблер не делает различия между заглавными и строчными буквами.

Мнемоническая команда указывает ассемблеру, какое действие должен выполнить данный оператор. В сегменте данных команда определяет поле, рабочую область или константу, а в сегменте кода - действие.

Операнд определяет начальное значение данных или элементы, над которыми выполняется действие по команде.

2.2.3 Директивы

Директивы (псевдооператоры) действуют только в процессе ассемблирования и не генерируют машинных кодов.

Директива **PAGE** указывает количество строк , распечатываемых на странице:

page 60,132

Директива **TITLE** указывает на печать заголовка:

title текст

Директива **SEGMENT** указывает в каком сегменте располагается информация (данные, команды) и имеет следующий формат:

<имя> segment [параметры]
.
.
<имя> ends

К параметрам относятся выравнивание, объединение и класс, например:

<имя> SEGMENT PARA STACK 'Stack'

Директива **PROC** обозначает начало процедуры с заданным именем и имеет формат:

имя_процедуры PROC FAR
:
RET

Директива **ASSUME** служит для сообщения ассемблеру назначения каждого сегмента(данных, кода или стека) и имеет формат:

ASSUME SS: имя_стека, DS: имя_данных, CS: имя_кода, ES:имя

2.2.4 Структура ассемблерной программы

1. Начало: директивы **PAGE** и **TITLE**
2. Объявление сегмента стека: директивы **SEGMENT** и **ENDS**
3. Объявление сегмента данных: директивы **SEGMENT** и **ENDS**
4. Начало кодового сегмента:

a) директива **SEGMENT**
b) директива **PROC (**главная процедура**)**
c) директива **ASSUME**

5. Настройка сегмента данных с помощью следующей последовательности команд:

MOV AX,<имя сегмента данных >
MOV DS,AX

6. Тело программы (набор команд и процедур)

7. Стандартное окончание:

- процедур:

```
                        RET
  <имя процедуры>   ENDP
```

- кодового сегмента:

```
                        MOV   AX,4C00
                        INT    21H
 <имя сегмента кода> ENDS
```

- программы:

```
                        END   <имя главной процедуры>
```

2.2.5 Методы адресации

При манипулировании регистрами, памятью и данными в МП используются различные методы адресации. Назовём пять основных:

- прямой;
- косвенный;
- непосредственный;
- по базе;
- с индексированием.

Следует отметить, что возможны варианты сочетания перечисленных методов адресации при выполнении ассемблера, например, косвенно-регистровый или метод адресации по базе с индексированием.

Таким образом, в любой команде мощно выделить три поля: выполняемое действие, адреса операндов или сами операнды, способ (метод) адресации. Выполнение команды начинается с фазы определения адресов операндов, их считывания и завершается циклом исполнения. Выполнение каждой фазы команды осуществляется за определённое число машинных тактов (один и более).

2.3 Задание на выполнение лабораторной работы

2.3.1 Создание исполнительного модуля

2.3.1.1 Набрать программу и сохранить в файле с именем exm.asm :

```
   page 60.132
title  exm (exe)
stacksg  segment  para  stack  'stack'
         db     12 dup  ('stackseg')        ;выделить под стек 128 байт
stacksg  ends
datasg  segment para 'data'
       adr1    db   ?                          ;выделить под данные 3 байта;
       adr2    dw   ?
datasg  ends
codesg   segment  para    'code'
 begin   proc     far              ;far-точка входа в процедуру, может быть near
 assume  ss:stacksg,cs:codesg,ds:datasg,es:nothing
        push ds
```

```
        sub ax,ax                       ;настройка
        push ax                             ;сегмента
        mov ax,datasg                         ;данных
        mov  ds,ax
        mov ax,0123h
        add ax,0025h
        mov bx,ax
        add bx,ax
        mov cx,ax
        sub cx,ax
        sub ax,ax
        nop
        ret
 begin   endp
codesg   ends
        end   begin
```

2.3.1.2 Отассемблировать при помощи команды "tasm.exe exm.asm" и получить объктный модуль exasm.obj.

2.3.1.3 Набрать команду "tlink.exe exm.obj"и получить исполнительный модуль exasm.exe.

2.3.1.4 Выполнить программу по шагам при помощи отладчика DEBUG. Команда "debug exm.exe".

2.3.1.5 Выписать назначение и функции команд и директив, встречающихся в программе.

2.3.2 *Изучение команд, использующих непосредственную адресацию*

Перед началом выполнения этого задания необходимо по электронному справочнику (программа ASS) ознакомиться с форматом и функциями арифметических, логических операций и операций сдвига.

2.3.2.1 Проанализировать программу, содержащую команды, использующие непосредственную адресацию, EXIMM и записать её в отчёт. К каждой строке написать комментарий (указать функцию оператора).

```
page 66,80
title eximm
stacksg segment para stack 'stack'
                dw      32 dup(?)       ;объем стека 64 байта
stacksg ends
datasg  segment para 'data'
      name1   db   100                ;сегмент данных
      name2   dw   626
datasg  ends
codesg segment para 'code'
```

```
        assume cs:codesg,ds:datasg,ss:stacksg,es:datasg
        begin  proc  far
        push ds              ;настройка
        sub  ax,ax           ;   сегмента
        push ax              ;     данных
        mov  ax,datasg
        mov  ds,ax
        mov es,ax
        cmp  al,ah
        adc  al,5
        add  bh,12
        sbb  al,5
        sub  name1,5
        rcl  bl,1
        rcr  ah,1
        rol  name2,1
        ror  al,1
        sal  cx,1
        sar  bx,1
        shr  name1,1
        and  al,00101100b
        or   bh,2ah
        test bl,7ah
        or   name1,23h
        ret
    begin  endp
    codesg ends
        end begin
```

2.3.2.2 Набрать и выполнить программу согласно пп.2.3.1.2…2.3.1.4.

2.3.2.3 Внести следующие изменения в программу:

- уменьшить в два раза сегмент стека;
- в сегмент данных внести строку символов 'КОНЕЦ','$' с адресом NAME3 и директивой DB;
- в конце программы записать команды вывода этой строки на экран по прерыванию INT 21H:

```
mov dx,offset name3      ;адрес вывода
mov ah,09                ;функция вывода DOS
int 21h                  ;вызов прерывания
mov ah,4ch               ;настройка DOS
int 21h                  ;вызов DOS
```

2.3.2.4 Проверить правильность выполнения программы(на экране должно появиться слово КОНЕЦ) .

2.3.3 *Использование процедур и команд организации цикла и ветвления, команды с косвенной адресацией*

Перед началом выполнения этого задания необходимо по электронному справочнику (программа ASS) ознакомиться с форматом и функциями команд LOOP, JMP, Jnnn, CALL, RET, PUCH и POP.

2.3.3.1 Проанализировать и выполнить программу расширенной пересылки данных EXMOVE:

```
   page 66,80
title exmove
stacksg segment para stack 'stack'
      dw     32 dup(?)
stacksg ends
datasg  segment para 'data'
      name1   db 'abcdefghi','$'
      name2   db 'jklmnopqr','$'
      name3   db 'stuvwxyz*','$'
datasg  ends
codesg segment para 'code'
     assume cs:codesg,ds:datasg,ss:stacksg,es:datasg
begin  proc  far      ;главная процедура
     push ds
     sub  ax,ax
     push ax
     mov  ax,datasg
     mov  ds,ax
     mov es,ax
     call b10move    ;вызов процедуры
     call c10move    ;вызов процедуры
     ret
begin  endp
b10move proc          ;начало процедуры
     lea si,name1
     lea di,name2
     mov cx,09            ;счётчик цикла
b20:   mov al,[si]       ; начало цикла
     mov [di],al
     inc si
     inc di
     dec cx
     jnz b20                ;переход на метку в20
     ret                    ;выход из процедуры
b10move endp
c10move proc
```

```
        lea si,name2
        lea di,name3
        mov cx,09           ;счётчик цикла
c20:    mov al,[si]     ; начало цикла
        mov [di],al
        inc di
        inc si
        loop c20            ;конец цикла
        ret
c10move endp
codesg ends
     end begin
```

2.3.3.2 В конце дополнить программу фрагментами вывода строк NAME2 и NAME3 и выполнить программу.

2.3.4 *Написать и выполнить программу по указанию преподавателя, используя приведённые примеры.*

2.4 Содержание отчёта

2.4.1 Название работы.
2.4.2 Цель работы.
2.4.3 Описание команд и директив, встречающихся в программах.
2.4.4 Алгоритм и текст программы контрольного задания.

2.5 Контрольные вопросы

2.5.1 При манипулировании регистрами, памятью и данными в МП используются различные методы адресации. Назовите пять основных.

2.5.2 Опишите алгоритм выполнения команды микропроцессором, т.е. обработку командных и информационных слов.

2.5.3 Дополните цепочку временных интервалов выполнения команд МП тактовая частота→……?….→ЦК.

2.5.4 В чём отличие между командой (оператором) и директивой языка ассемблера?

2.5.5 Какие функции выполняет программа tasm.exe?

2.5.6 Какие функции выполняет программа tlink.exe?

2.5.7 При объявлении процедур используются атрибуты **far** или **near.** Для чего?

2.5.8 Назовите и опишите команды языка ассемблера, используемые для организации циклов и передачи управления.

2.5.9 Опишите алгоритм выполнения вызова подпрограммы командой *CALL*. Какие регистры МП при этом задействованы?

3 Обработка данных в ASCII и BCD-форматах

3.1 Цель работы

Целью работы является изучение механизма преобразования ассемблером данных, предоставляемых пользователем в различных форматах, в машинные коды, рассмотрение ASCII- и BCD-форматов данных и приобретение навыков преобразования этих форматов в двоичный в ассемблерной программе.

3.2 Общие положения

3.2.1 Директивы определения данных

Ассемблер обеспечивает два способа определения данных: во-первых, через указание длины данных и, во-вторых, по их содержимому. Основной формат определения данных:

[имя] Dn выражение

Для определения элементов данных имеются следующие директивы: DB (байт), DW (слово), DD (двойное слово), DQ (учетверённое слово), DT (десять байт). Например, в приведенном ниже фрагменте программы

```
.DATA
N1  DB  25, 26, 37,112, …..
.CODE
:
MOV AL,N1+3
:
```

в регистр AL загружается значение 112 (10H).

Для определения повторяющихся данных можно использовать директиву DUP:

```
DW  10  DUP(?)   ; десять неопределённых слов
```

При указании в программе непосредственных данных (констант) необходимо указывать их формат: шестнадцатиричный (102H), десятичный (291) или двоичный (10101100B).

3.2.2 ASCII- и BCD-форматы данных

Данные, вводимые с клавиатуры, имеют ASCII-формат (американский международный код), т.е. каждый символ (буква, цифра, знак и т.д.) имеет длину в один байт и определенный двоичный код, занесённый в стандартную таблицу. Так, например, буква A имеет код 01000001(41H), а цифра 1 - 00110001(31H).

BCD-формат, или двоично-десятичный код, в основном используется некоторыми командами ассемблера для действий над числовыми данными. Так, например, число 56 имеет представление 01010110.

С помощью следующих ассемблерных команд можно выполнять арифметические операции непосредственно над числами в ASCII-формате:

AAA - коррекция для сложения;

AAD - коррекция для деления;

AAM - коррекция для умножения;

AAS - коррекция для вычитания.

3.2.3 Пример выполнения арифметической операции

В приведенном ниже примере суммируются два символьных числа (ASCII-код) по следующему алгоритму:

- адреса последних символов (8 и 6) слагаемых запоминаются в регистрах SI и DI,а суммы - в BX;
- производят их суммирование в аккумуляторе AL(команда ADC);
- при помощи команды AAA преобразовывают сумму в формат BCD и прибавляют к AL код 30h с целью получения ASCII-кода последнего символа суммы;
- после изменения адресов символов слагаемых (DEC SI и DEC DI) цикл повторяют(команда LOOP <метка>).

Пример 3.1 Составить программу сложения двузначных символьных чисел, используя информацию пункта 3.2.2 и описанный выше алгоритм.

```
title  summa(exe)
stsg  segment  stack 'stack'
        dw      32 dup (?)
stsg  ends
;-----------------------------------------------
datasg  segment  para 'data'
asc1    db   '78'
asc2    db   '96'
asc3    db   '000','$'
datasg  ends
;-----------------------------------------------
codesg  segment para 'code'
begin   proc near
assume  cs:codesg,ds:datasg,
ss:stsg,es:datasg
        push ds
sub  ax,ax
        push ax
        mov  ax,datasg
        mov  ds,ax
        mov  es,ax
        clc
        lea  si,asc1+1
```

```
        lea  di,asc2+1
        lea  bx,asc3+2
        mov  cx,02
        mov  ax,0000
a20:    mov  ah,00
        adc  al,[si]
        adc  al,[di]
        aaa
        or al,30h
        mov [bx],al
        mov al,ah
        dec si
        dec di
        dec bx
        loop a20
        or  ah,30h
        mov  [bx],ah
        mov  dx,offset asc3
        mov  ah,09
        int  21h
        mov ah,4ch
        int 21h
        ret
begin     endp
codesg  ends
          end begin
```

3.3 Задание на выполнение лабораторной работы

3.3.1 Выполнить и прокомментировать программу, приведённую в примере.

3.3.2 Изменить программу таким образом, чтобы она суммировала четырёхзначные числа.

3.3.3 Написать программу выполнения арифметической операции по одному из указанных ниже образцов (по указанию преподавателя):

Образец 1:

```
        data segment
        asci1 db "79+"
        asci2 db  "8="
        asci3 db  "000",13,10,'$'
        data  ends
        code segment
        main proc far
        assume ss:stk,cs:code,ds:data
```

```
mov ax,data
mov ds,ax
clc
.
. <арифметические и логические операции>
. <операция вывода>
.
mov ah,01h
int 21h
mov ax,4c00h
int 21h
main endp
code ends
end main
```

Образец 2:

```
data segment
asci1 db "9+"
asci2 db  "68="
asci3 db  "000",13,10,'$'
data  ends
:
```

Образец 3:

```
data segment
asci1 db "56/"
asci2 db  "8="
asci3 db  "0",13,10,'$'
data  ends
:
```

Образец 4:

```
data segment
asci1 db  "7*"
asci2 db  "8="
asci3 db  "00",13,10,'$'
data  ends
:
```

Образец 5:

```
data segment
asci1 db "56-"
asci2 db  "8="
asci3 db  "00",13,10,'$'
data  ends
:
```

Образец 6:

```
data segment
asci1 db "99+"
asci2 db  "8="
asci3 db  "000",13,10,'$'
data  ends
:
```

Образец 7:

```
data segment
asci1 db "56-"
asci2 db  "12="
asci3 db  "00",13,10,'$'
data  ends
:
```

Образец 8:

```
data segment
asci1 db  "75*"
asci2 db  "8="
asci3 db  "000",13,10,'$'
data  ends
:
```

3.3.4 Выполнить и прокомментировать программу преобразования прописных букв в строчные:

```
model small
.stack 100h
.data
simvol DB 'ABCDEFG',13,10,'$'
.code
mov ax,@data
mov ds,ax
lea bx, simvol
mov cx,7
b20: mov ah,[bx]
cmp ah,41h
jb b30
cmp ah,5ah
ja b30
or ah,00100000b
mov [bx],ah
b30: inc bx
loop b20
mov ah,9
mov dx,offset simvol
```

```
int 21h
mov ah,4ch
int 21h
end
```

3.3.5 Составить программу преобразования символьной строки abcdefgi в строку IGFEDCBA.

3.3.6 Составить программу преобразования двузначного символьного числа в двоичный формат и программу преобразования двоичного формата в ASCII-формат, используя программные фрагменты в примерах 3.2 и 3.3.

Пример 3.1 - Напишем программу преобразования символьного числа в двоичное число, полагая, что (cx) = 10, (si) = адрес символьной строки, по адресу "adr1" в сегменте данных находится преобразуемое число, по адресу "adr2" будет помещено двоичное число (результат), адреса " adr3" и " adr4" используются для хранения числа символов и множителя соответственно, а (bx) = число обрабатываемых символов.

```
      mov cx,10
      lea si,adr1-1
      mov bx,adr3
a10:  mov al,[si+bx]
      and ax,000fh
      mul ax,adr4
      add adr2,ax
      mov ax,adr4
      mul cx
      mov adr4,ax
      dec bx
      jnz a10
```

Пример 3.2 - Напишем программу преобразования двоичного числа в символьное число, полагая, что (cx) = 10, (si) = адрес символьной строки, (ax) = двоичное число.

```
b10:  cmp ax,0010
      jb  b20
      xor dx,dx
      div cx
      or dl,30h
      mov [si],dl
      dec si
      jmp b10
b20:  or al,30h
      mov [si],al
```

3.4 Содержание отчёта

Отчёт должен содержать описание алгоритмов, команд и директив, встречающихся в программах, включая название и цель работы, листинг контрольной программы.

3.5 Контрольные вопросы

3.5.1 Определить физический адрес (ячейку памяти) в оперативной памяти, с которого начинается символьная строка, используя текст программы и распечатку её выполнения в отладчике DEBUG:

```
dts segment 'data'
n1 db 3 dup(?)
n2 db 'строка символов','$'
dts ends
cds segment 'code'
beg proc far
mov ax,dts
mov ds,ax
lea dx,n2
mov ah,09
int 21h
ret
beg endp
cds ends
end beg

1DA8:0000 B8A61D        MOV     AX,1DA6
-t
AX=1DA6  BX=0000  CX=0072  DX=0000  SP=0000  BP=0000  SI=0000
DI=0000 DS=1D92  ES=1D92  SS=1DA2  CS=1DA8  IP=0003   NV UP EI PL NZ
NA PO NC   1DA8:0003 8ED8          MOV     DS,AX
-t
 :
-t
AX=09A6  BX=0000  CX=0072  DX=0003  SP=0000  BP=0000  SI=0000
DI=0000 DS=1DA6  ES=1D92  SS=1DA2  CS=1DA8  IP=000A   NV UP EI PL NZ
NA PO NC 1DA8:000A CD21          INT     21
-g cd23
строка символов
```

3.5.2 Каким образом представляется в ЭВМ текстовая информация?

3.5.3 Расшифруйте абревеатуру ASCII-формат.

3.5.4 Расшифруйте абревеатуру BCD-формат.

3.5.5 Опишите алгоритм преобразования символьного числа в двоичное и обратно.

4 Программные модели аппаратных средств вычислительных систем

4.1 Цель работы

Приобретение навыков программного моделирования логических и функциональных элементов электронно-вычислительной аппаратуры и микропроцессорных систем.

4.2 Краткие сведения из теории

4.2.1 Способы построения программных моделей

В общем случае аппаратные средства вычислительных микропроцессорных систем состоят из множества электронных компонентов: микропроцессора, оперативного запоминающего устройства, постоянного запоминающего устройства, регистров, логических схем и т. п. Несмотря на их разнообразие, все они могут быть сведены к совокупности логических элементов и элементов памяти (логического базиса).

Развитие средств микропроцессорной техники привело к тому, что применение готовых микроконтроллеров и микро-ЭВМ становится экономически выгодным в сравнении с проектированием специальных логических схем. При использовании микроконтроллера в качестве специализированной логической схемы на него возлагается новая задача - программное моделирование аппаратных средств.

Основные принципы замены аппаратных средств программными формулируются очень просто:

- программы могут заменить аппаратные средства, если эта замена удовлетворяет требованиям к быстродействию микропроцессорной системы и если эта замена экономически целесообразна;
- программы, заменяющие аппаратные средства, должны моделировать функции аппаратных средств, а именно: восприятие, хранение, обработку и выдачу цифровой информации.

В общем случае программная модель аппаратных средств содержит программу работы микро-ЭВМ и наборы значений входных переменных (сигналов), которые программа перерабатывает в наборы выходных сигналов.

Различают два способа построения программной модели: компиляционный и интерпретирующий.

При *компиляционном способе* программной реализации для каждого дискретного элемента вычислительной системы строится своя программа, которая для своего выполнения не требует никаких исходных данных, кроме задаваемого извне входного набора переменных.

При *интерпретирующем способе* программной реализации в памяти находится одна универсальная программа, которая настраивается на реализацию

заданного устройства с помощью некоторого заранее задаваемого массива исходных данных. Для реализации какой-либо другой функции изменяется не программа, а лишь массивы исходных данных.

Алгоритм программной реализации иллюстрирует рисунок 4.1. Значения заданной булевой функции на всех 2^n наборах значений своих переменных моделируемого элемента записываются в память микро-ЭВМ. Считываемое извне слово значений входных переменных суммируется с адресом начала описания и образует адрес в массиве, где записано значение реализуемой функции, которое и выдается из ВС. Модель представляется в памяти массивом пар слов, состоящих из слова следующего состояния и слова значений выходных переменных.

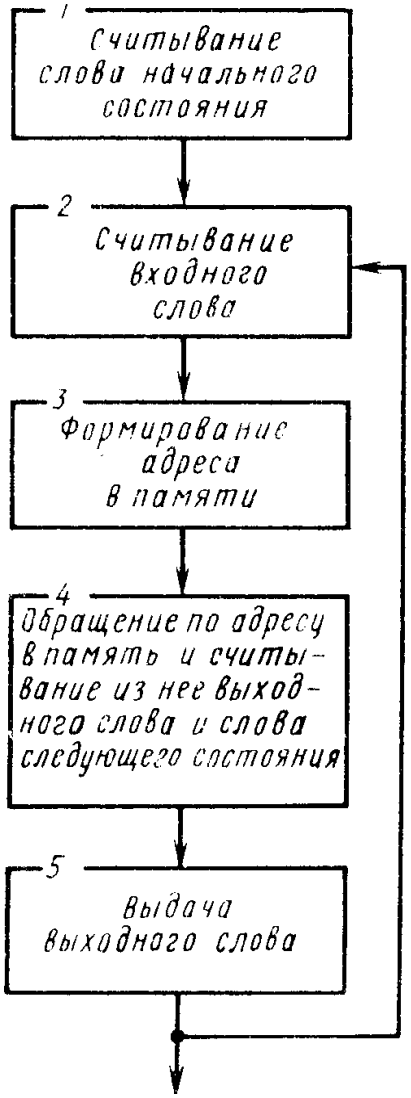


Рисунок 4.1- Схема программной реализации модели элемента ВС

4.2.2 Компиляционный метод программного моделирования

Алгоритмы моделирования комбинационных логических схем соответствуют описанию функций алгебры логики (ФАЛ) и основаны на использовании общего подхода к синтезу программных моделей, при котором одна про-

грамма обслуживает целый ряд моделей схем, формально представленных в виде таблиц истинности. Программы, реализующие ФАЛ, работают с упорядоченными массивами данных.

На практике часто бывает необходимо создавать программные модели для одной или небольшого числа логических схем. Если выражения для булевых функций содержат малое число термов, то наиболее целесообразен индивидуальный подход к моделированию каждой логической схемы. При этом работа схемы описывается как с помощью микрокоманд микропроцессора, так и на языках программирования Ассемблер и Паскаль для микро-ЭВМ семейства PC, и представляет собой самостоятельную подпрограмму или процедуру.

4.2.3 Моделирование элементов двухступенчатой логики

Рассмотрим программу, моделирующую работу логического элемента К155ЛР1, функциональная схема которого представлена на рисунке 4.2.

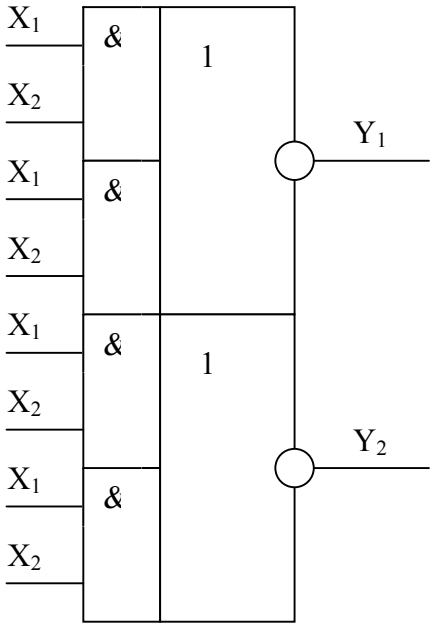


Рисунок 4.2 – Функциональная схема логического элемента

Для этого логического элемента можно получить следующие ФАЛ, связывающие входы и выходы:

$$Y1=\overline{X1/\backslash X2\backslash/X3/\backslash X4} \quad (1)$$

$$Y2=\overline{X5/\backslash X6\backslash/X7/\backslash X8} \quad (2)$$

ФАЛ будем моделировать на языке программирования Паскаль с помощью операторов *AND* и *OR*, сохраняя наименования переменных и используя выражения (1) и (2). Текст варианта программы представлен ниже в примере 4.1.

Пример 4.1 - Написать программу, модулирующую работу логической микросхемы К155ЛР1. Текст программы на языке Паскаль:

```
program K155LR1;
uses crt;
var    y,i:Integer;
       x:array[1..8] of integer;
       ch:char;
 begin
   repeat
   clrscr;
   writeln('        логическая микросхема  К155ЛР1           ');
   writeln('введите входные переменные X1..X2');
   y:=1;
     for i:=1 to 8 do begin
     write('X',y,'=');
     readln(X[y]);
     if (X[y]<>0) and (X[y]<>1) then
       begin
       writeln('переменные принимают значения только 1 или 0!');
       i:=i-1;    y:=y-1;
       end;
  y:=y+1;
  end;
  x[1]:=x[1] and x[2];
  x[3]:=x[3] and x[4];
  x[5]:=x[5] and x[6];
  x[7]:=x[7] and x[8];
  x[1]:=x[1] or x[3];
  x[1]:= not x[1];
  x[1]:=x[1]+2;
  x[5]:=x[5] or x[7];
  x[5]:= not x[5];
  x[5]:=x[5]+2;
  writeln     ('  _____________');
  writeln('Y1=X1&X2+X3&X4=',x[1]);
  writeln    ('  _____________');
  writeln('Y2=X5&X6+X7&X8=',x[5]);
  writeln;
  writeln('Esc-выход, Enter - заново');
ch:=readkey;
until ch=#27;
end.
```

4.2.4 Моделирование комбинационных логических схем

К комбинационным логическим схемам относятся преобразователи кодов, шифраторы, дешифраторы, мультиплексоры, устройства сдвига и другие элементы вычислительной техники. Работа устройств, реализующих комбинационные логические схемы, как правило, определяется таблицей истинности, устанавливающей зависимость между входными (входное слово) и выходными (выходное слово) сигналами. Пример такой таблицы для дешифратора 4 на 16 (например, К155ИДЗ) приведён ниже.

Таблица 4.1 - Таблица истинности для дешифратора К155ИДЗ

входные сигналы				выходные сигналы															
x_1	x_2	x_3	x_4	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}	y_{16}
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	0	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Пользуясь этой таблицей, составим программу на языке Ассемблера микро-ЭВМ семейства PC, моделирующую работу дешифратора. В этой модели также используется компиляционный метод программной реализации.

Алгоритм выполнения программы состоит в следующем. Входные переменные (исходные данные) *X1,X2,X3, X4* вводятся с клавиатуры в сегмент данных в поле INDAT1, преобразуются из символьного формата в двоичное число (поле INDAT2), а затем пересылаются в регистр CX.

Идея работы программы состоит в том, что в соответствии с кодом (N) регистра CX будет осуществляться N-кратный регистра AX. После операции сдвига содержимое регистра AX преобразуется к символьному виду и соответствующей процедурой выводится на экран монитора (см. пример 4.2).

Пример 4.2 - Написать программу моделирования дешифратора К155ИД3 на языке Ассемблера. Текст программы:

```
title  modell_id3
;---------------------------------------
stacksg segment para  stack  'stack'
      dw      32 dup (?)
stacksg  ends
;---------------------------------------
datasg  segment  para  'data'
indat1   db  '0000',10,13              ;входные данные
indat2   dw   0
outdat   db  '0000000000000000','$' ;выходные данные
datasg  ends
;---------------------------------------
codesg  segment  para  'code'
begin   proc  far
assume  cs:codesg,ds:datasg,ss:stacksg
      push  ds
      sub   ax,ax
      push   ax
      mov   ax,datasg         ;настройка сегмента
      mov   ds,ax                   ;данных
      call  vvod                ;ввод  данных
;----------------------------------------
      lea   si,indat1-1
      mov   bx,04           ;преобразование
      mov   cx,01             ;символьных
a10:                              ;данных
      mov   al,[si+bx]         ;(ASCII-код)
      and   ax,000fh             ;в
      mul   cx                     ;двоичное
      add   indat2,ax              ;число
      shl   cx,1
      dec   bx
      jnz   a10
      mov   cx,indat2         ;(cx)= число сдвигов
      mov   ax,0fffeh         ;(ax)= 1111111111111110
      rol   ax,cl               ;циклический сдвиг влево
; преобразование двоичного числа в символьное
;----------------------------------------
```

```
        mov   indat2,ax
        lea   si,outdat+15
        mov   cx,16
a20:
        and   ax,0001h
        or    al,30h
        mov   [si],al
        mov   ax,indat2
        shr  ax,1
        mov  indat2,ax
        dec  si
        loop a20
; конец преобразования двоичного числа в символьное
;-----------------------------------------
        lea  dx,outdat
        call wiw                ;вывод данных
        mov   ah,4ch
        int  21h
        ret
begin   endp
;   процедура  ввода
;-----------------------------------------
vvod    proc
        mov   ah,3fh
        mov   bx,00
        mov   cx,06
        lea   dx,indat1
        int   21h
        ret
vvod    endp
;   процедура вывода
;-----------------------------------------
wiw     proc
        mov  ah,09
        int  21h
        ret
wiw     endp
codesg  ends
        end  begin
```

Аналогично с программным моделированием комбинационных схем разрабатываются модели и последовательностных устройств, имеющих элементы памяти, т. е. регистров, счётчиков и т.п. Только в этом случае необходимо учитывать внутреннее состояние устройства и использовать соответствующие таблицы переходов.

4.3 Задания к лабораторной работе

4.3.1 Исследовать программы моделирования логических элементов.

4.3.2 По заданию преподавателя составить программные модели логических элементов (не менее двух) на языках Ассемблера и Паскаль.

4.3.3 Ввести программы моделирования в микро-ЭВМ и проверить их работу.

4.4 Содержание отчета

4.4.1 Цель работы и постановка задачи.

4.4.2 Функциональные схемы модулируемых устройств и их таблицы переходов.

4.4.3 Описание алгоритмов работы разрабатываемых программ.

4.5.2 Листинги программных моделей.

4.5 Контрольные вопросы

4.5.1 Работа логического элемента определяется таблицей истинности :

X	Y	F
0	0	0
1	0	1
0	1	1
1	1	0

Какую Булеву функцию (название) реализует данный элемент и как он обозначается на схеме?

4.5.2 В чём отличие функционирования комбинационных и последовательностных схем (элементов аппаратуры) ?

4.5.3 Где и с какой целью используются карты(диаграммы) Карно-Вейча? Приведите пример для двух переменных.

4.5.4 Напишите характеристическое уравнение одноразрядного полусумматора, полагая, что A и B - входные переменные, S - сумма и P - перенос.

4.5.5 Какие функции выполняет мультиплексор, демультиплексор?

4.5.6 Какие функции выполняет шифратор , дешифратор?

4.5.7 В чём отличие синхронного счётчика от асинхронного?

4.5.8 Как классифицируют регистры по способу записи, хранения и чтения информации?

4.5.9 Запишите уравнения, описывающие работу одноразрядного сумматора, и синтезируйте его схему на пороговом элементе с входными весами, равными +1,+1,+1,+2, и порогом срабатывания, равным+3, и на трёхвходовом мажоритарном элементе.

5 Система прерываний компьютера на примере экранных операций в BIOS и DOS

5.1 Цель работы

Ознакомиться с системой прерываний МП Intel 8086 и с требованиями для вывода информации на экран, а также для ввода данных с клавиатуры.

5.2 Основные положения

Схематичное взаимодействие видеосистемы и клавиатуры (консоли) с микропроцессором проиллюстрировано на рисунке 5.1 (буквами A и D обозначены шины адреса и данных, а сочетание M/IO означает магистраль ввода-вывода. Конкретная же реализация этого взаимодействия и конструктивные особенности всей системы зависят от архитектуры персонального компьютера, типа микропроцессора и использования конкретных компонентов (так называемых "чипсет" – наборов микросхем).

Так, например, в последних моделях компьютеров наряду с "обычной" архитектурой шин может использоваться архитектура с локальной шиной и шиной AGP, кроме того, в качестве устройств управления монитором применяется видеокарта (отдельная плата с процессором и локальной (буферной) памятью), а для реализации обмена информацией с оперативной памятью используется контроллер прямого доступа к памяти.

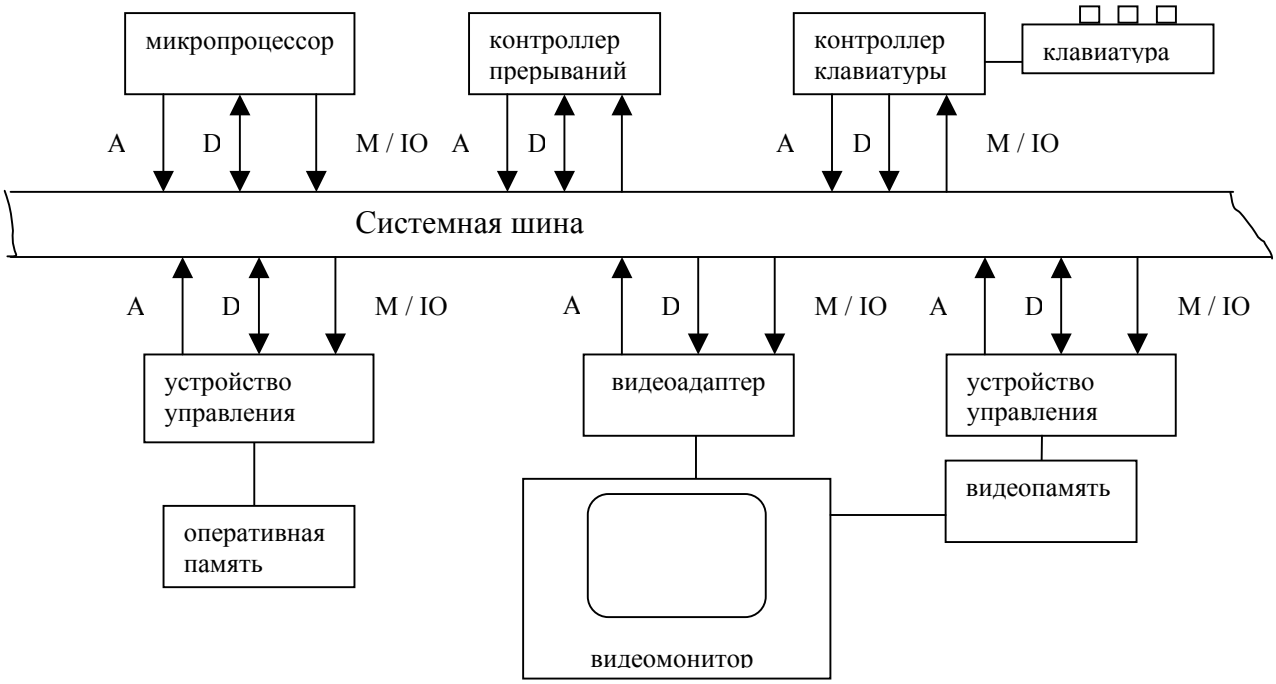


Рисунок 5.1 - Подключение устройств компьютера к системной шине

5.2.1 Система прерываний компьютера

Сигналы аппаратных прерываний, возникающих в устройствах, входящих в состав компьютера или подключенных к нему, поступают в процессор через два контроллера прерываний, один из которых является ведущим (рисунок 5.2).

К входным выводам IRQ1…IRQ15 (IRQ-Interrupt Request, запрос прерывания) подключаются выводы устройств, на которых возникают сигналы прерываний. При этом контроллеры передают в МП по линиям данных номер вектора, который образуется путём сложения базового номера контроллера с номером входной линии с запросом прерывания.

Процессор, получив сигнал прерывания, выполняет последовательность стандартных действий, называемых процедурой прерывания. Объекты вычислительной системы, принимающие участие в процедуре прерывания, и их взаимодействия показаны на рисунке 5.2.

Самое начало оперативной памяти от адреса 0000h до 03FFh отводится под векторы прерываний - четырёхбайтовые области, в которых хранятся адреса обработчиков прерываний (ОбрПР на рисунке 5.2). Всего в выделенной области памяти помещается 256 векторов.

Получив сигнал на выполнение процедуры прерываний процессор сохраняет в стеке содержимое трёх регистров: регистра флагов, CS и IP. Далее процессор загружает CS и IP из соответствующих векторов прерываний, осуществляя переход на программу - обработчик прерываний, связанную с этим вектором. Обработчик прерываний всегда оканчивается командой IRET (возврат из прерывания), после чего происходит возврат в основную программу в ту самую точку, где она была прервана.

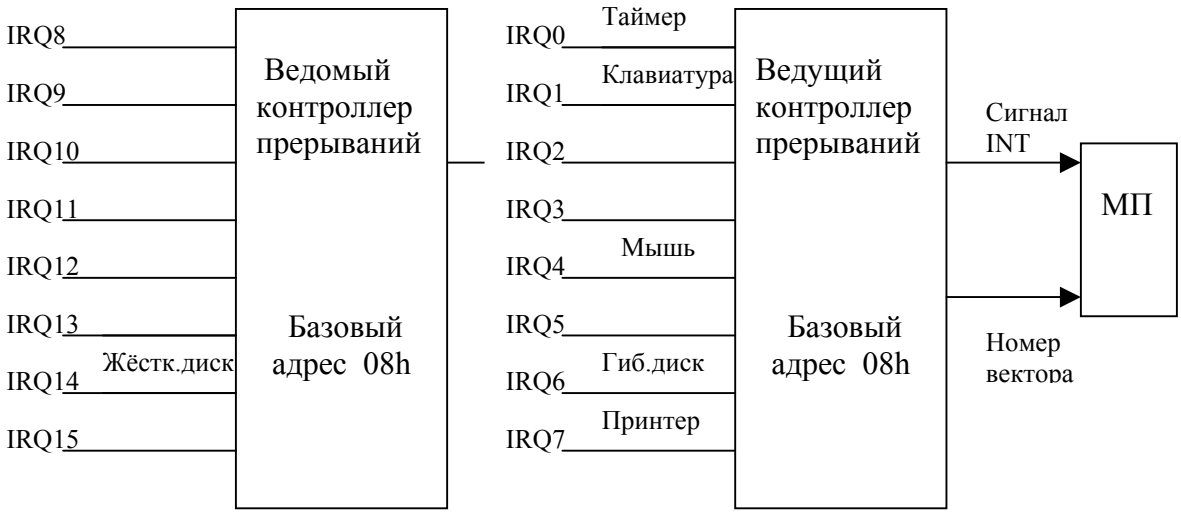


Рисунок 5.2- Аппаратная организация прерываний

Помимо описанных выше аппаратных прерываний от периферийных устройств, называемых внешними, имеются ещё два типа прерываний: внутренние и программные.

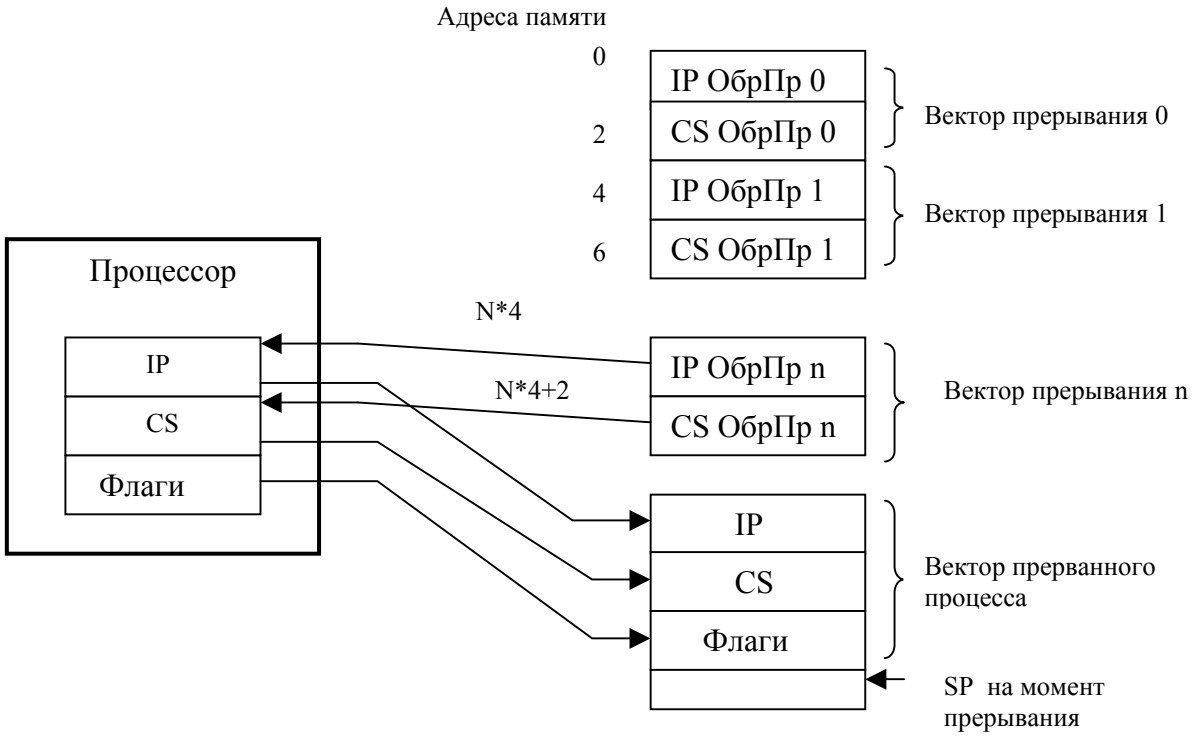


Рисунок 5.3 - Процедура обслуживания прерывания

Внутренние прерывания возбуждаются самим процессором при возникновении одной из специальных ситуаций, например, деление на ноль (вектор 0) или при попытке выполнить неправильную команду (вектор 6).

Программные прерывания вызываются командой INT с числовым аргументом, рассматриваемым процессором как номер вектора прерывания. Например, команда INT 10H вызывает системные обслуживающие программы BIOS монитора.

5.2.2 Примеры использования вектора прерывания 10h

Прерывание INT 10H обеспечивает управление всем экраном. В регистре AH устанавливается код, определяющий функцию прерывания. Ниже описаны все возможные функции и приведены примеры их использования.

1) INT10_SET_MODE = 000h ; set video mode (установка режима).

Порядок установки режима:

- предварительно необходимо выбрать режим согласно таблице 5.1;
- установить выбранный режим:

```
mov  ah,00    ; установка режима
mov  al,03    ; стандартный 16-цветовой текстовый
int   21h
```

Для определения типа адаптера используйте прерывание 11H. Данная команда возвращает в регистре AX значение, в котором 5 и 4 биты указывают на видеорежим: 10 - 80*25, черно-белый режим в цветном адаптере; 11 - 80*25, черно-белый режим в черно-белом адаптере.

Таблица 5.1 - Режимы для видеоадаптеров

AL	Размер экрана (в символах для текстового режима и в пикселях - для графического)	Режимы для видеоадаптеров
02	80*25	Черно-белый текстовый
03	80*25	Стандартный 16-цветовой текстовый
05	320*200	Черно-белый графический
06	640*200	Черно-белый графический
0D	320*200	16-цветовой графический
0E	640*200	16-цветовой графический
10	640*350	64-цветовой графический

2) INT10_SET_CURSOR_SHAPE = 001h ; set cursor shape (установка размера курсора).

Установка курсора в его нормальном виде:

```
mov ah,01
mov ch,06   ; минимум = 00
mov cl,07   ; максимум =13
int   21h
```

3) INT10_WRITE_ATTR_CHAR = 009h ; write attribute/char (вывод атрибута (BL)/символа(AL) в текущую позиции курсора; BH-страница, CX-число повторений).

Программа на языке Ассемблера использования описанной функции BIOS:

```
codesg segment para 'code'
  assume cs:codesg
    mov ah,09
    mov al,'*'
    mov bh,00
    mov bl,01h
    mov cx,05
    int 10h
    mov ah,01h
    int 21h
    mov ax,4C00h
    int 21h
codesg endsend
```

4) INT10_SET_COLOR_PALETTE = 00Bh ; set color palette (установка цветовой палитры; BH-идентификатор цвета палитры(0 или 1), BL-палитра (цвет))

Программа выбора графического режима "320*200 - 16-цветовой графический" и установки палитры с зелёным фоном:

```
; выбор режима
   mov ah,00
   mov al,0dh
   mov bh,00
   int 10h
   ;-------------------------
   ; установка палитры
   mov ah,0bh
   mov bl,02h
   int 10h
```

5) INT10_WRITE_PIXEL = 00Ch ; write graphics pixel (вывод точки на экран; AL-цвет, CX-горизонтальная координата, DX-вертикальная)

Фрагмент программы вывода разноцветных точек на экран (20 строк по 320 столбцов):

```
      mov bx,00
      mov cx,00
      mov dx,00
a1:   mov ah,0ch
      mov al,bl
      int 10h
      inc cx
      cmp cx,320
      jne a1
      mov cx,00
      inc bl
      inc dx
      cmp dx,20
      jne a1
```

5.2.3 Системные вызовы BIOS и DOS экранных операций

Экранные операции в BIOS вызываются по вектору прерываний 10h - BIOS Int 10h Video Service interrupt (см. таблицу 5.2).

Таблица 5.2 - Системные вызовы BIOS

Наименование функции	Значение функции	Выполняемая Операция
1	2	3
INT10_SET_MODE	000h	set video mode (установка видеорежима(AL))

Продолжение таблицы 5.2

1	2	3
INT10_SET_CURSOR_SHAPE	001h	set cursor shape (установка размера курсора (обычный: CH-верхняя, CL - нижняя граница))
INT10_SET_CURSOR_POS	002h	set cursor position (установка позиции курсора; BH - страница, DH-строка, DL- столбец)
INT10_READ_CURSOR	003h	get cursor position service (чтение текущего положения курсора(DX); CX- размер)
INT10_READ_LIGHT_PEN	004h	read light pen position (чтение текущего положения светового пера)
INT10_SEL_DISPLAY_PAGE	005h	select display page (выбор активной страницы (AL))
INT10_SCROLL_UP	006h	scroll window up (прокрутка экрана вверх; AL-количество строк, BH-атрибут, CX, DX-координаты)
INT10_SCROLL_DOWN	007h	scroll window down (прокрутка экрана вниз)
INT10_READ_ATTR_CHAR	008h	read attribute/char (чтение атрибута (AH) / символа(AL) в текущей позиции курсора)
INT10_WRITE_ATTR_CHAR	009h	write attribute/char (вывод атрибута (BL) /символа (AL) в текущую позиции курсора; BH-страница, CX-число повторений)
INT10_WRITE_CHAR	00Ah	write character (вывод символа(AL) в текущую позиции курсора; BH-страница, CX-число повторений)
INT10_SET_COLOR_PALETTE	00Bh	set color palette (установка цвето вой палитры; BH-режим(фон), BL-палитра (цвет))
INT10_WRITE_PIXEL	00Ch	write graphics pixel (вывод точки на экран; AL-цвет, CX - горизонтальная координата, DX-вертикальная)
INT10_READ_PIXEL	00Dh	read graphics pixel (чтение цвета точки с экрана)
INT10_WRITE_TTY	00Eh	write text in tty mode (вывод символа (AL) в режиме телетайпа; BH-страница)

Продолжение таблицы 5.2

1	2	3
INT10_GET_MODE	00Fh	get video display mode (получение текущего видеорежима(AL); AH-число символов в строке; BH-страница)
INT10_SET_PALETTE_REGS	010h	set palette registers
INT10_FONT_SIZE	011h	determine the # of rows
INT10_WRITE_STRING	013h	write string (вывод символьной строки)
INT10_GET_VIDEO_BUFFER	0FEh	get video buffer (получение видео-буфера)
INT10_UPDATE_VIDEO_ BUFFER	0FFh	update video buffer (изменение ви-деобуфера)

Экранные операции в DOS вызываются по вектору прерываний 021h - Dos Function call (см. таблицу 5.3).

Таблица 5.3 - Системные вызовы DOS

Наименование функции	Значе-ние функции	Выполняемая Операция
DOS_WRITE_STRING	009h	display a '$' terminated string (вывод символьной строки; DS:DX- адрес)
DOS_WRITE_TO_HANDLE	040h	write to File Handle

5.2.4 Ввод-вывод символов

Все необходимые экранные и клавиатурные операции можно выполнить с помощью команды INT 10H (прерывание), которая передает управление в BIOS. Для выполнения некоторых более сложных операций существует прерывание более высокого уровня INT 21H,которое сначала передаёт управление в DOS для выполнения дополнительных вычислений. Команда INT выполняет следующее:

- уменьшает указатель стека на 2 и заносит в стек содержимое флагового регистра;
- очищает флаги TF и IF;
- заносит в стек содержимое регистров CS и IP;
- обеспечивает выполнение необходимых действий и восстанавливает из стека значения регистров, возвращает управление в прерванную программу на команду, следующую за INT.

В таблице 5.4 приведены примеры положений курсора на экране для обычного видиомонитора, имеющего 25 строк и 80 столбцов.

Таблица 5.4 - Положение курсора

Положение	строка	столбец
Верхний левый угол	00	00
Верхний правый угол	00	4F
Центр экрана	0C	27/28
Нижний левый угол	18	00
Нижний правый угол	18	4F

Пример 5.1 - Подпрограмма установки курсора:

```
mov   ah,02      ;запрос на установку курсора
mov   bh,00      ;экран 0
mov   dx,050ch ;строка 05,столбец 12
int   10h
```

Пример 5.2 - Подпрограмма очистки экрана:

```
mov   ax,0600h  ;прокрутка(06) на весь экран(00)
mov   bh,07       ;атрибут черно/белый
mov   cx,0000   ;верхняя левая позиция
mov   dx,184fh  ;нижняя правая позиция
int   10h
```

При использовании прерывания DOS INT 21H в регистр AH заносится либо функция вывода на экран 09, либо функция ввода с клавиатуры 0AH,а в регистр DX - требуемый адрес.

5.3 Задание на выполнение лабораторной работы

5.3.1 Напишите программу на языке Ассемблера по указанию преподавателя. Не забывайте, что программа должна иметь следующую структуру:

```
title primer_con_gr
codesg segment para 'code'
assume cs:codesg
begin proc far
   .                ;сохранение в стеке используемых регистров
   .
   .                ;тело программы
   mov ah,01h   ;останов до нажатия клавиши
   int 21h
   mov ah,00      ;  восстановление текстового режима
```

```
    mov al,03
    int 10h
    .                   ; восстановление из стека
    .                   ; сохранённых регистров
    mov ax,4C00h    ; выход в DOS
    int 21h
    ret
begin endp
codesg ends
end begin
```

5.3.3 Напишите программу ввода-вывода вашего имени, используя примеры 5.1, 5.2, 5.3.

Пример 5.3 - Фрагмент программы ввода-вывода символьной строки:

```
:
datasg segment para 'data'
namepar label byte          ;имя списка параметров,
maxlen   db  20             ;макс. длина имени,
namelen  db  ?
namefld  db  20  dup(' '),'$'  ;имя
prompt   db  'name?','$'
datasg   ends
;-------------------------------------------------------
codesg segment para 'code'
begin  proc   far
assume cs:codesg,ds:datasg,ss:stacksg
      push  ds
      sub   ax,ax
      push  ax
      mov   ax,datasg
      mov   ds,ax
      mov   es,ax
      call  q10clr       ; вызов процедуры очиститки экрана (пример 5.2)
a20: mov   dx,0000
      call  q20curs      ; вызов процедуры установки курсора (пример 5.1)
      call  b10prmp
      call  d10inpt
call  q10clr
      cmp   namelen,00
      je    a30
      call  e10code
      call  f10cent
      jmp   a20
a30:  ret
;-----------------------------------------------
b10prmp  proc  near             ;  вывод текста запроса
```

```
        mov   ah,09
        lea   dx,prompt
        int   21h
        ret
    b10prmp endp
    begin   endp
    ;-----------------------------------------------
    d10inpt  proc  near                ; ввод имени с клавиатуры
        mov   ah,0ah
        lea   dx,namepar
        int   21h
        ret
    d10inpt  endp
    ;-----------------------------------------------
    e10code  proc  near                 ;  установка сигнала и ограничителя
        mov   bh,00
        mov   bl,namelen
        mov   namefld[bx],07
        mov   namefld[bx+1],'$'
        ret
    e10code  endp
    ;-----------------------------------------------
    f10cent  proc  near                 ;  центрирование и вывод имени
        mov   dl,namelen
        shr   dl,1
        neg   dl
        add   dl,40
        mov   dh,12
        call  q20curs
        mov   ah,09
        lea   dx,namefld
        int   21h
        ret
    f10cent  endp
    codesg   ends
    end begin
```

5.3.4 Измените программу таким образом, чтобы она помещала ваше имя в левом нижнем, правом верхнем и правом нижнем углу.

5.4 Содержание отчёта

5.4.1 Название работы и цель работы.
5.4.2 Описание команд и директив, встречающихся в программах.
5.4.3 Листинги самостоятельно разработанных программ.

5.5 Контрольные вопросы

5.5.1 Какие действия выполняют команды, вызывающие программные прерывания?

5.5.2 Что такое Interrupt и IRQ (Interrupt Request)? Сколько линий используется для IRQ?

5.5.3 Оперативная память имеет пять основных логических областей: -CM (Conventional Memory);-EMS;-UMA;-HMA;-XMS. Поясните.

5.5.3 Что такое кэш-память? Для чего она используется?

5.5.4 Для чего необходим прямой доступ к памяти?

5.5.5 Назовите основные категории прерываний в ЭВМ.

5.5.6 Какое место в иерархии прерываний занимают прерывания от внешних устройств?

5.5.7 С какой целью в ЭВМ реализован режим прерываний?

5.5.8 Перечислите критерии классификации периферийных устройств?

5.5.9 Чем различаются растровая, матричная и векторная развёртки и какими техническими средствами они реализуются?

Список использованных источников

1 Пятибратов А.П. и др. Вычислительные системы, сети и телекоммуникации: Учеб. – М.: ФиС, 2002. - 512 с.

2 Вычислительные машины и системы: Учеб. / Под редакцией В.Д.Ефремова, В.Ф.Мелехина – М.: Высш. школа,1993. – 400 с.

3 Колесниченко О.В. Аппаратные средства PC.- СПб.: БХВ – Петербург,2000. – 1024 с.

4 Фрир Д. Построение вычислительных систем на базе микропроцессоров. –М.: Мир, 1990 – 413с.

5 Ан П. Сопряжение ПК с внешними устройствами. – М.: ДМК Пресс, 2001. – 320 с.

6 Абель П. Язык Ассемблера для IBM PC и программирования. – М.: Высш. шк., 1992. – 447 с.

7 Нортон П. Программно-аппаратная реализация компьютера IBM PC. – М.:, Изд. Айсберг, 1992. – 587 с.

8 Новиков Ю. Персональные компьютеры: аппаратура, системы, Интернет. Учебный курс. – СПб.: Питер, 2001. – 464 с.

9 Бродин В.Б., Шагурин И. И. Микропроцессор i486. Архитектура, программирование, интерфейс. – М.: Диалог-МИФИ, 1993.