ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

С.Г. Михальченко

АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ

Руководство к организации самостоятельной работы

ТОМСК — 2007

Федеральное агентство по образованию

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра промышленной электроники

С.Г. Михальченко

АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ

Руководство к организации самостоятельной работы

2007

Михальченко С.Г.
Аппаратное и программное обеспечение ЭВМ: Руководство к организации самостоятельной работы. — Томск: Томский государственный университет систем управления и радиоэлектроники, 2007. — 103 с.

ОГЛАВЛЕНИЕ

1 ВВЕДЕНИЕ

Курс «Аппаратное и программное обеспечение ЭВМ» (АПО ЭВМ) совместно с курсами «Информатика» и «Операционные системы» составляет основу подготовки инженеров специальности «Промышленная электроника» и играет роль базы, без которой невозможна успешная деятельность инженера в области компьютерной техники и технологий.

Целью настоящих методических указаний является помощь студентам при изучении курса АПО ЭВМ, практическое закрепление знаний по структурам микропроцессоров семейства Intel, приобретение навыков разработки и наладки автоматизированных систем на основе современных компьютерных технологий.

Основными задачами данного учебно-методического пособия являются:

1. Оказание помощи студентам очной формы обучения в более глубоком изучении аппаратного и программного обеспечения ЭВМ.
2. Помощь студентам при изучении структуры и программных моделей микропроцессоров семейства Intel, на которых базируются современные персональные и промышленные компьютеры.
3. Оказание помощи студентам в овладении приемами и методами программирования компьютерных систем и их составляющих частей как на низком (Assembler), так и на высоком (Turbo Pascal) уровнях.
4. Помощь в освоении методов, устройств, алгоритмов, интерфейсов и протоколов, позволяющих реализовать программный диалог персонального компьютера и периферийных устройств (или технологических процессов) находящихся под управлением компьютера.
5. Формирование навыков разработки и наладки автоматизированных систем на основе современных компьютерных технологий.

1.1 Самостоятельная работа по учебным пособиям

Ввиду стремительности изменения аппаратно-программного обеспечения ПК, нельзя ограничиваться только тем материалом, который предложен в учебном пособии [1]. Полезно воспользо-

ваться дополнительной литературой, список которой будет приведен в разделе 6 настоящих методических указаний. Не возбраняется использование дополнительной литературы, не вошедшей в список раздела 6.

Выбрав одно или несколько учебных пособий в качестве основных для определенной части курса, следует, прорабатывая пункты рабочей программы, составлять конспект. С помощью такого конспекта, составленного для каждого из изучаемых в семестре разделов или тем, удобно решать типовые задачи, контрольные работы, готовиться к экзамену.

1.2 Требования, предъявляемые к выполнению контрольных и лабораторных работ

Выполнение контрольных и лабораторных работ по курсу АПО ЭВМ в основном связано с программированием на встроенном ассемблере IDE Borland Pascal или макроассемблере. Поэтому, перед выполнением контрольных работ необходимо повторить материал по использованию встроенного ассемблера из курсов «Информатика», «Средства отладки микропроцессорных систем» и изучить документацию по макроассемблеру для персональных компьютеров. Кроме того, необходимо изучить систему команд процессора 80x86 и математического сопроцессора 80x87, и методы адресации.

Форма отчетности при выполнении контрольных и лабораторных работ следующая. При оформлении фрагментов программ, как на ассемблере, так и на языке высокого уровня, обязательно наличие комментариев по каждому функционально завершенному блоку программы или перед фрагментом. Необходимо так же описать в отчете алгоритм работы программы (начертить блок-схему алгоритма) решения поставленной задачи. Программы без комментариев, алгоритма (блок-схемы) не принимаются.

Все программы должны быть предварительно отлажены и работоспособны.

2 СОДЕРЖАНИЕ ЛЕКЦИОННОГО КУРСА

Учебное пособие Аппаратное и программное обеспечение ЭВМ» состоит из 10 глав и разбито на 17 лекций. Для успешного выполнения контрольных и лабораторных работ по курсу необходимо изучить это учебное пособие. Краткое содержание лекционного курса приведено ниже:

Лекция 1

Открытая архитектура. IBM PC, XT, AT, PS/2, PS/1 совместимость и отличия. Поколения микропроцессоров серии x86 фирмы Intel. Процессоры Intel 80286 /80386 /80486, Intel Pentium, Intel Pentium Pro, 5x86 (Cyrix), 6x86 (Cyrix), M2 (Cyrix), K6 (AMD), K6-2 (AMD), K6-III (AMD), VIA Cyrix III (Cyrix), Celeron (Intel), Athlon (AMD), Duron (AMD), Morgan (AMD), Pentium II (Intel), Pentium III (Intel), Pentium 4 (Intel).

Лекция 2

Архитектура процессора. SX, DX, SX2, DX2 и DX4. Ядро, кэш и конвейер процессора. Микронная технология, зерно, напряжение питания ядра процессора. Тактовая частота и Bus Factor. Кэш L1, L2 и L3. Математический сопроцессор. Типы корпусов микросхем центрального процессора (PQFP, SQFP, PGA, SPGA, PPGA). SEC-картридж, ZIF, Socket и Slot.

Лекция 3

Программная модель микропроцессоров серии x86. Процессоры (cisc, risc, misk, hll). Регистры процессора. Сопроцессор (модуль плавающей точкой). Регистры сопроцессора. MMX-технология, команды MMX. Другие SIMD-технологии (XMM, MMX2/3DNow!, Enhanced 3DNow!, SSE, 3DNow! Professional и SSE2). Конвейер процессоров Pentium (P5/P6). Гиперконвейер Intel Pentium 4.

Лекция 4

Процессоры Intel Pentium 4. NetBurst — седьмое поколения процессоров. Технология Hyper-Pipelined. Execution Trace Cache.

Rapid Execution Engine. Технологии Advanced Dynamic Execution и Advanced Transfer Cache. Streaming SIMD Extensions 2 (SSE2).

Лекция 5

Материнская плата. Форм-фактор материнской платы. Частота платы и внутренний множитель процессора. Chipset (набор интегральных микросхем), Мостовая и концентраторная архитектура наборов микросхем. Магистральные интерфейсы (шины). Пропускная способность шины, информационные потоки в шинах, сбалансированность наборов микросхем. Интегрированные на материнской плате звуковые процессоры, сетевые адаптеры, модемы и т.п.

Лекция 6

Магистральные интерфейсы PC. Универсальные шины (XT, ISA, MCA, EISA, VESA (VLB), PCI, PCMCIA, AGP). PC-карты и технология Plug-and-play. Системные шины PC (GTL/GTL+(Intel), EV6 (AMD), AGTL/AGTL+(Intel)). Внутренние шины чипсета PC (Внутренний 32-бит PCI-интерфейс, V-Link (VIA), Hub Interface (Intel), MuTIOL (SiS)). Шины памяти (SDR SDRAM, DDR SDRAM, RDRAM). ACPI-интерфейс.

Лекция 7

Чипсеты фирмы Intel (i430XX PCIset, i440XX AGPset, i810XX, i815XX, i820XX, i840XX, i850XX, i845XX). Чипсеты фирмы VIA Technologies (VIA Apollo XX, VIA Apollo Pro XX). Чипсеты фирмы SiS (SiS5XX, SiS6XX, SiS 645). Чипсеты фирмы ALi (ALi Aladdin XX, ALiMAGIK и MobileMAGiK). Чипсеты фирмы AMD (AMD-640, AMD-750, AMD-760).

Лекция 8

Микросхемы памяти. Тип, объем и структура RAM. Кэш память. Статическая и динамическая RAM. Синхронная и асинхронная память. Время доступа, диаграмма циклов чтения/записи, CAS и RAS. Контроль правильности передачи данных Parity и ECC (Error Correction Code). Технология Presence Detect и Serial Presence Detect.

Лекция 9

Статические микросхемы памяти (Asynchronous SRAM, SyncBurst SRAM, PB SRAM). Динамические микросхемы памяти (PM DRAM, FPM DRAM, EDO, Burst EDO, SDRAM, DDR SDRAM, RDRAM). Комбинированные типы микросхем памяти (DRAM-SRAM, Enhanced SDRAM, CDRAM, High Speed SDRAM). Обозначения корпусов микросхем и типов модулей памяти (DIP, SIP, SIPP, SIMM, DIMM, CELP или COAST, RIMM). Предел производительности ОЗУ разных типов.

Лекция 10

Технологии VCM (Virtual Channel Memory), Active Link и Intellectual RAM. Оперативная память. Программная модель памяти. Conventional memory, Expanded Memory, eXtended Memory Specification, High Memory Area, Upper Memory Blocks. Защищенный режим процессора 80286. Защищенный режим процессоров 80386/80486. Интерфейс DPMI Интерфейс VCPI.

Лекция 11

Дисплей. Мониторы. LCD-дисплеи. Видеокарты CGA, EGA, VGA, SVGA, MDA, MCGA, Hercules, IBM8514. Видео ускорители. Шина AGP. Технология DirectX. Современные видео акселераторы.

Лекция 12

Дисковая подсистема PC. Стороны, дорожки и сектора. Емкость, плотность записи и время доступа. Форматирование, редактирование, дефрагментация, запись и чтение. Таблица FAT и структура каталогов. Boot sector, Boot record. Файловые системы (FAT-16, NFS). НГМД. НЖМД ST412/ ST506, ESDI, IDE (EIDE), SCSI.

Лекция 13

BIOS Setup. Настроечные параметры (Standart BIOS setup, Advanced BIOS Features, Advanced Chipset Control, Integrated Peripherals, PnP/PCI Configurations, Power management setup, PC health status, Frequency/Voltage Control).

Лекция 14

Интерфейсы ввода/вывода. Параллельный интерфейс PC. LPT-порт. Стандарт IEEE 1284-1994. Интерфейсы LPT-порта PC (Compatibility Mode, Centronics, SPP, Nibble Mode, Byte Mode, EPP, ECP). Регистры LPT-интерфейсов, прерывания и команды. Согласование режимов IEEE 1284. Физический и электрический интерфейсы. Кабели и коннекторы.

Лекция 15

Порт последовательной передачи данных, интерфейс RS-232. Интерфейс «токовая петля». Game-порт. USB, USB 2.0. IEEE 1394 (firewire, iLink). SCSI, SCSI-II /Ultra-Wide SCSI.

Лекция 16

Модем. Виды модуляции. Цифровые (ISDN) модемы. Сетевые адаптеры. ETHERNET, ARCNET, TOKEN RING, FAST ETHERNET, 100VG, ATM, FDDI. Методы доступа.

Лекция 17

Внешние запоминающие устройства различных типов. Магнитооптические дисководы (CD-R, CD-RW, DVD-ROM, DVD-RW). Стримеры и сменяемые диски, Hard card, MO, LS120,Arvid. ZIP-устройства, DAT. Периферийные устройства. Принтеры, плоттеры и сканеры. Цифровые и веб-камеры. Клавиатура XT/AT. MS Mouse. Световое перо, джойстик, планшет, дигитайзер. Источники питания. UPS. Конструктивное исполнение (BIG-TOWER, TOWER, AT, COMPACT, SLIM, LAPTOP, EURO, NOTEBOOK).

3 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КОНТРОЛЬНЫХ РАБОТ

3.1 Контрольная работа № 1

Тема: Разработка фрагмента программы на встроенном ассемблере языка Turbo Pascal v7.0

Продолжительность — 6 часов.

Максимальный рейтинг — 10 баллов.

Теоретическая часть

Как и большинство языков высокого уровня, язык Turbo Pascal имеет в своем арсенале такой удобный и необходимый для разработчика ПО инструмент, как возможность использования в составе программного кода блоков, написанных непосредственно на языке ассемблера. Синтаксис этого блока ассемблерной вставки приведен на листинге Listing 1.

Listing 1

```
(*=======================================*)
asm
(* фрагмент программного кода *)
end;
(*=======================================*)
```

Этот инструмент позволяет сохранить необходимые функции и гибкость языка ассемблера, а так же предлагает удобный интерфейс с переменными, регистрами и сегментами, характерный для языка высокого уровня. Ответственность за это берет на себя компилятор языка Turbo Pascal v 7.0.

Задание к контрольной работе № 1

1. Освоить методы работы со встроенным ассемблером языка Turbo Pascal (блок ассемблерной вставки, техника отладки, навыки просмотра и трассировки, диалог изменения переменных и регистров, окно регистров).

2. Повторить изученные в предыдущих курсах понятия: программная модель и система команд процессора 8086, регистры, методы адресации.

3. Реализовать фрагмент кода в соответствии с индивидуальным заданием на языке Turbo Pascal и на встроенном ассемблере языка Turbo Pascal v7.0.

4. Проиллюстрировать идентичность блока ассемблерного кода блоку, реализованному на языке Turbo Pascal.

5. Сравнить время выполнения блоков программы.

6. Сделать вывод о проделанной работе.

Примечания

1. Для оценки времени выполнения блоков программы использовать процедуру GetTime модуля DOS.

2. В случае если фрагмент программы выполняется быстрее 1/100 доли секунды, повторить его в цикле необходимое число раз, а полученную величину разделить на количество повторений.

Пример

Рассмотрим в качестве примера фрагмент программы, приведенный на листинге (Listing 2) эквивалентный исходному блоку (Listing 3), реализованному на языке Turbo Pascal:

Listing 2

```
(*=========================================*)
asm                {-------------------------}
   mov i, 10       { i:=10                   }
   mov k, 101      { k:=101                  }
@m3:               {-------------------------}
   mov AX, k       { (i*k+k*k) div 5         }
   mul I           {                         }
   mov BX, AX      {                         }
   mov AX, k       {                         }
   mul k           {                         }
   add AX, BX      {                         }
   mov DX, 0       {                         }
   mov CX, 5       {                         }
   div CX          {-------------------------}
   cmp DX, 1       { если остаток (DX)=1     }
   jne @m1         { иначе goto m1           }
   mov CX, k       { то k:=k-i+10            }
   sub CX, I       {                         }
   add CX, 10      {                         }
```

```
   mov k, CX      {-------------------------}
   jmp @m2        { goto endif              }
@m1:              {-------------------------}
   add k, 1       { иначе i:=i*(k+1)        }
   mov AX, k      {                         }
   mul I          {                         }
   mov i, AX      {                         }
   sub k, 1       {-------------------------}
@m2:              {-------------------------}
   mov DX, k      { сравнение i=k           }
   cmp i, DX      {-------------------------}
   jnge @m3       { если i>=k то goto m3    }
end;              {-------------------------}
(*=======================================*)
```

Переменные i и k, используемые в ассемблерном фрагменте описываются средствами языка Turbo Pascal, так же как и в (Listing 3). Обратите внимание на то, что при программировании цикла, проверка условия производится после выполнения расчетного блока (аналогично паскалевскому блоку repeat-until), таким образом, данные фрагменты кода являются полностью эквивалентными по своим алгоритмам.

Listing 3

```
(*=======================================*)
...
var i, k :word;
...
  i:=10;
  k:=101;
  repeat
    if ((i*k+k*k) mod 5)<> 1
      then i:=i*(1+k)
      else k:=k-i+11;
   until i>=k;
...
(*=======================================*)
```

Варианты индивидуальных заданий к контрольной работе № 1

1.
```
i:=57; k:=19;
while i<655 do
  if ((i + k div 2) mod 11) = 5
    then k:=k*k+(i div 2)
    else i:=i+k;
```

2.
```
i:=1;
for k:=91 to 155 do
  if ((k mod 5)=1) or (k div 130=0)
    then i:=i+k
    else i:=i-k;
```

3.
```
i:=1; k:=5;
while i<120 do
  if ((i + k) mod 7) = 3
    then i:=i*i+1
    else i:=k*i+17;
```

4.
```
i:=2;
for k:=565 downto 9 do
  if ((k mod 4)=3) or (k div 31=0)
    then i:=i+k
    else i:=i+13;
```

5.
```
i:=7; k:=9;
while i<120 do
  if ((i + k) mod 7) = 2
    then i:=i*2+k
    else k:=k*2+i;
```

6.
```
i:=1; k:=1024;
repeat
  if ((i*i + k) mod 23) = 3
    then i:=i*(i+1)
```

```
    else k:=k-i
until i>k;
```

7.

```
i:=587; k:=19;
while i>199 do
  if ((i + k) mod 11) <> 5
    then i:=i-k
    else k:=k*k+(i div 2);
```

8.

```
i:=96;
for k:=240 downto 15 do
  if ((k mod 7) = 2) xor ((k div 17) = 2)
    then i:=i-3*k
    else i:=i+k;
```

9.

```
i:=1024; k:=11;
repeat
  if ((i + k*k) mod 5) <> 1
    then i:=i+1
    else k:=k+i;
until i>k;
```

10.

```
i:=1; k:=1024;
repeat
  if ((i*i + k) or 9) = 3
    then i:=i*(i+k)
    else k:=k-i;
until i>k;
```

11.

```
i:=96;
for k:=24 downto 15 do
  if ((k mod 7) > 3) xor ((k div 27) <10)
    then i:=i-k
    else i:=i+k*k;
```

12.

```
i:=7; k:=9;
```

```
while i<123 do
  if ((i + k) mod 31) = 2
    then i:=i*2+k
    else k:=k*2+i;
```

13.

```
i:=2;
for k:=158 downto 19 do
  if ((k or 4)=3) or (k div 23=0)
    then i:=i-k
    else i:=i+k;
```

14.

```
i:=1; k:=5;
while i<123 do
  if ((i - k) mod 13) <> 6
    then i:=i*i-k
    else k:=k*k+i;
```

15.

```
i:=1;
for k:=255 downto 9 do
  if ((k xor 5)=1) or (k mod 130=0)
  then i:=k mod i
  else i:=i+k;
```

16.

```
i:=96;
for k:=211 downto 15 do
  if ((k mod 7)<>2) xor ((k div 27) <> 2)
  then i:= k div 13
  else i:= i + k mod 31;
```

17.

```
i:=1024; k:=10;
repeat
  if ((i + k) mod 13) <> 4
    then i:=i-1
    else k:=k+i;
until i<k;
```

18.

```
i:=1; k:=1024;
repeat
```

```
  if ((i*k - k+i) mod 9) = 13
    then i:=i*(i+5)
    else k:=k-i-5;
until i>k;
```

19.

```
i:=587; k:=19;
while i>199 do
  if ((i + k) div 17) <> 5
    then i:=i-(k mod (13*5))
    else k:=k*k+(i div 2);
```

20.

```
i:=102; k:=10;
repeat
  if (((i*i + k*k) mod 5 = 17) or (i*k div 13 = 0))
    then i:=i+1-k
    else k:=k+13;
until i<k;
```

21.

```
i:=102; k:=10;
repeat
  if ((i*i + k*k) mod 5) <> 1
    then i:=i+11+k
    else k:=k+i;
until i<k;
```

22.

```
i:=1;
for k:=9 to 255 do
  if ((k div 5)=1) and (k div 130=0)
    then i:=i*k
    else i:=i+k;
```

23.

```
i:=1; k:=5;
while i<120 do
  if ((i + k) div 7) <> 6
    then i:=i*i+k
    else k:=k*k-i;
```

24.

```
i:=212;
```

```
for k:=$AFA downto 9 do
  if ((k div 4)=3) or (k mod 100 = 13)
  then i:=i mod k
  else i:=i+1;
```

25.

```
i:=7; k:=9;
while i<120 do
  if ((i + k) div 7) <> 2
    then i:=i*2+k
    else k:=k*2+i;
```

26.

```
i:=1; k:=1024;
repeat
  if ((i*i + k) div 11) = 3
    then i:=i*(i+1)
    else k:=k-i
until i>k;
```

27.

```
i:=557; k:=19;
while i>199 do
  if ((i + k) div 115) <> 5
    then i:=i-k
    else k:=k*k+(i div 2);
```

28.

```
i:=96;
for k:=140 downto 15 do
  if ((k div 7)=2) xor ((k mod 27) = 2)
    then i:=i*13 mod (k*2)
    else i:=k;
```

29.

```
i:=1024; k:=10;
repeat
  if ((i + k) mod 5) <> 1
    then i:=i-1
    else k:=k+1;
until k>i;
```

30.

```
i:=96;
```

```
for k:=124 downto 15 do
  if ((k mod 7) > 2) xor ((k div 27) <10)
  then i:=i-k
  else i:=i+k;
```

31.

```
i:=1024; k:=10;
repeat
  if ((i*k) mod 5) > 1
    then i:=i-k*k div 13
    else k:=k+i;
until i<k;
```

32.

```
i:=45; k:=19;
repeat
  if ((i*i + i*k + k*k ) div 3) < 1
    then i:=i+k mod 17
    else k:=k+i-34;
until k>i;
```

33.

```
i:=1; k:=5;
while i<120 do
  if ((i * k) div 19) <> 3
    then i:=i+ k
    else i:=i+ k mod 11;
```

34.

```
i:=2;
for k:=200 downto 9 do
  if ((k xor 4) = 13) or ((k div 7) = 0)
    then i:=i*k - (k*k+i)
    else i:=2* (i div k);
```

35.

```
i:=9; k:=7;
while i<120 do
  if ((i + k) div (k mod 9)) <> 2
    then i:=i+k*2
    else k:=k+i*2;
```

36.

```
i:=12; k:=104;
```

```
repeat
  if ((i*i + k) or 9) = 3
    then i:=i*(i+k)
    else k:=k-i;
until i>k;
```

37.

```
i:=257; k:=19;
while i>199 do
  if ((i + k div 2) mod 11) = 5
    then i:=i-k
    else k:=k+(i div 2);
```

38.

```
i:=19; k:=487;
while i<=199 do
  if ((i + k) mod 11) <> 5
    then i:=(i+k)*2
    else k:=k*k+(i div 2);
```

39.

```
i:=241; k:=10;
repeat
  if ((i + 2*k) div 3) < 1
    then i:=i-17
    else k:=k+i;
until i<=k;
```

40.

```
i:=387; k:=39;
while i>199 do
  if ((i + k) mod 13) = 5
    then i:=i-k
    else k:=k+(i div 2);
```

3.2 Контрольная работа № 2

Тема:

Массивы данных. Исследование методов адресации

Продолжительность — 6 часов

Максимальный рейтинг — 10 баллов

Теоретическая часть

Особенности распределения адресного пространства в компьютерах IBM-PC

В микропроцессоре ***i8086*** указатель команд и регистры имеют разрядность 16 бит, и, можно подумать, что процессор может обратиться только к памяти объемом не более 64 кБ (65536 Б, т.е. 216 байт). Однако на самом деле микропроцессор *8086* всегда генерирует 20-битовые адреса, получаемые сложением сдвинутого на 4 позиции влево регистра сегмента и смещения. Например, если смещение адреса равно `390Fh`, а сегмент равен `0100h`, то исполнительный адрес будет равен `0490Fh`:

```
0000 0001 0000 0000          — сдвиг сегмента;
0011 1001 0000 1111          — смещение;
0000 0100 1001 0000 1111     — адрес.
```

При использовании 20-битовой шины адреса непосредственно можно обратиться к 1 МБ (2^{20} = 1 048 576 Б).

Более поздние процессоры имеют еще большее адресное пространство (*80286* — 16 МБ, *80386* — 4 ГБ, и т.д.). С одной стороны, это достигается за счет расширения разрядности смещения в сегменте, с другой стороны — адреса сегментов задаются в таблице дескрипторов, где под этот адрес отводится больше 32-х бит (под селектор отводится 16 бит). В связи с чем появились следующие понятия:

– ***дескриптор*** — это набор параметров, описывающих некоторый участок (сегмент) памяти: адрес, размер, права программ на запись, чтение и исполнение команд из этого сегмента, тип данных, хранимых в сегменте и т.д. Дескрипторы хранятся в специальных таблицах в ОЗУ;

– ***селектор*** — это номер дескриптора в таблице;

– ***сегмент*** — это участок памяти, описанный дескриптором.

При выполнении лабораторных работ этого курса мы будем пользоваться только обычной, сегментной системой адресации. Логическое распределение оперативной памяти определяется не только применяемой операционной системой, но и особенностями аппаратной реализации IBM-совместимых машин.

Для машин на базе процессоров до *80586* можно выделить следующие логические области адресного пространства:

– ***Conventional Memory*** — стандартная оперативная память;

– ***UMA*** (*Upper Memory Area*) — область верхней памяти;

– ***HMA*** (*High Memory Area*) — область высшей памяти;

– ***XMS*** (*eXtended Memory Specification*) — расширенная память;

– ***EMS*** (*Expanded Memory Specification*) — дополнительная память.

Следует также упомянуть о наличии КЭШ-памяти первого и второго уровней, однако они «прозрачны» для программистов и могут управляться минимально, в основном, в интересах ОС. Управление КЭШем первого уровня специфично для каждой модели процессора, а управление КЭШем второго уровня аналогичным образом специфично для конкретного чипсета.

Кроме того, начиная с процессора *80286*, существует понятие виртуальной памяти. Фактически, это дисковое пространство, обслуживаемое операционной системой защищенного режима. Процессоры, начиная с *80286*, сообщают операционной системе о необходимости догрузить тот или иной блок с диска в физическую память. Обратите внимание: такое распределение памяти диктуется не процессором, а используемым чипсетом. Естественно, что из соображений совместимости оно совпадает для существующих чипсетов.

• ***Стандартная оперативная память*** (*Conventional memory*).

Процессоры *8086*, *8088*, как было сказано выше, имели 20 адресных линий и. следовательно, могли адресовать только 1 МБ адресного пространства. В машинах класса XT это пространство делилось на 384 кБ специального назначения (*UMA*) и 640 кБ оперативной памяти. Позднее эти 640 кБ стали называть стандартной или основной оперативной памятью, они доступны любому процессору. Conventional memory наиболее проста для обращения в реальном режиме, присутствует в любой современной машине и обслуживается любой операционной системой в минимальной конфигурации.

В реальном режиме эта память занимает адреса 00000h-9FFFFh.

• ***Область верхней памяти*** (*Upper Memory Area — UMA*).

Эта область также существует для любого процессора клона *80x86*. Располагается на адресах A0000h-FFFFFh, размер 384 кБ, доступна в любом режиме любому процессору, т.к. не выходит за предел 1 МБ.

В машинах класса XT следующие области этой памяти:

A0000h-BFFFFh — адресное пространство для доступа к памяти видеосистемы машины;

C0000h-C7FFFh — адресное пространство ПЗУ BIOS видеосистемы;

C8000h-EFFFFh — свободно (может использоваться для дополнительной памяти (машин XT), ПЗУ BIOS SCSI контроллеров, т.д.);

F0000h-FFFFFh — ПЗУ системной BIOS.

• ***Область высшей памяти*** (*High Memory Area — HMA*).

Это самая маленькая область памяти, ее размер чуть меньше 64 кБ.

Эта область памяти существует у процессоров *80286* и выше, однако адресуется в реальном режиме и образована следующим образом. Попробуем рассчитать реальный адрес (комбинацию на 20-битной адресной шине) для адреса, представленного следующими значениями сегмента и смещения FFFEh: EFFDh:

FFFEh +EFFDh = 10EFDDh

Однако при 20-битной адресной шине останется только пять цифр, следовательно, реальный адрес получится равным 0EFDDh, т.е. в начале памяти. Однако для 24-битной адресной шины *80286* получится именно рассчитанный адрес: 10EFDDh, т.е. получается дополнительно 64 кБ адресного пространства. Для того чтобы программы, написанные для *8086* и адресующиеся на начало памяти через верхние адреса описанным выше образом, исполнялись на *80286* и более поздних процессорах, чипсет должен блокировать использование линии ***A20***, всегда обнуляя ее. Однако если A20 не блокировать, в реальном режиме появляются дополнительные 64 кБ адресного пространства, называемого HMA, ко-

торое используется как окно для отображения дополнительной памяти.

• ***Расширенная память*** (*extended Memory Specification — XMS*).

В машинах класса AT и выше, содержащих *80286* и более поздние процессоры, для расширения памяти стал использоваться новый (по сравнению с дополнительной памятью XT) метод. Вся память, как conventional, так и extended, располагается в одном адресном пространстве процессора (без деления на окна), однако для обращения к адресам выше 1 МБ требуется хотя бы временно перевести процессор в защищенный режим. Несложные программы, желавшие использовать XMS, могли обращаться к этой памяти через менеджер *HIMEM.SYS*, хотя этот способ не очень быстрый.

• ***Дополнительная память*** (*Expanded Memory Specification — EMS*).

Хотя адресная шина процессоров XT была только 20 бит, допускалась установка дополнительной памяти теоретически неограниченного объема. Однако дополнительная память в этом случае была «видна» в одном или нескольких свободных окнах UMA фрагментами по 64 кБ, а необходимый фрагмент выбирался специальными программными переключателями (регистром управления памятью). Физически такая память представляла собой обычную карту с микросхемами памяти и управления. В дальнейшем, при расширении адресного пространства последующих процессоров, от дополнительной памяти в виде карт отказались, а вышеописанный метод доступа теперь эмулируется драйвером *EMM386.EXE* для устаревших программ, «не умеющих» использовать современный вариант адресации.

EMM386.EXE требует для своей работы драйвера HIMEM.SYS и фактически является для последнего не более чем одним из клиентов. Таким образом, в машинах на процессорах 80386 и выше EMS память — это лишь способ обращения к XMS памяти.

Методы адресации

Большинство команд процессора вызываются с аргументами (операндами). Например, команда сдвига SHRD требует трех ар-

гументов. Рассмотрим методы задания адреса хранения операндов — методы адресации.

Микропроцессоры *8086, 8088* и более поздние имеют развитую систему адресации, предоставляющую пользователю широкие возможности обращения к операндам. В зависимости от типа команды операнд может представлять собой байт (8 бит), слово (16 бит), двойное слово (32 бита), или другие размеры (64, 80 бит — с ними, в основном, оперирует сопроцессор) и храниться в регистре процессора (или сопроцессора) или памяти. Всего МП *8086* имеет девять методов адресации, в более поздних процессорах количество методов увеличено.

Рассмотрим несколько простых команд. Операция уменьшения значения на единицу записывается так:

DEC d,

где операнд d — некоторый аргумент (регистр или ячейка памяти).

Команда логического отрицания записывается так:

NOT d,

при ее выполнении содержимое d поразрядно инвертируется.

Команда пересылки данных:

MOV d, s,

где s — некоторый аргумент (регистр или ячейка памяти). Не допускается ситуация, когда s и d одновременно являются ячейками памяти. При выполнении этой команды содержимое источника s копируется в приемник d. Значение s не изменяется.

Команда сложения имеет формат:

ADD d, s.

Содержимое источника s складывается с приемником и пересылается в приемник d.

Для адресации операндов процессором зачастую используются регистры общего назначения, в которые можно записать либо адрес, либо сам операнд. Но тогда в команде необходимо ука-

зать, как используется этот регистр, вот тут и возникает потребность в различных методах адресации. Рассмотрим несколько возможных методов адресации.

• ***Прямой регистровый метод*** характерен тем, что операнд находится в выбранном регистре общего назначения. Обращение к нему будет наиболее простым и быстрым, но регистров мало, и этот метод используется для хранения промежуточных или специально выделенных данных. При использовании регистровой адресации операнд извлекается из регистра (или загружается в регистр). Регистром может являться любой из регистров ЦП или сопроцессора.

Допустим, необходимо уменьшить на единицу значение регистра CX. Символическое обозначение команды на языке ассемблера выглядит следующим образом:

```
DEC CX,
```

где в качестве операнда выступает CX — имя уменьшаемого регистра.

• ***Непосредственный метод*** адресации предполагает запись операнда непосредственно в программе после команды. Метод удобен при работе с константами — величинами не изменяемыми при работе программы, которые можно записывать прямо в команде на языке ассемблера. Но непосредственный метод нельзя использовать при работе с именами (переменными), т.к. адрес операнда связан с местом нахождения команды, но явно не определен. При использовании непосредственной адресации микропроцессор в качестве операнда-источника берет константу, которая содержится в команде. Например, команда

```
MOV AL, 31
```

загружает значение 31 в регистр AL, а команда

```
MOV SI, FFFE
```

загружает значение `FFFEh` в регистр SI.

• ***Прямая адресация.*** При использовании прямой адресации 8086-совместимый процессор берет указанное в команде смещение, добавляет его к сдвинутому содержимому указанного реги-

стра сегмента (по умолчанию берется регистр сегмента данных DS) и по полученному адресу выбирает операнд. Более поздние процессоры работают так же, за исключением защищенного режима, в котором адрес сегмента берется из таблицы дескрипторов. Например, команда

```
MOV BX, [00FEh]
```

в реальном режиме заносит в регистр BX значение, расположенное по смещению `00FEh` относительно сегмента DS.

• ***Косвенный регистровый метод*** адресации определяет содержимое выбранного регистра общего назначения как адрес операнда. Такой метод позволяет обратиться по абсолютному адресу к любой ячейке оперативной памяти или к любому регистру внешнего устройства. Обращение к операнду производится через канал, что требует дополнительного времени.

При использовании косвенной регистровой адресации микропроцессор берет в указанном регистре смещение, добавляет к сдвинутому регистру сегмента и по полученному адресу выбирает операнд.

Напимер, команда

```
NOT word ptr [BX],
```

где word ptr — префикс размерности 16-битного операнда выполняет операцию NOT с операндом, расположенным по сдвигу, размещенному в регистре BX относительно сегмента DS, т.е. содержимое ячейки, на которую указывает пара DS:BX, дополняется до единицы.

Процессоры до *80386* могли использовать, в качестве индексных, только регистры BX, SI, DI и BP, более поздние допускают использование также EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP.

Если регистр сегмента не указан, подразумевается использование DS, за исключением случая адресации через EBP, ESP или BP — в этом случае сегментный регистр по умолчанию — SS.

• ***Индексный метод адресации*** предполагает суммирование адреса, записанного в регистре общего назначения, с некоторым

индексом или смещением, записанным либо в следующем слове после команды либо в специальном регистре. Этот регистр может иметь название регистр базы, сегментный регистр и т.п. Использование индекса позволяет обратиться к другой ячейке памяти или к другому регистру ВУ, не изменяя исходный адрес. Индексный метод широко используется при работе с массивами данных: обращение к нужному элементу массива производится по постоянному адресу (имя массива) с учетом номера элемента (индекс).

• ***Вариант индексного метода*** — базовый метод адресации. Принципиальное отличие его от предыдущего состоит в том, что смещение записано не в команде, а в специальном аппаратно назначенном регистре базы. Обеспечивается возможность не только сдвинуть на фиксированную величину адрес одного или группы операндов при постоянных программных адресах, но и сместить в оперативной памяти программу или обратиться к другому блоку данных. Другими словами, имеется возможность записывать уже готовую программу или блок данных в нужное место памяти. Этот метод используется в многозадачных операционных системах для распределения памяти между задачами, страницами, разделами или сегментами.

При использовании адресации по базе *8086*-микропроцессор берет в регистре BX, SI, DI или BP смещение, складывает его со значением сдвига, заданным в команде, добавляет к сдвинутому регистру сегмента и по полученному адресу берет операнд. Рассмотрим команду

```
MOV AX, [0Eh+BX]
```

Здесь из ячейки памяти со смещением [000Eh + содержимое BX] взят операнд и помещен в аккумулятор.

• ***Косвенная регистровая адресация с индексированием***. При использовании косвенной регистровой адресации с индексированием процессор берет в регистре BX или BP первое смещение (базу), складывает со смещением, взятым в регистре SI или DI (индекс), добавляет к сдвинутому регистру сегмента и по полученному адресу берет операнд. Рассмотрим команду,

```
INC byte ptr [BX+DI]
```

Здесь byte ptr — это префикс размерности 8-битового операнда, т.е. операнд по адресу [BX+DI] увеличивается на 1.

Процессоры до *80386* допускали использование следующих комбинаций регистров: BX+SI, BX+DI, BP+SI, BP+DI. Более поздние процессоры допускают также использование любых комбинаций регистров EAX, EBX, ECX, EDX, ES1, EDI, ESP, EBP. Запрещенной является только комбинация ESP+ESP.

• ***Адресация по базе с индексированием.*** При использовании индексированной базовой адресации микропроцессор берет смещение (базу) в регистре BX или BP, складывает со смещением (индексом), взятым в регистре SI или DI, складывает со сдвигом, заданным в команде, добавляет к сдвинутому регистру сегмента и по полученному адресу берет операнд. Рассмотрим команду

```
MOV AX,[0200+BX+DI]
```

по адресу [0200+BX+DI] взят операнд и помещен в аккумулятор.

Для работы с именами используется относительный метод адресации. Суть его состоит в том, что в команде, содержащей два слова, указан адрес операнда в виде смещения относительно этой команды. В ассемблере адрес операнда указывается прямо в команде, но при сдвиге команды по программе адрес операнда будет изменяться. А это означает, что если мы в программе расположим последовательно несколько одинаково записанных команд, то каждая из них будет обращаться к своему операнду.

В языках высокого уровня данный метод используется для работы с переменными. Поскольку адреса данных определяются по отношению к соответствующим командам, появляется возможность легко строить перемещаемые по оперативной памяти программы. Это свойство используется, если необходимо загрузить в память одновременно несколько задач, а начальные адреса загрузки для каждой программы не определены.

Относительная адресация имеет ограниченное применение, т.к. используется только в командах перехода. При использовании относительной адресации процессор извлекает смещение из следующего после кода операции байта и прибавляет его к указателю команд IP. Байт смещения рассматривается как число со знаком. Следует отметить, что IP во время выполнения команды

всегда указывает на первый байт следующей команды, т.е. команда перехода с нулевым смещением не задает какого либо изменения порядка выполнения команд.

Рассмотрим пример:

```
JMP 122
```

Здесь произошел переход по адресу 0100h+2h+20h=0122h. Для процессоров до 80386 в качестве смещения использовался только байт, в более поздних могут использоваться два или четыре байта.

• ***Косвенная регистровая адресация с масштабированием.*** Этот метод адресации доступен только на процессорах 80386 и более поздних. Он подобен косвенной регистровой адресации, но значение регистра предварительно может быть умножено на число 2, 4 или 8:

```
MOV AX, [ESI*2]
```

Этот метод может использовать в качестве регистра-указателя только регистры EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP.

• ***Адресация по базе с индексированием и масштабированием.*** Этот метод, аналогично предыдущему, доступен только на процессорах 80386 и более поздних. Он подобен методу адресации по базе с индексированием, отличаясь от него лишь тем, что значение регистра индекса будет предварительно умножено на 2, 4 или 8. Таким образом, смещение операнда в сегменте будет вычислено следующим образом:

```
база + индекс * множитель + сдвиг
```

Например:

```
MOV AX, [EBX+ESI*4+0020]
```

В качестве регистра базы может быть использован любой: EAX. EBX, ECX, EDX, ESI, EDI, ESP, EBP, в качестве регистра индекса тоже, за исключением ESP.

Задание к контрольной работе № 1

1. Повторить изученные в предыдущих курсах понятия: Оперативная память, системная память, система адресации, виртуальная память.

2. Повторить (при необходимости освоить заново) следующие основные методы адресации: непосредственный метод, прямой регистровый, индексный (базовый), косвенный регистровый, базовый индексный и базовый индексный со смещением.

3. Используя встроенный ассемблер Turbo Pascal v7.0 запрограммировать фрагмент кода в соответствии с индивидуальным заданием каждым из четырех косвенных методов адресации (косвенно-регистровый, базовый, базовый индексный и базовый индексный со смещением) эквивалентный указанному блоку на языке Turbo Pascal v7.0.

4. Сделать вывод о проделанной работе.

Примечания

1. a[i] и b[i] — одномерные массивы 1×N. Элементы массивов a[i] и b[i] — знаковое двухбайтное целое число (Integer). Счетчик массива i=1..N — беззнаковое целое однобайтное число (Byte).

2. Перед началом расчетов заполнить массивы a[i] и b[i] случайными целыми числами из диапазона от 0 до 256 (Randomize, Random(256))

3. Запрограммировать четыре эквивалентных программных блока, реализующих индивидуальный фрагмент кода каждым из четырех косвенных методов адресации: обращение к элементам массивов организовать косвенно-регистровым, базовым, базовым индексным и базовым индексным со смещением методами.

4. Вывести заполненные массивы a[i] и b[i] в файл перед началом расчетов и после каждого программного блока для проверки идентичности расчетов.

5. В отчете привести файл с результатами расчетов.

Пример

Рассмотрим листинг Listing 5, иллюстрирующий применение различных методов косвенной адресации для обращения к

операндам. Листинг реализует фрагмент, эквивалентный коду, приведенному на языке Pascal на листинге Listing 4.

Listing 4

```
(*=============================================*)
var n:word;
    k,i:array [1..1000] of word;
begin
  n:=34;
  i[n]:=587;
  k[n]:=19;
  while i[n]>199 do
    if ((i[n] + k[n]) div 11) = 5
       then i[n]:=i[n]-k[n]
       else k[n]:=k[n]*k[n]+(i[n] div 2);
end.
(*=============================================*)
```

Фрагмент (Listing 5) иллюстрирует использование косвенного регистрового, базового, базового индексного и базового индексного со смещением методов адресации.

Listing 5

```
(*=============================================*)
(* Косвенно—регистровая адресация *)
asm
   Mov AX, 34         (* n:=34                   *)
   Mov n,  AX         (*-------------------------*)
   Mov AX, 587        (* i[n]:=587               *)
   Mov BX, offset i   (*                         *)
   Mov CX, n          (*                         *)
   Shl CX, 1          (*                         *)
   Add BX, CX         (*                         *)
   Mov [BX], AX       (*-------------------------*)
   Mov AX, 19         (* k[n] :=19               *)
   Mov BX, offset k   (*                         *)
   Add BX, CX         (*                         *)
   Mcv [BX], AX       (*-------------------------*)
WhileLabel1:
   Mov BX, offset i   (* while(i[n]>199)         *)
   Mov CX, n          (*                         *)
   Shl CX, 1          (*                         *)
   Add BX, CX         (*                         *)
   Mov AX, [BX]       (*                         *)
```

```
   Cmp AX, 199       (*                          *)
   Jbe EndWhileLabel1(*-------------------------*)
   Mov CX, n         (* i[n]                     *)
   Shl CX, 1         (*                          *)
   Xov BX, offset i  (*                          *)
   Add BX, CX        (*                          *)
   Mov AX, [BX]      (*-------------------------*)
   Xov BX, offset k  (* k[n]                     *)
   Add BX, CX        (*-------------------------*)
   Add AX, [BX]      (* i[n]+k[n]                *)
   Xor DX, DX        (*-------------------------*)
   Mov CX, 11        (* (i[n]+k[n] div 11)=5    *)
   Div CX            (*                          *)
   Cmp AX, 5         (*-------------------------*)
   Jne ElseLabel1    (* then                     *)
   Mov CX, n         (* i[n]:= i[n]-k[n]         *)
   Shl CX, 1         (*                          *)
   Mov BX, offset k  (*                          *)
   Add BX, CX        (*                          *)
   Mov AX, [BX]      (*                          *)
   Mov BX, offset i  (*                          *)
   Add BX, CX        (*                          *)
   Sub [BX], AX      (*                          *)
   Jmp EndIfLabel1   (*-------------------------*)
ElseLabel1:
   Mov CX, n         (* else                     *)
   Shl CX, 1         (*k[n]=k[n]*k[n]+(i[n]div2)*)
   Mov BX, offset i  (* i[n] div 2               *)
   Add BX, CX        (*                          *)
   Mov SI, [BX]      (*                          *)
   Snr SI, 1         (*                          *)
   Mov BX, offset k  (* k[n]*k[n]                *)
   Add BX, CX        (*                          *)
   Mov AX, [BX]      (*                          *)
   Mov DX, AX        (*                          *)
   Mul DX            (*                          *)
   Add AX, SI        (*                          *)
   Mov [BX], AX      (*-------------------------*)
EndIfLabel1:
   Jmp WhileLabel1
EndWhileLabel1:
   end;
(*============================================*)
(* Базовая адресация *)
```

```
asm
   Mov AX, 34          (* n:=34                    *)
   Mov n, AX           (*-------------------------*)
   Mov AX, 587         (* i[n]:=587                *)
   Mov BX, n           (*                         *)
   Shl BX, 1           (*                         *)
   Mov [BX+offset i], AX (*---------------------*)
   Mov AX, 19             (* k[n] :=19            *)
   Mov [BX+offset k], AX (*---------------------*)
WhileLabel2:
   Mov AX, [BX+offset i] (* while(i[n]>199)     *)
   Cmp AX, 199           (*                      *)
   Jbe EndWhileLabel2    (*---------------------*)
   Mov AX, [BX+offset i] (* i[n]                 *)
   Add AX, [BX+offset k] (* i[n]+k[n]            *)
   Xor DX, DX            (*---------------------*)
   Mov CX, 11            (* i[n]+k[n]div11)=5   *)
   Div CX                (*                      *)
   Cmp AX, 5             (*---------------------*)
   Jne ElseLabel2        (* then                 *)
   Mov AX, [BX+offset k] (* i[n]:=i[n]-k[n]     *)
   Sub [BX+offset i], AX (*                      *)
   Jmp EndIfLabel2       (*---------------------*)
ElseLabel2:
   Mov SI, [BX+offset i] (* else                 *)
   Shr SI, 1             (*k[n]*k[n]+(i[n]div2)*)
   Mov AX, [BX+offset k] (*                      *)
   Mov DX, AX            (*                      *)
   Mul DX                (*                      *)
   Add AX, SI            (*                      *)
   Mov [BX+offset k], AX (*---------------------*)
EndIfLabel2:
   Jmp WhiIeLabel2
EndWhileLabel2:
end;
(*=============================================*)
(* Базовая индексная адресация *)
asm
   Mov SI, offset i    (*----------------------*)
   Mov DI, offset k    (*                      *)
   Mov AX, 34          (* n:=34                *)
   Mov n, AX           (*----------------------*)
   Mov AX, 587         (* i[n]:=587            *)
   Mov BX, n           (*                      *)
```

```
   Shl BX, 1           (*                        *)
   Mov [BX+SI], AX     (*------------------------*)
   Mov AX, 19          (* k[n]:=19               *)
   Mov [BX+DI], AX     (*------------------------*)
WhileLabel3:
   Mov BX, n           (* while(i[n]>199)        *)
   Mov AX, [BX+SI]     (*                        *)
   Cmp AX, 199         (*                        *)
   Jbe EndWhileLabei3  (*------------------------*)
   Mov AX, [BX+SI]     (* i[n]                   *)
   Add AX, [BX+DI]     (* i[n]+k[n]              *)
   Xor DX, DX          (*------------------------*)
   Mov CX, 11          (* (i[n]+k[n] div 11)=5   *)
   Div CX              (*                        *)
   Cmp AX, 5           (*                        *)
   Jne ElseLabel3      (* then                   *)
   Mov AX, [BX+DI]     (* i[n]:=i[n]-k[n]        *)
   Sub [BX+SI], AX     (*                        *)
   Jmp EndIfLabel3     (*------------------------*)
EiseLabei3:
   Mov CX, [BX+SI]     (* else                   *)
   Shr CX, 1           (* k[n]*k[n]+(i[n]div2)   *)
   Mov AX, [BX+DI]     (*                        *)
   Mov DX, AX          (*                        *)
   Mul DX              (*                        *)
   Add AX, CX          (*                        *)
   Mov [BX+DI], AX     (*                        *)
EndIfLabel3:
   Jmp WhileLabei3
EndWhileLabel3:
end;
(*=============================================*)
(* Базовая индексная адресация со смещением*)
asm
   Mov SI, offset I        (*------------------*)
   Mov DI, offset k        (*                  *)
   Mov AX, 34              (* n:=34            *)
   Mov n, AX               (*------------------*)
   Mov AX, 587             (* i[n]:=587        *)
   Mov BX, n               (*                  *)
   Shl BX, 1               (*                  *)
   Mov [BX+SI+2], AX       (*------------------*)
   Mov AX, 19              (* k[n]:=19         *)
   Mov [BX+DI+2], AX       (*------------------*)
```

```
WhileLabel4:
   Mov BX, n                (* while(i[n]>199)  *)
   Mov AX, [BX+SI+4]        (*                  *)
   Cmp AX, 199              (*                  *)
   Jbe EndWhileLabel4       (*------------------*)
   Mov AX, [BX+SI+2]        (* i[n]             *)
   Add AX, [BX+DI+2]        (* i[nl+k[n]        *)
   Xor DX, DX               (*------------------*)
   Mov CX, 11               (*[i[n]+k[n]div11)=5*)
   Div CX             (*                       *)
   Cmp AX, 5          (*-----------------------*)
   Jne ElseLabel4     (* then                  *)
   Mov AX, [BX+DI+2]  (* i[n]:=i[n]-k[n]       *)
   Sub [BX+SI+2], AX  (*                       *)
   Jmp EndIfLabel4    (*-----------------------*)
ElseLabel4:
   Mov CX, [BX+SI+2]  (* else                  *)
   Shr CX, 1          (* k[n]*k[n]+(i[n]div2)  *)
   Mov AX, [BX+DI+2]  (*                       *)
   Mov DX, AX         (*                       *)
   Mul DX             (*                       *)
   Add AX, CX         (*                       *)
   Mov [BX+DI+2], AX  (*                       *)
EndIfLabel4:
   Jmp WhileLabel4
EndWhileLabel4:
end;
(*=========================================*)
```

Варианты индивидуальных заданий к контрольной работе № 2

1.

```
N=11
a[N div 2]:= $AFA;
b[N mod 7]:= $AA;
for i:=N downto 1 do
  if ((b[i] div 4)=3) or (b[i] mod 100=0)
    then a[i]:=a[i]+b[i]
    else b[i]:=a[i]+b[i];
```

2.

```
N=12
a[N div 3]:= $CDE;
b[N mod 4]:= $C83;
i:=1;
while i<N do
begin
  if ((a[i] div b[i]) mod 13) = 2
    then a[i]:=a[i]+b[i]
    else b[i]:=b[i]-a[i];
  i:=i+1;
end;
```

3.

```
N=13
a[N div 4]:= $C11;
b[N mod 2]:= $F03;
i:=N;
repeat
  if((a[i] - b[i]) mod 9) < 5
    then a[i]:=a[i]-(b[i]+17)
    else b[i]:=b[i]-a[i];
  i:=i-1;
until i<1;
```

4.

```
N=14
i:=N;
a[N div 9]:= $0CA;
b[N mod 3]:= $F00;
while i>=1 do
begin
  if ((a[i] + b[i]) div 7 + (a[i] mod 9)) = 2
    then a[i]:=a[i]+b[i]*2
    else b[i]:=b[i]+b[i]*2;
  i:=i-1;
end;
```

5.

```
N=15
a[N div 2]:= $0FA;
b[N mod 7]:= $1AA;
for i:=N downto 1 do
  if ((b[i] div 7)=2) xor ((b[i] mod 27) = 2)
```

```
    then a[i]:=a[i] - b[i]
    else b[i]:=a[i]+b[i] div b[i];
```

6.

```
N=16
a[N div 6]:= $C11;
b[N mod 3]:= $003;
i:=1;
repeat
  if ((a[i] + b[i]) or 17) = 3
  then a[i]:=a[i]- (a[i] div b[i])
  else b[i]:=b[i]- a[i];
  inc(i);
until i>N;
```

7.

```
N=17
a[N div 5]:= $00E;
b[N mod 8]:= $F21;
i:=N;
repeat
  if ((a[i] + b[i]) mod 5) <2
    then a[i]:=a[i] div 12
    else b[i]:=b[i]+ 3*a[i];
  dec(i);
until i<1;
```

8.

```
N=18
a[N div 2]:= 1234;
b[N mod 10]:= 4321;
i:=1;
repeat
  if ((a[i] + a[i] - b[i]) or 9) = 3
    then a[i]:=a[i]*(a[i] mod b[i])
    else b[i]:=b[i]-a[i];
  i:=i+1;
until i>N;
```

9.

```
N=19
a[N div 7]:= $0F1;
b[N mod 4]:= $11C;
for i:=1 to N do
```

```
  if ((b[i] mod 5)=1) or (b[i] div 130=0)
    then a[i]:=17*a[i] - 3*b[i]
    else b[i]:=3*a[i] + 17*b[i];
```

10.

```
N=20
i:=1;
b[N div 9]:= $0EE;
a[N mod 3]:= $100;
while i<=N do
begin
  if ((a[i] + b[i]) div 11) = 5
    then a[i]:=b[i]-(a[n] mod 2)
    else b[i]:=b[i]+a[i]+(a[n] div 2);
  inc(i);
end;
```

11.

```
N=11
a[N div 3]:= $123;
b[N mod 4]:= $321;
i:=N;
repeat
  if ((a[i]*a[i] + b[i]) or 9) = 3
    then a[i]:=a[i]+(a[i]-b[n])
    else b[i]:=b[i]+b[n];
  dec(i);
until i<1;
```

12.

```
N=12
i:=N;
b[N div 9]:= $C07;
a[N mod 3]:= $E10;
while i>=1 do
begin
  if ((a[i]*a[i] + b[i]) or 5) = 3
    then a[i]:=a[i]*a[n]+(a[i]+b[i])
    else b[i]:=b[i]-a[i];
  dec(i);
end;
```

13.

```
N=13
```

```
i:=1;
b[N div 7]:= $1C0;
a[N mod 4]:= $1E0;
while i<=N do
begin
  if ((a[i] * b[i]) mod 9) = 4
    then a[i]:=a[i]+b[n]
    else b[i]:=b[i]*b[i]-a[n];
  inc(i);
end;
```

14.

```
N=14
i:=1;
b[N div 7]:= $E01;
a[N mod 4]:= $CEC;
while i<=N do
begin
  if ((a[i] + b[i] div 2) mod 11) = 5
    then a[i]:=a[i]-b[i]
    else b[i]:=b[i]-b[n]+(a[i] div 2);
  inc(i);
end;
```

15.

```
N=15
i:=1;
b[N div 2]:= $EEE;
a[N mod 6]:= $CCC;
while i<=N do
begin
  if ((a[i] + b[i]) mod 2) = 0
    then a[i]:=a[i]-b[n]
    else b[i]:=b[i]+b[n]+(a[i] div 4);
  inc(i);
end;
```

16.

```
N=16
i:=N;
b[N div 5]:= $AAA;
a[N mod 5]:= $BBB;
while i>=1 do
begin
```

```
  if ((a[i] + b[i]) div 3) <= 17
    then a[i]:=a[i]*a[n]+b[i]
    else b[i]:=b[i]*b[n]-a[n];
  i:=i-1;
end;
```

17.

```
N=17
a[N div 4]:= $CCC;
b[N mod 3]:= $DDD;
i:=N;
repeat
  if ((a[i] + b[i]*b[i]) div 5) <= 21
    then a[i]:=a[i]+1
    else b[i]:=b[i]+a[i];
  dec(i);
until i<1;
```

18.

```
N=18
a[N div 2]:= $10A;
b[N mod 10]:= $10D;
i:=1;
repeat
  if ((a[i] + b[i]) mod 4) < 2
    then a[i]:=a[i]+ a[i] mod b[i]
    else b[i]:=b[i]- b[n] +11*a[n];
  inc(i);
until i>N;
```

19.

```
N=19
a[N div 2]:= $AFE;
b[N mod 10]:= $DFD;
i:=1;
repeat
  if ((a[i]+b[n]+b[i]) mod 5) = 2
    then a[i]:=a[i]+11
    else b[i]:=b[i]+a[n];
  inc(i);
until i>N;
```

20.

```
N=20
a[N div 7]:= $111;
b[N mod 6]:= $012;
i:=N;
repeat
  if ((a[i]*a[i] + b[i]) div 4) <= 3
    then a[i]:=a[n]*(a[i]+7)
    else b[i]:=b[i] mod a[i];
  dec(i);
until i<1;
```

21.

```
N=11
a[N div 2]:= $F1E;
b[N mod 4]:= $EEE;
for i:=1 to N do
  if ((b[i] mod 7)<>2) xor ((b[i] div 27) <> 2)
    then a[i]:=a[i]-b[i]
    else a[i]:=a[i]+b[i] div a[n];
```

22.

```
N=12
a[N div 3]:= $111;
b[N mod 3]:= $111;
for i:=N downto 1 do
  if ((a[i] * b[i]) mod 4) < 2
    then a[i]:=a[i]+b[n]
    else b[i]:=b[i] div b[n]-a[n];
```

23.

```
N=13
a[N div 2]:= $FF0;
b[N mod 4]:= $CEC;
for i:=1 to N do
  if ((a[i]*a[n] + a[i]*b[i] + b[i]*b[n]) div 3)<2
    then a[i]:=a[i]+b[i]
    else b[i]:=b[i]+a[i];
```

24.

```
N=14
a[N div 5]:= $07E;
b[N mod 6]:= $0FF;
for i:=N downto 1 do
```

```
  if ((a[i] - b[i]) mod 9) < 4
    then a[i]:=a[i]*a[n]-b[i]
    else b[i]:=b[n]*b[i]+a[i];
```

25.

```
N=15
a[N div 3]:= $77F;
b[N mod 3]:= $88E;
for i:=1 to N do
  if ((b[i] div 5)=1) or (b[i] div 130=0)
    then a[i]:=a[i]-b[i]
    else a[i]:=a[i]+b[i];
```

26.

```
N=16
a[N div 2]:= $0F7;
b[N mod 2]:= $63C;
for i:=N downto 1 do
  if ((a[i]*a[n] + b[i]) or 9) <> 3
    then a[i]:=a[i]+(a[n]+b[i])
    else b[i]:=b[n]*a[i];
```

27.

```
N=17
i:=N;
b[N div 2]:= $567;
a[N mod 7]:= $765;
while i>=1 do
begin
  if ((a[i] + b[i]) div 11) = 5
    then a[i]:=a[i]-b[n]
    else b[i]:=b[i]+b[n]+(a[i] div 2);
  i:=i-1;
end;
```

28.

```
N=18
a[N div 7]:= $098;
b[N mod 3]:= $980;
i:=N;
repeat
  if ((b[i] xor 4)=3) or ((b[i] div 30) = 0)
    then a[i]:=a[n]*b[i]-b[n]*b[i]+a[i]
    else a[i]:=a[i]*2+b[n];
```

```
  dec(i);
until i<1;
```

29.

```
N=19
i:=N;
b[N div 5]:= $9AB;
a[N mod 4]:= $CDE;
while i>=1 do
begin
  if ((a[i] + b[i]) div 7+(a[i]*b[i] mod b[n]))<12
    then a[i]:=a[i]+b[n]*2
    else b[i]:=b[i]+a[n]*2;
  dec(i);
end;
```

30.

```
N=20
a[N div 2]:= $89A;
b[N mod 3]:= $9AB;
i:=N;
repeat
  if ((a[i] + b[i])mod7 + (a[i]*b[i] div a[n]))= 2
    then a[i]:=(a[i] div 3)+b[n]
    else b[i]:=b[i]*2+(a[n] div 2);
  dec(i);
until i<1;
```

3.3 Контрольная работа № 3

Тема: Исследование сложных структур данных на встроенном ассемблере языка TPascal v7.0

Продолжительность — 6 часов.

Максимальный рейтинг — 10 баллов.

Теоретическая часть

Каждое устройство ввода/вывода, каждое системное устройство имеет один или несколько регистров, доступ к которым осуществляется через адресное устройство ввода/вывода. Эти регистры имеют разрядность 8,16 или 32 бит. Адресное пространство ввода/вывода физически независимо от пространства опера-

тивной памяти и имеет ограниченный объем, составляющий 2^{16}, или 65536 адресов ввода/вывода. Таким образом, понятие порта ввода-вывода можно определить как 8, 16 или 32-разрядный аппаратный регистр, имеющий определенный адрес в адресном пространстве ввода/вывода.

Вся работа системы с устройствами на самом низком уровне выполняется с использованием портов ввода-вывода. На рис. 1 показана упрощенная, концептуальная схема управления оборудованием компьютера.

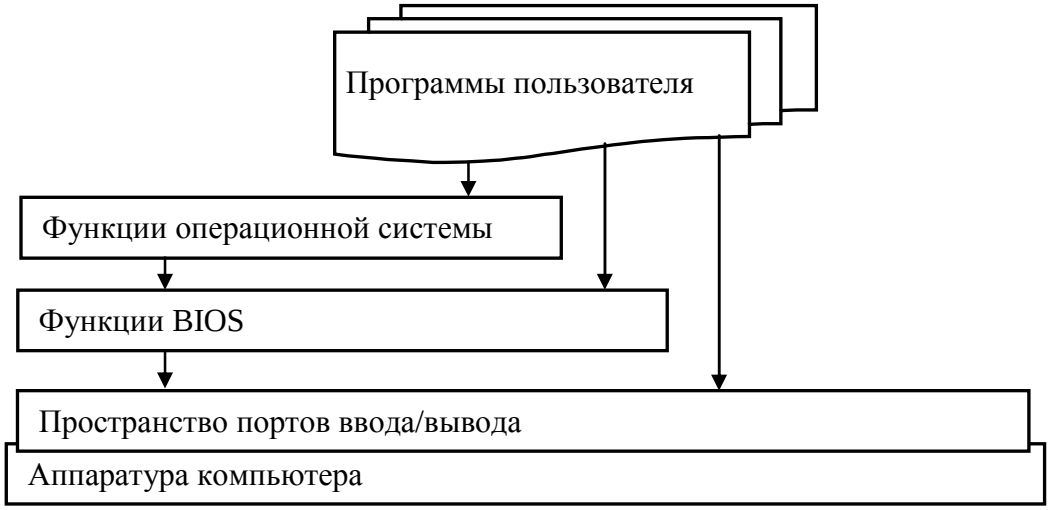

Рис. 1 — Схема управления оборудованием компьютера

Самым нижним уровнем является уровень BIOS, на котором работа с оборудованием ведется напрямую через порты. Тем самым реализуется концепция независимости от оборудования. При замене оборудования необходимо будет лишь подправить соответствующие функции BIOS, переориентировав их на новые адреса и логику работы портов.

Принципиально управлять устройствами напрямую через порты несложно. Сведения о номерах портов, их разрядности, формате управляющей информации приводятся в техническом описании на устройство. Необходимо знать лишь конечную цель своих действий, алгоритм, в соответствии, с которым работает устройство, и порядок программирования его портов. То есть, фактически, нужно знать, что и в какой последовательности нужно послать в порт (при записи на него) или считать из него (при чтении) и как следует трактовать эту информацию. Для этого

достаточно всего двух команд, присутствующих в системе команд микропроцессора:

in аккумулятор, номер_порта — ввод в аккумулятор из порта с номером номер_порта;

out порт, аккумулятор — вывод содержимого аккумулятора в порт с номером номер_порта.

Напомним еще раз способы задания операндов команды:

• Операнд задается неявно на микропрограммном уровне. В этом случае команда явно не содержит операндов. При выполнении команды некоторые объекты (регистры, флаги eflags и т.д.) используются по умолчанию. Например, команды cli и sti неявно работают с флагом прерывания if в регистре eflags, а команда xlat неявно обращается к регистру al и строке в памяти по адресу, определяемому парой регистров ds:bx.

• Операнд задается в самой команде (непосредственный операнд). Операнд находится в коде команды, т.е. является ее частью. Для хранения такого операнда в команде выделяется поле длиной до 32 бит. Непосредственный операнд может быть только вторым операндом (источником). Операнд-получатель может находиться либо в памяти, либо в регистре. Например, `mov ax, 0FFFFh` пересылает в регистр ax шестнадцатеричную константу `FFFF`. Команда `add sum, 2` складывает содержимое поля по адресу sum с целым числом 2 и записывает результат по месту первого операнда, т.е. в память.

• Операнд задается в одном из регистров. Регистровые операнды указываются именами регистров, в качестве которых могут использоваться:

– 32-разрядные регистры EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP;

– 16-разрядные регистры AX, BX, CX, DX, SI, DI, SP, BP;

– 8-разрядные регистры AH, AL, BH, BL, CH, CL, DH, DL;

– сегментные регистры CS, DS, SS, ES, FS, GS.

Например, команда `add ax, bx` складывает содержимое регистров ax и bx и записывает результат в bx; команда `dec si` уменьшает содержимое si на 1.

• Операнд располагается в памяти. Наиболее гибкий способ задания операндов, он реализует прямую и косвенную адресации.

Прямая адресация имеет следующие разновидности:

– относительная прямая адресация;

– абсолютная прямая адресация.

Косвенная адресация может быть одной из следующих разновидностей:

– косвенная базовая (регистровая) адресация;

– косвенная базовая (регистровая) адресация со смещением;

– косвенная индексная адресация со смещением;

– косвенная базовая индексная адресация;

– косвенная базовая индексная адресация со смещением.

• Операндом является порт ввода-вывода. Помимо адресного пространства оперативной памяти микропроцессор поддерживает адресное пространство ввода-вывода, которое используется для доступа к устройствам ввода-вывода. Объем адресного пространства ввода-вывода составляет 64 Кбайт. Для любого устройства компьютера в этом пространстве выделяются адреса. Конкретное значение адреса в пределах этого пространства называется портом ввода-вывода. Физически порту ввода-вывода ставится в соответствие аппаратный регистр, доступ к которому осуществляется с помощью специальных команд ассемблера in и out. Например, команда `in al, 60h` предписывает ввести байт из порта 60h.

Регистры, адресуемые с помощью порта ввода-вывода, могут иметь размерность 8, 16 или 32 бит, но для конкретного порта разрядность регистра фиксирована. При работе с регистрами ввода-вывода в качестве источника информации или получателя применяются регистры-аккумуляторы EAX, AX, AL. Выбор регистра определяется размерностью порта. Номер порта может задаваться как непосредственно в командах in и out, так и значением в регистре DX. Последний способ позволяет динамически определять номер порта в программе:

```
mov   dx, 20h     ; номер порта 20h в регистр dx
mov   al, 20h     ; значение 20h в регистр al
out   dx, al      ;вывести значение 20h в порт 20h.
```

Прерывание — инициируемый определенным образом процесс, временно переключающий микропроцессор на выполнение

другой программы с последующим возобновлением выполнения прерванной программы. Некоторые операционные системы используют механизм прерываний не только для обслуживания внешних устройств, но и для инициации выполнения собственных процедур. Таким образом, прерывания могут быть как внутренними, итак и внешними. Внешние прерывания вызываются внешними по отношению к микропроцессору событиями. На рис. 2 схематически изображена подсистема прерываний компьютера на базе микропроцессора Intel.

Из рис. 2 видно, что у микропроцессора есть два контакта INTR и NMI, на которых формируются внешние по отношению к процессору сигналы о том, что некоторое внешнее устройство просит уделить ему внимание.

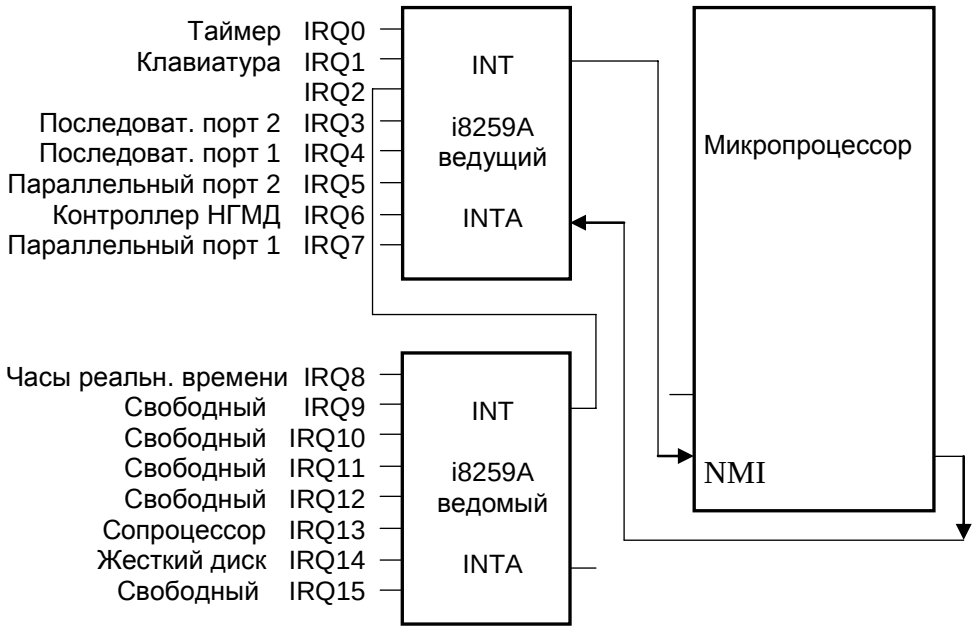

Рис. 2 — Подсистема прерываний на базе микропроцессора Intel

Вход ***INTR*** (***INTerrupt Request***) предназначен для фиксации запросов от различных периферийных устройств (системные часы, клавиатура, жесткий диск и т.д.) Вход ***NMI*** (***NonMaskable Interrupt***) — немаскируемое прерывание, используется для сообщения микропроцессору о некотором событии, требующем безотлагательной обработки, или катастрофической ошибке.

Микропроцессоры Intel имеют два режима работы — ***реальный*** и ***защищенный.*** В этих режимах обработка прерываний осуществляется принципиально разными методами.

К *аппаратным средствам* системы прерываний относятся:

• выводы микропроцессора:

• INTR — вывод для входного сигнала внешнего прерывания;

• INTA — вывод микропроцессора для выходного сигнала подтверждения получения сигнала микропроцессором;

• NMI — вывод для входного сигнала немаскируемого прерывания.

• микросхема программируемого контроллера прерываний 8259A, предназначенная для фиксирования сигналов прерываний от восьми различных внешних устройств.

• внешние устройства: таймер, клавиатура, магнитные диски и т.д.

К *программным средствам* системы прерываний реального режима относятся:

• таблица векторов прерываний, в которой содержатся указатели на процедуры обработки соответствующих прерываний.

• следующие флаги регистра флагов:

• IF (Interrupt Flag) — флаг прерывания; предназначен для маскирования (запрещения) аппаратных прерываний (по входу INTR); при обработке прерываний других типов флаг IF не учитывается. Если IF=1, микропроцессор обрабатывает внешние прерывания, если IF=0, микропроцессор игнорирует сигналы на входе INTR;

• TF (Trace Flag) — флаг трассировки

Задание

1. Повторить изученные в предыдущих курсах понятия: структура данных, символьная строка, массив, стек, очередь.

2. Используя встроенный ассемблер языка Turbo Pascal v7.0 написать и отладить программу в соответствии с индивидуальным заданием.

3. Привести алгоритм (в виде блок-схемы) работы программы и реализовать его на языке Turbo Pascal v7.0 и на встроенном ассемблере.

4. Определить время выполнения асемблерного кода и блока программы, выполненого на языке Turbo Pascal v7.0.

5. Сделать вывод о проделанной работе.

Примечания

1. Для оценки времени выполнения блоков программы использовать процедуру GetTime модуля DOS.

2. В случае если фрагмент программы выполняется быстрее 1/100 доли секунды, повторить его в цикле необходимое число раз, а полученную величину разделить на количество повторений.

3. Привести в отчете алгоритм (в виде блок-схемы) работы программы.

Пример к контрольной работе № 3

Рассмотрим пример реализации программного блока, выполненного на встроенном ассемблере (Listing 7), эквивалентный блоку, приведенному на листинге (Listing 6), написанному на языке Turbo Pascal и выполняющему следующее задание:

Вариант X: Составить словарь слов заданного фрагмента текста

В программе подготовлены массивы для хранения словаря, состоящие из 99 строк типа string[15] (слово не может быть длиннее, чем из 15 букв).

Listing 6

```
{================================================}
Type sltem = SET of 0..255;
sArray = Array [0..99] of String[15];
Var Index       :sltem;
Spisck, Spisok2 :sArray;
Texts,TmpStr    :String;
Texts2          :String;
IndOld,IndNew   :Byte;
Lengths,IndLast :Byte;
TmpCamp,Ferem   :Byte;
l,b             :Boolean;
...
Repeat
  Inc(i);
```

```
  b:=true;
  IndNew:= pos(' ',Texts);   {Нашли первый пробел}
  if IndNew=0
    then begin           {Заносим последнее слово}
      i:=true;
      IndNew:=Length(Texts)+1;
    end;
   Lengths:=IndNew-IndOld;{Определили длину слова}
   TmpStr:=Copy(Texts, IndOld, Lengths-1); {Скопировал}
   for j:=0 to k do begin
     TmpComp:=pos(TmpStr,Spisok[j]);
     if (TmpComp <> 0) or (TmpStr='')}
     then b:=false;
   end;
   if b then begin
     Spisok[k]:=TmpStr;      {Добавили в словарь}
     inc(У.};
   end;
   Delete(Texts,IndOld,Lengths);{Удал. из текста }
Until l;
...
```

На листинге (Listing 7) приведен алгоритмически эквивалентный блок, выполненный на ассемблере.

Listing 7

```
{=============================================}
asm
   PUSH  DS
   POP   ES
   LEA   DI, Texts2+1     {первый элемент Texts2}
   CLD                   {Просмотр строки вперед}
   MOV   CL, BYTE PTR Texts2 {заносим длину стр.}
   MOV   CH, 0
   MOV   AL, 32                {Искомый символ}
   XOR   DX, DX                   {Обнулили DX}
@@LOOPL:
   MOV   Perem, 0         {копирование разрешено}
   SCASB                        {Поиск в Texts2}
   PUSHF
   INC   DX               {В DX - длина слова+1}
```

```
   DEC   CX
   CMP   CX,0
   JE    @@NEXT                    {Конец строки}
   POPF
   JNE   @@LOOPL
@@NEXT:
   PUSH  AX
   PUSH  CX
   PUSH  SI
   PUSH  DI
   MOV   SI, DI    {SI указывает на начало слова}
   SUB   SI, DX
   DEC   DX
   MOV   CX, DX               {В CX длина слова}
   XOR   DX, DX
   MOV   AL, p                          { p*16 }
   MOV   AH, 16
   MUL   AH
   MOV   BX, AX
   LEA   DI, Spisok2[BX]+1 {DI указывает на нач.}
{-----------------------------------------------}
                    {проверка уникальности слова}
   CMP   CX, 0           {исключаем пустые слова}
   JZ    @@NEXTEL
   PUSH  BP
   MOV   BP, 0    {В BP количество слов в словаре}
@@SLED:
   MOV   BL, p
   MOV   BH, 0
   CMP   BP, BX
   JE    @@TECK
   PUSH  SI
   PUSH  DI
   PUSH  CX
   MOV   AX, BP                         { p*16 }
   MOV   AH, 16
   MUL   AH
   MOV   BX, AX
   LEA   DI, Spisok2[BX]+1 {DI указывает на нач.}
REPE CMPSB                      {[DS:SI]=[ES:DI]-?}
   JNE   @@TUDA
   CMP   CX, 0
   JNE   @@TUDA
   MOV   Perem, 1          {копирование запрещено}
```

```
@@TUDA:                                  {иначе разрешено}
   POP   CX
   POP   DI
   POP   SI
   INC   BP
   JMP   @@SLED
@@TECK:
   POP   BP
   CMP   Perem, 0
   JNE   @@NEXTEL
   MOV   BYTE PTR [DI]-1,CL
REP MOVSB   {Скопировали слово [DS:SI] -> [SS:DI]}
   INC   p    {Указатель на след. элемент массива}
@@NEXTEL:
   POP   DI
   POP   SI
   POP   CX
   POP   AX
   CMP   CX, 0
   JNE   @@LOOPL
@@Finish:
   NOP
    end;
```

Варианты индивидуальных заданий к контрольной работе № 3

1. Разработать процедуры сортировки строки байтов пузырьковым методом.

2. Дан массив N×M байт, необходимо получить две строки индексов (si и sj) элементов в порядке возрастания.

3. Разработать процедуры определения количества дней от рождества Христова по текущий день в формате DD/MM/YYYY (с учетом високосных лет).

4. Дан фрагмент текста, необходимо составить словарь слов (сортированный по алфавиту, строчные буквы).

5. Разработать процедуры для работы со строками, подобными строкам в Turbo Pascal. но имеющими максимальную длину 64 КБ, т.е. количество символов определяют первые два байта строки (аналоги DELETE. CONCAT, POS, COPY, INSERT).

6. Разработать процедуру сортировки N последовательностей байтов (строки байтов) методом вставки.

7. Дана строка символов S0, необходимо получить строку S1 из упорядоченных неповторяющихся символов исходной строки и строку S2, состоящую из количества соответствующих символов S1 в строке S0. (SO="sDaDFaa" -> Sl="DFas" S2="2131").

8. Определить ближайшее простое число к заданному числу N (Word).

9. Разработать процедуру умножения двух 16-байтных двоичных чисел в дополнительном коде с анализом переполнения.

10. Дан фрагмент текста, необходимо составить словарь слов (сортированный по алфавиту, строчные буквы не различать).

11. Текст задан массивом строк T, словарь задан массивом строк S ("слово — слово"), необходимо "перевести" текст T с помощью словаря S (слово, не найденное в словаре, выделять фигурными скобками).

12. Разработать процедуры поиска в строке символов минимального, максимального и среднего значения длины слова.

13. Разработать процедуру вычисления частного и остатка от деления двух полиномов (полиномы представлены в виде массивов коэффициентов типа BYTE).

14. Разработать процедуру перевода римских цифр (строка символов) в арабские (WORD).

15. Разработать процедуру транспонирования матрицы N×M (размер элемента L байтов).

16. Разработать процедуру циклического кодирования строки байтов с помощью ключа из N битов (кодирование функцией исключающее или).

17. Даны два числа A и B (Word), необходимо разложить каждое на простые множители и затем определить их наибольший общий делитель.

18. Разработать процедуры перевода десятичных целых чисел DWORD в HEX (и обратно).

19. Разработать процедуру перестановки букв в словах строки в обратном порядке ('abc defg'->'cba gfed')

20. Разработать процедуру вычисления количества секунд от начала года до заданного времени (формат времени MM/DD/HH/MM/SS).

4 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

4.1 Лабораторная работа № 1

Тема: Программирование драйвера на языке Turbo Pascal v7.0. Видеоадаптер VGA

Продолжительность — 6 часов.

Максимальный рейтинг — 10 баллов.

Теоретическая часть

В современных вычислительных системах реализовано раздельное обращение программных продуктов к аппаратным устройствам компьютера, т. е. практически никакие программы, реализуемые на компьютере, непосредственно к устройствам не обращаются. С контроллерами взаимодействуют лишь программы драйверов соответствующих устройств. Под понятием ***драйвер*** (***driver***) обычно понимается программный модуль, который:

– непосредственно управляет некоторым внешним устройством, взаимодействуя с его контроллером с помощью команд ввода-вывода компьютера;

– обрабатывает прерывания от контроллера устройства;

– предоставляет прикладному программисту естественный интерфейс работы с устройством, экранируя от него низкоуровневые детали управления устройством и организации его данных.

– взаимодействует с ядром ОС с помощью строго оговоренного интерфейса, описывающего формат передаваемых данных, способы включения драйвера в состав ОС, способы вызова драйвера, набор общих процедур подсистемы ввода-вывода, которыми драйвер может пользоваться, и т.д.

Таким образом в современных вычислительных системах реализуется идея многослойного подхода к организации программного обеспечения. Контроллер представляет собой нижний слой управления устройством, выполняющий операции в терминах блоков и агрегатов устройства. Драйвер выполняет более сложные операции по организации взаимодействия контроллера, ядра ОС и прикладной программы. В результате, прикладная программа работает с данными, преобразованными в естествен-

ную для программиста форму (файлы, символы, таблицы данных).

По-существу, драйвер обеспечивает ввод/вывод с устройства отдельных единиц информации. При работе с экраном или клавиатурой в качестве этой информации могут рассматриваться выводимые на экран символы.

Программная организация работы с видеоадаптером

Современные видеоконтроллеры поддерживают разнообразные текстовые и графические режимы. Текстовые режимы различают по разрешению (по числу символов по горизонтали и вертикали) и цветовой палитре (монохромный или 16-цветный режим). Для графических режимов основным признаком является количество одновременно отображаемых цветов (или иначе, количество двоичных разрядов, необходимых для хранения одной точки изображения):

- монохромный (1-битное кодирование цвета точки);
- 4-цветный CGA (2-битное кодирование);
- 16-цветный EGA/VGA (4-битное кодирование);
- 256-цветный SVGA (8-битное кодирование);
- HiColor (16-битное кодирование);
- TrueColor (24-битное или 32-битное кодирование).

Прежде чем начинать работу в каком-либо режиме, необходимо перевести систему в этот режим.

• ***Внимание!*** Не рекомендуется ставить эксперименты по прямому управлению режимами работы видеоконтроллера через его регистры: некорректная установка его параметров может вывести из строя монитор!

Для видеоконтроллеров разделение операций на программные и аппаратные особенно жестко: любые переключения режимов должны выполняться при помощи функций BIOS видеоконтроллера или фирменных драйверов, а выводить информацию можно напрямую в видеопамять. Устанавливать необходимый видеорежим рекомендуется один раз при запуске прикладной программы — на весь период ее выполнения. Переключать режимы в процессе работы крайне нежелательно. Существует два основных способа программной установки видеорежима: с помощью функций VGA BIOS и с помощью функций VESA BIOS.

Функции VGA BIOS

В настоящее время графические режимы VGA сильно устарели, но текстовые продолжают успешно применяться, поэтому интерес представляет только подгруппа текстовых функций из стандартного набора VGA BIOS:

- установка видеорежима;
- управление положением и размером курсора;
- переключение видеостраниц;
- управление шрифтами.

Для вызова функций VGA BIOS используется прерывание Int 10h. Набор функций очень большой, но устаревший. Рассмотрим лишь наиболее распространимые из них.

- Int 10h, функция 00h: установить видеорежим.
- Int 10h, функция 00h: установить видеорежим.
- Int 10h, функция 01h: установить размер курсора.
- Int 10h, функция 02h: установить позицию курсора.
- Int 10h, функция 03h: получить позицию и размер курсора.
- Int 10h, функция 05h: установить видеостраницу.
- Int 10h, функция 10h, подфункция 00h: установить один регистр палитры.
- Int 10h, функция 10h, подфункция 01h: установить цвет рамки экрана.
- Int 10h, функция 10h, подфункция 02h: установить все регистры палитры.
- Int 10h, функция 10h, подфункция 03h: переключить бит атрибута "мерцание/яркость".
- Int 10h, функция 10h, подфункция 07h: прочитать один регистр палитры.
- Int 10h, функция 10h, подфункция 08h: прочитать один регистр палитры.
- Int 10h, функция 10h, подфункция 09h: прочитать все регистры палитры.
- Int 10h, функция 10h, подфункция 10h: установить один регистр ЦАП.
- Int 10h, функция 10h, подфункция 12h: перезагрузить группу регистров ЦАП.
- Int 10h, функция 10h, подфункция 15h: прочитать один регистр ЦАП.

• Int 10h, функция 10h, подфункция 17h: прочитать группу регистров ЦАП.

• Int 10h, функция 11h, подфункция 00h: загрузить шрифт пользователя для текстового видеорежима.

Функции VESA BIOS

Фирмы-изготовители поставляют драйверы, в основном, только для Windows, поэтому для остальных ОС возникла необходимость в стандартизации операций по работе с видеоконтроллерами. Эту работу выполнила ассоциация VESA (Video Electronics Standard Association), полное описание стандарта VESA 3.0 на английском языке можно найти на сервере ассоциации (www.vesa.org).

Обращение к VESA BIOS осуществляется по прерыванию 10h с номером функции 4Fh. После выполнения этого вызова в регистре AX будет возвращен код результата (статус возврата)

• Int 10h, функция 4Fh, подфункция 00h: получить информацию о версии VESA BIOS.

• Int 10h, функция 4Fh, подфункция 01h: получить информацию о параметрах видеорежима.

• Int 10h, функция 4Fh, подфункция 02h: установить видеорежим с заданным номером.

• Int 10h, функция 4Fh, подфункция 03h: определить код текущего видеорежима.

• Int 10h, функция 4Fh, подфункция 04h: сохранить или восстановить состояние видеоконтроллера.

• Int 10h, функция 4Fh, подфункция 05h: управление окнами видеопамяти.

• Int 10h, функция 4Fh, подфункция 06h: получить или установить длину логической строки развертки.

• Int 10h, функция 4Fh, подфункция 07h: получить или установить координаты левого верхнего угла экрана.

• Int 10h, функция 4Fh, подфункция 08h: получить или изменить формат регистров палитры.

• Int 10h, функция 4Fh, подфункция 09h: сохранить или изменить содержимое регистров ЦАП.

Регистры видеоконтроллера. Необходимость непосредственной работы с регистрами видеоконтроллера реально возникает

только после перехода в защищенный режим работы процессора *Intel x86*, когда недоступны функции BIOS. Все регистры видеоконтроллера являются 8-разрядными. В состав стандартного контроллера VGA-типа входит шесть групп управляющих регистров:

- внешние регистры;
- регистры контроллера электронно-лучевой трубки (ЭЛТ);
- регистры синхронизатора;
- регистры графического контроллера;
- регистры контроллера атрибутов;
- регистры ЦАП VGA.

В настоящее время изделия различных изготовителей совершенно несовместимы друг с другом на уровне регистров. Но с целью сохранения совместимости с устаревшим ПО, современные видеоадаптеры имитируют работу контроллера VGA в области некоторых (не всех) основных регистров.

Работа в текстовом режиме видеоконтроллера. Текстовый режим до сих пор широко применяется (например, при загрузке: 80×25, 16-цветный, код 03h), т.к. для многих задач он является вполне достаточным, простым, надежным и обладает невысокими требованиями к аппаратуре. Основные операции в текстовом режиме: вывод символов, управление курсором и загрузка шрифта.

Видеопамять в текстовом режиме организована следующим образом: на каждый символ приходится по два байта информации, причем первый байт хранит ASCII-код символа, а второй — цвет символа и цвет фона знакоместа этого символа. В этом режиме для видеопамяти выделено окно размером 32 Кбайт в первом мегабайте адресного пространства процессора, начальный линейный адрес окна `B8000h`.

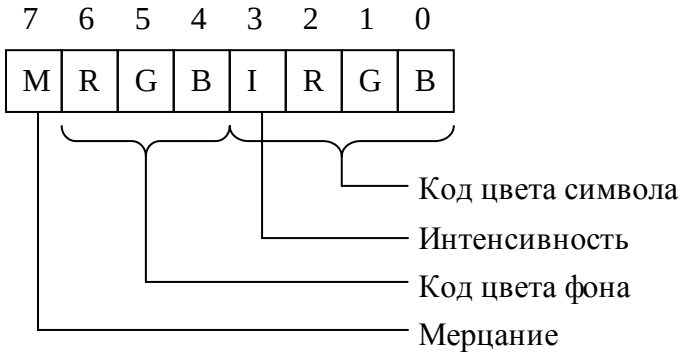

Рис. 3 — Формат байта цвета символа и фона в текстовом режиме

Для вывода символа в видеопамять обычно используется дополнительный сегментный регистр данных ES, в который перед началом записи в видеопамять нужно занести число, равное абсолютному начальному адресу буфера, поделенному на 16.

Таблица 1 — **Цветовые коды для текстового режима**

Код	Цвет
0000	Черный
0001	Синий
0010	Зеленый
0011	Бирюзовый
0100	Красный
0101	Фиолетовый
0110	Коричневый
0111	Светло серый

Код	Цвет
1000	Серый
1001	Ярко синий
1010	Ярко зеленый
1011	Ярко бирюзовый
1100	Ярко красный
1101	Ярко фиолетовый
1110	Желтый
1111	Белый

Таблица 2 — **Знакоместо символов в текстовом 80×25 режиме**

	Столбец 0		Столбец 1		Столбец 2			Столбец 79	
Строка 0	Символ 0		Символ 1		Символ 2		…	Символ 79	
	код	цвет	код	цвет	код	цвет		код	цвет
Строка 1	Символ 80		Символ 81		Символ 82		…	Символ 159	
	код	цвет	код	цвет	код	цвет		код	цвет
Строка 2	Символ 160		Символ 161		Символ 162		…	Символ 239	
	код	цвет	код	цвет	код	цвет		код	цвет
	…		…		…		…	…	
Строка 24	Символ 1920		Символ 1921		Символ 1922		…	Символ 1999	
	код	цвет	код	цвет	код	цвет		код	цвет

Чтобы вывести символ в заданное знакоместо нужно помножить номер строки знакоместа на 160 (длину строки в байтах) и прибавить номер столбца знакоместа, после чего записать результат в индексный регистр. Далее, в соответствующий байт заносится ASCII-код символа с помощью косвенной адресации (относительно сегмента ES и избранного индексного регистра). При необходимости значение индексного регистра инкрементируется и в следующий байт записывается код цвета символа и фона.

Приведем пример кода программы, в котором осуществляется вывод символа «#» в 15-й позиции третьей строки экрана желтым цветом по синему фону.

Листинг 8

```
;Загрузить в ES адрес текстовой видеопамяти
mov     AX,0B800h
mov     ES,AX
;Умножить номер строки на длину строки в байтах
mov     AL,160
mov     AH,3
mul     AH
;Прибавить к произведению номер колонки
add     AX,15
;Переписать полученное смещ. в индексный регистр
mov     BX,AX
;Записать код символа в видеопамять
mov     byte ptr ES:[BX],'#'
;Записать код цвета в следующий байт
inc BX
mov     byte ptr ES:[BX],01Eh
```

Кроме вывода символов, в текстовом режиме часто применяется еще два вида операций — перемещение курсора и переключение видеостраниц. Управление курсором осуществляется при помощи регистров видеоконтроллера или функций BIOS.

Установить размер прямоугольника текстового курсора можно при помощи регистров начальной и конечной линии курсора или при помощи функции 01h прерывания Int 10h.

Перемещение курсора по тексту производится путем записи значения смещения курсора относительно начала видеопамяти в регистры старшего и младшего байт адреса курсора. Позиционировать курсор можно так же при помощи функции 02h прерывания Int 10h, но координаты курсора при вызове прерывания задаются в виде номера строки и колонки относительно начала видеостраницы.

Переключение видеостраниц осуществляется путем записи смещения левого верхнего угла видеостраницы относительно начала видеопамяти в регистры старшего и младшего байт начального адреса. Эту же операцию можно выполнить при помощи

функции 05h прерывания Int 10h. Видеоконтроллер в текстовом режиме обеспечивает 8 видеостраниц, но используется обычно только основная (нулевая) страница. Дополнительные страницы, как правило, используются для выдачи сообщений оператору, не разрушая текст на основной странице.

Работа в графическом режиме

Современные видеоконтроллеры отличаются от VGA-контроллеров тем, что обеспечивают работу с высокими разрешениями и позволяют использовать линейную адресацию видеопамяти.

В 256-цветных режимах каждой точке изображения на экране соответствует один байт видеопамяти, в который записывается код цвета точки. Этот код не используется непосредственно, а служит индексом в специальном массиве, содержащем 256 строк по 3 элемента — таблице цветов ЦАП. Каждый из трех элементов задает интенсивность одного из основных цветов (RGB) электронно-лучевой трубки (ЭЛТ). Значения интенсивностей, выбранные из строки, соответствующей хранящемуся в видеопамяти коду, передаются в ЦАП.

Видеопамять адресуется на экран слева направо и сверху вниз как показано в табл. 3.

Таблица 3 — **Отображение видеопамяти в 256-цветном режиме 640×480 точек**

	Столбец 0	Столбец 1	Столбец 2		Столбец 639
Строка 0	байт 0	байт 1	байт 2	…	байт 639
Строка 1	байт 640	байт 641	байт 642	…	байт 1279
Строка 2	байт 1280	байт 1281	байт 1282	…	байт 1919
	…	…	…	…	…
Строка 479	байт 306560	байт 306561	байт 306562	…	байт 307199

В режимах группы ***DirectDraw*** (*HiColor* и *TrueColor*) информация поступает на ЦАП непосредственно из видеопамяти. Соответственно красная, зеленая и синяя (RGB) составляющие цвета точки представлены отдельными полями в выделенной для хранения точки области видеопамяти (от 2 до 4 байт на точку).

В режимах ***HiColor*** точка кодируется 16-разрядным словом, двумя вариантами: HiColor15 (формат 1:5:5:5) и HiColor16 (формат 5:6:5) как показано на рис. 4.

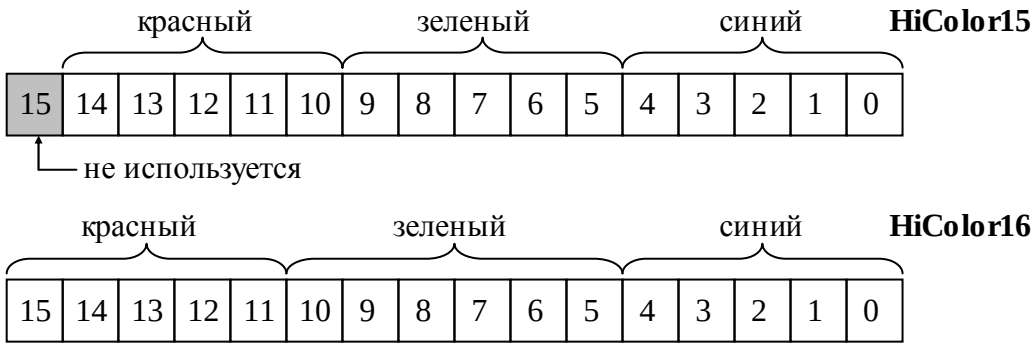

Рис. 4 — Форматы данных группы HiColor

В режимах TrueColor для хранения каждого компонента цвета точки выделено по одному байту видеопамяти. Существуют два варианта представления данных (см. рис. 5).

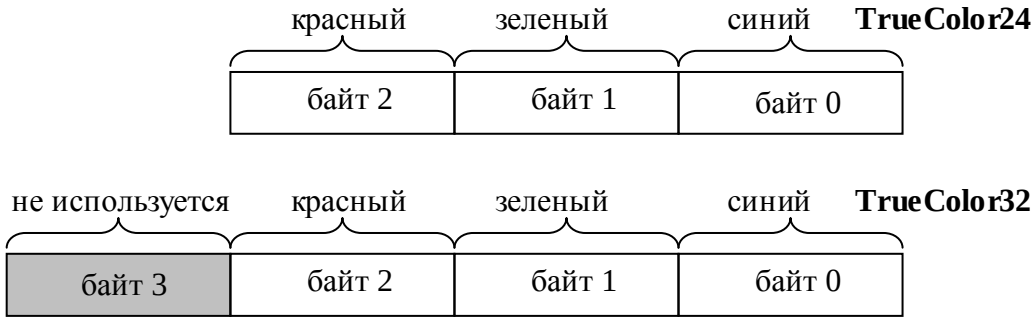

Рис. 5 — Форматы данных группы TrueColor

Дополнительный байт добавлен в режим TrueColor32 для выравнивания — на экран он не отображается. Наличие лишнего байта объясняется тем, что процессор может пересылать информацию лишь байтами, 16-разрядными или 32-разрядными словами, а так же тем, что передача данных словами приблизительно в три раза быстрее, чем побайтная.

Задание на лабораторную работу № 1

1. Изучить структуру и принцип работы видеоадаптера VGA (Video Graphics Adapter).

2. Изучить принципы работы, размещения в памяти и корректной выгрузки из памяти резидентной программы.

3. На встроенном ассемблере языка Turbo Pascal v7.0 запрограммировать блок процедур «драйвер», реализующий обращение к графическому адаптеру VGA. Написать вызывающую программу, обращающуюся к процедурам драйвера и выполняющую с его помощью простейшие графические операции.

4. Процедуры, демонстрирующие взаимодействие с адаптером, реализовать на уровнях прерывания BIOS (INT 10h, Fn00..Fn0F), изучить графические режимы VGA и вывод информации на экран непосредственно через видеопамять как для текстовых, так и для графических режимов.

5. Проиллюстрировать следующие функции графической системы:

В текстовом режиме:

– поместить курсор в указанную позицию экрана;

– задать размеры курсора;

– реализовать перевод строки;

– очистить экран;

– поместить символ в указанную позицию экрана;

– задать цвет символа и цвет фона;

– вывести строку символов начиная с указанной позиции экрана;

– переход в графический режим.

В графическом режиме:

– определить автоматически размеры экрана;

– поместить точку указанного цвета в определенную позицию;

– очистить экран;

– реализовать вывод символа (строки символов);

– переход в текстовый режим.

6. В соответствии с индивидуальным заданием реализовать тот или иной графический примитив в режиме VGA.

7. Сделать вывод по проделанной работе.

Примечания

1. Дополнительные баллы выставляются за исполнение резидентной версии драйвера (с определением его адреса в оперативной памяти, корректной загрузкой и освобождением памяти).

Варианты индивидуальных заданий к лабораторной работе № 1

В графическом режиме работы видеоадаптера VGA реализовать на встроенном ассемблере языка Turbo Pascal попиксельно следующие графические примитивы:

1. 10 окружностей разного цвета с одним центром и различными радиусами.
2. 8 расположенных в различных частях экрана разноцветных прямоугольников различного размера.
3. 12 расположенных в различных частях экрана разноцветных треугольников различного размера.
4. Две пятиконечные звезды различного радиуса и цвета.
5. Залитая (заполненная сплошным цветом) пятиконечная звезда.
6. Залитый (заполненный сплошным цветом) круг.
7. Две пересекающиеся окружности различного диаметра и цвета с залитым (заполненный сплошным цветом) пересечением.
8. Два заполненных пересекающихся круга различного диаметра и цвета с очищенным (под цвет фона) пересечением.
9. Два пересекающихся прямоугольника различного размера и цвета с залитым (заполненный сплошным цветом) пересечением.
10. Два пересекающихся треугольника различного размера и цвета с залитым (заполненный сплошным цветом) пересечением.
11. Два заполненных пересекающихся прямоугольника различного размера и цвета с очищенным (под цвет фона) пересечением.
12. Залитый (заполненный сплошным цветом) треугольник.
13. 10 окружностей разного цвета с различными центром и различными радиусами.
14. Вращающийся прямоугольник.
15. Перемещающийся круг
16. Вращающийся треугольник

17. Перемещающийся треугольник
18. Раздувающийся и сжимающийся треугольник
19. Раздувающийся и сжимающийся круг
20. Раздувающийся и сжимающийся прямоугольник
21. Перемещающийся прямоугольник
22. 12 окружностей разного цвета с различными центром и одинаковыми радиусами.
23. Последовательно изменяющий цвет прямоугольник.
24. Последовательно изменяющий цвет круг
25. Последовательно изменяющий цвет треугольник
26. Вращающаяся пятиконечная звезда.
27. Раздувающаяся и сжимающаяся пятиконечная звезда.
28. Перемещающаяся пятиконечная звезда.
29. Последовательно изменяющая цвет пятиконечная звезда.
30. Вращающийся выпуклый шестиугольник.
31. Перемещающийся выпуклый шестиугольник.
32. Раздувающийся и сжимающийся выпуклый шестиугольник.
33. Последовательно изменяющий цвет выпуклый шестиугольник.
34. 5 выпуклых шестиугольников различного цвета и радиуса, с разными центрами.
35. 8 выпуклых шестиугольников различного цвета и радиуса, с одним центром.
36. Залитый (заполненный сплошным цветом) выпуклый шестиугольник.

4.2 Лабораторная работа № 2

Тема: Программы для работы с оперативной памятью MS-DOS

Продолжительность — 6 часов.

Максимальный рейтинг — 10 баллов.

Задание

1. Изучить структуру оперативной памяти MS-DOS (LOW, Extended (XMS), Expanded (EMS), High Memory Area (HMA), Upper Memory Blocks (UMB)).

2. В составе MS-DOS имеются функции Int 21h, позволяющие программам распределять, изменять размер и освобождать блоки памяти. Изучить их.

3. Изучить драйвера, позволяющие операционной системе MS-DOS работать с участками оперативной памяти большими 640 кБ (conventional memory).

4. Научиться правильно подключать (загружать/выгружать) изученные драйвера (config.sys, autoexec.bat).

5. Изучить утилиты выдающие информацию о состоянии оперативной памяти в MS DOS:

– CHECKIT v3.0.
– mem (утилита MS DOS)
– mi (Memory Info, пакет PCTOOLS v9.0)
– si (System Info, пакет NU v6.01).

6. Протестировать оперативную память компьютера с подключенными (отключенными) драйверами и провести сравнение.

7. Сделать выводы по проделанной работе.

Примечания

1. Отразить в отчете результаты работы (протоколы, информацию, выведенную на экран) изученных утилит.

4.3 Лабораторная работа № 3

Тема: Программа-обработчик клавиатуры IBM

Продолжительность — 6 часов.

Максимальный рейтинг — 10 баллов.

Теоретическая часть

Обработчик прерывания `INT 16h` выполняет несколько функций, связанных с обслуживанием клавиатуры. С помощью функций обслуживания клавиатуры можно выполнить ввод кода нажатой клавиши как с ожиданием нажатия, так и без ожидания. В последнем случае функция сразу после вызова возвращает код нажатой клавиши или признак того, что никакая клавиша не нажималась. Символы, введенные с клавиатуры, помещаются в специальный клавиатурный буфер. Функция ввода символа без ожидания нажатия на клавишу проверяет состояние буфера —

есть в нем символы, или нет. Если в буфере есть символы, первый помещенный в буфер символ возвращается программе. Этот символ затем может быть считан функцией ввода с ожиданием нажатия — фактически ожидания при этом не будет.

Для машин класса не ниже AT обработчик прерывания INT 16h выполняет и другие функции: установку задержки, запись символов в буфер клавиатуры, обслуживание расширенной клавиатуры.

Ввод информации с клавиатуры при помощи функций BIOS:

• Int 16h, функция 00h: прочитать символ с клавиатуры.

• Int 16h, функция 01h: получить состояние клавиатуры.

• Int 16h, функция 02h: получить состояние флагов клавиатуры.

• Int 16h, функция 03h: управление режимом автоповтора.

• Int 16h, функция 04h: включить/выключить звуковой сигнал клавиш.

• Int 16h, функция 05h: поместить символ в буфер клавиатуры.

• Int 16h, функция 10h: прочитать символ с расширенной клавиатуры.

• Int 16h, функция 11h: получить состояние расширенной клавиатуры.

• Int 16h, функция 12h: получить состояние флагов расширенной клавиатуры.

Задание

1. Изучить структуру и принцип работы клавиатуры.

2. На языке Turbo Pascal v7.0 с использованием встроенного ассемблера разработать программу, взаимодействующую с клавиатурой IBM.

3. Программа должна обрабатывать следующие возможности клавиатуры:

– ввод символа в режиме "эхо-ввода";

– ввод символа без "эхо-ввода" с последующим отображением на экране;

– циклический ввод строки символов в обоих режимах;

– отображение состояния индикаторов (NumLock, CapsLock, ScrollLock) и управление ими;

– динамическое отображение информации о состоянии сдвиговых клавиш (Shift, Alt, Control);

– нажатие функциональных клавиш (F1-F12);

– использование управляющих клавиш (Ins, Del, Home, End, PageUp, PageDown, Left, Right, Up, Down);

– обработка альтернативных (серых, калькуляторных) клавиш (NumLock).

4. Разработать подпрограммы демонстрирующие взаимодействие с адаптером на уровнях:

– прерывания DOS (INT 21h);

– прерывание BIOS (INT 16h);

– порты ввода-вывода (64h).

5. Сделать вывод по проделанной работе.

Примечания

1. Дополнительные баллы выставляются за исполнение резидентной версии драйвера (с определением его адреса в оперативной памяти, корректной загрузки и освобождением).

2. Дополнительные баллы выставляются за реализацию собственного обработчика аппаратного прерывания (INT 09H).

4.4 Лабораторная работа №4

Тема: Программирование с использованием команд математического сопроцессора

Продолжительность — 6 часов.

Максимальный рейтинг — 10 баллов.

Теоретическая часть

В этом разделе рассмотрим такое устройство, как арифметический сопроцессор *Intel 8087/80287/80387*. Он подключен непосредственно к центральному процессору и предназначен для выполнения операций с числами в формате с плавающей точкой (вещественные числа) и длинными целыми числами. Арифметический сопроцессор значительно (в десятки раз) ускоряет вычисления, связанные с вещественными числами. Он используется при вычислениях значений таких функций, как синус, косинус, тангенс, логарифмы и т.д.

Основная область применения арифметического процессора — научные расчеты и машинная графика. Большинство современных пакетов САПР (например, AutoCAD версии 10 и выше) задействуют ресурсы *8087* сопроцессора, и, при его отсутствии на компьютере, не функционируют. Современные процессоры (выше *80486*) содержат встроенный математический сопроцессор совместимый с *Intel 8087/80287/80387*.

Сопроцессор запускается центральным процессором. После запуска сопроцессор выполняет все вычисления самостоятельно и параллельно с работой центрального процессора. Если центральный процессор выдает очередную команду в момент времени, когда тот еще не закончил выполнение предыдущей команды, центральный процессор переводится в состояние ожидания. Если же сопроцессор ничем не занят, центральный процессор, выдав команду сопроцессору, продолжает свою работу, не дожидаясь завершения вычисления. Впрочем, есть специальные средства синхронизации (команда FWAIT).

Команды, предназначенные для выполнения сопроцессором, записываются в программе как обычные машинные команды центрального процессора, но все эти команды начинаются с байта, соответствующего команде центрального процессора ESC. Встретив такую команду, процессор передает ее на выполнение сопроцессору, а сам продолжает выполнение программы со следующей команды.

Ассемблерные мнемоники всех команд сопроцессора начинаются с буквы F, например: FADD, FDIV, FSUB и т.д. Команды сопроцессора могут адресоваться к операндам, аналогично обычным командам центрального процессора. Операндами могут быть либо данные, расположенные в основной памяти компьютера, либо внутренние регистры сопроцессора (они организованы в стек).

Возможны все виды адресации данных, используемые центральным процессором. Разумеется, что с помощью сопроцессора можно выполнять и простейшие арифметические операции — сложения, вычитания, умножения и деления как с числами в формате с плавающей точкой, так и с целыми числами.

Вещественные числа, использующиеся в научной записи в общем виде записываются:

ЗНАК×МАНТИССА×10 $^{\text{ЗНАК×ПОРЯДОК}}$, например: -1.035*10 $^{-15}$

Математический сопроцессор производит операции над десятибайтными числами, представляемыми в нормализованном виде.

В случае фиксированной разрядной сетки числа (для фиксированного количества цифр в числе) нормализованные числа имеют наименьшую погрешность округления и, следовательно, — наибольшую точность. Кроме того, нормализованное представление исключает неоднозначность (каждое число с плавающей точкой может быть представлено различными «ненормализованными» способами).

Сопроцессор *8087/80287/80387* может работать с вещественными числами в трех форматах:

– одинарной точности (4 байта);
– двойной точности (8 байт);
– расширенной точности (10 байт).

Хотя математический сопроцессор распознает форматы целых, упакованных десятичных и чисел с плавающей запятой различной длины, внутри него все величины хранятся в формате с плавающей запятой в регистровом стеке 8х80 бит в виде восьми десятибайтовых чисел.

зн	порядок	мантисса
1	8 бит	23 бита

зн	порядок	мантисса
1	11 бит	52 бита

зн	порядок	мантисса
1	15 бит	64 бита

В математических операциях могут быть как неявные ссылки на верхние элементы стека, так и явные на другие регистры. Статусный регистр (status register, 16 бит) содержит указатель вершины стека, флаги, идентифицирующие особые случаи (например, переполнение) и коды состояний, отражающие результат последней команды. Регистр управления (control register, 16 бит) содержит биты вариантов и масок, которые программист может

устанавливать для выбора алгоритма округления, интерпретации бесконечности, а также задания того, как будут обрабатываться особые случаи — сопроцессором или программно.

Задание

1. Изучить принцип функционирования и команды математического сопроцессора (Floating Point Unit FPU).
2. Реализовать фрагмент программы при помощи команд математического сопроцессора на встроенном ассемблере языка Turbo Pascal v7.0 в соответствии с индивидуальным заданием.
3. Проиллюстрировать эквивалентность созданного программного кода соответствующему фрагменту на языке высокого уровня.
4. Оценить скорости выполнения блока программы на языке Turbo Pascal v7.0 и встроенном ассемблере.
5. Сделать вывод по проделанной работе.

Примечание:

1. Переменные f, x, y, z определены в основной программе, тип — double или extended.
2. Исходные значения x,y и z подобрать с тем, чтобы в программе не требовалась нормализация аргументов команд сопроцессора.
3. Дополнительные баллы начисляются за разработку программы для произвольных значений x,y,z.

Пример

Расчет сложной функции

f:=(ln(x+exp(z*sin(y+z))))+(sin(x*exp(x+ln(y+z))))

с использованием команд математического сопроцессора на встроенном ассемблере языка Turbo Pascal приведен на листинге (Listing 15).

Listing 15

```
{=============================================}
...
VAR   f, p, r, result :DOUBLE;
CONST x :DOUBLE= 0.214;
      y :DOUBLE= 0.63;
```

```
      z :DIUBLE= 0.29;
...
{================================================}
asm
   fld y;                { y+z                    }
   fld z;
   fadd;
   fptan;                { sin(y+z)               }
   fxch st(1);
   fmul st(0),st(0);
   fxch st(1);
   fmul st(0),st(0);
   fadd st(0),st(1);
   fdiv st(1),st(0);
   fstp result;
   fsqrt;
   fld z;                { z*sin(y+z)             }
   fmul;
   fldl2e;               { exp(z*sin(y+z))        }
   fmul;
   f2xm1;
   fld1;
   fadd;
   fld x;                { x+exp(z*sin(y+z))      }
   fadd;
   fld1;                 { ln(x+exp(z*sin(y+z))) }
   fxch st(1);
   fyl2x;
   fldl2e;
   fdiv;
   fstp p;
   fld y;                { y+z                    }
   fld z;
   fmul;
   fld1;                 { ln(y*z)                }
   fxch st(1);
   fyl2x;
   fldl2e;
   fdiv;
   fld x;                { x+ln(y+z)              }
   fadd;
   fldl2e;               { exp(x+ln(y+z))         }
   fmul;
   f2xm1;
```

```
   fld1;
   fadd;
   fld x;                { x*exp(x+ln(y+z))       }
   fmul;
   fptan;                { sin(x*exp(x+ln(y+z))) }
   fxch st(1);
   fmul st,st(0);
   fxch st(1);
   fmul st,st(0);
   fadd st,st(1);
   fdiv st(1),st;
   fstp result;
   fsqrt;
   fld p;
   fadd;
   fstp p;
end;
{=============================================}
```

Варианты индивидуальных заданий к лабораторной работе № 4

1. f:=sin(x*exp(x+ln(y*z)))+sin(x/exp(x*ln(y+z)))
2. f:=sin(x/exp(x*ln(y+z)))+cos(x+ln(x+exp(y/z)))
3. f:=sin(x+exp(x*ln(y+z)))+th(x+exp(z*tg(y+z)))
4. f:=exp(x+sin(x*ln(y+z)))+exp(x+ln(x+cos(y/z)))
5. f:=exp(x+ln(x+cos(y/z)))+exp(x+tg(y*ln(x+z)))
6. f:=exp(x+tg(y*ln(x+z)))+sin(x+cos(x*ln(y+z)))
7. f:=cos(x+sin(y*exp(y+z)))+sin(x*exp(x+ln(y*z)))
8. f:=sin(x+cos(x*ln(y+z)))+exp(x+ln(z*sin(y+z)))
9. f:=exp(x+lg(z*sin(y+z)))+sin(x+cos(y*exp(y+z)))
10. f:=sin(x+cos(y*exp(y+z)))+exp(x*cos(x+ln(y*z)))
11. f:=sin(x*exp(x+ln(y*z)))+exp(x+sin(x*ln(y+z)))
12. f:=ln(x+exp(z*sin(y+z)))+cos(x+sin(y*exp(y+z)))
13. f:=exp(x+tg(z*sin(y*z)))+cos(x*ln(x+exp(y*z)))
14. f:=sin(x+exp(x*ln(y+z)))+cos(x+ln(x+exp(y/z)))
15.
f:=exp(x+cos(z/siny+z*ln(y)))+cos(x+exp(y*siny+z))
16. f:=sin(x/exp(x*ln(y+z)))+cos(x+ln(x+exp(y/z)))
17. f:=cos(x+ln(x+exp(y/z)))+tg(x+exp(y*ln(x+z)))
18. f:=exp(x+tg(z*sin(y*z)))+cos(x*ln(x+exp(y*z)))
19. f:=tg(x+exp(y*ln(x+z)))+exp(x+sin(x*ln(y+z)))
20. f:=exp(x+lg(z*sin(y+z)))+sin(x+cos(y*exp(y+z)))

```
21. f:=ln(x/exp(x*sin(y+z)))+exp(x+ln(x+cos(y/z)))
22. f:=tg(x+exp(y*ln(x+z)))+exp(x+sin(x*ln(y+z)))
23. f:=exp(x+sin(x*ln(y+z)))+sin(x/exp(x*ln(y+z)))
24. f:=ln(x+exp(z*sin(y+z)))+sin(x*exp(x+ln(y*z)))
25. f:=sin(x*exp(x+ln(y*z)))+exp(x+sin(x*ln(y+z)))
26. f:=sin(x+cos(y*exp(y+z)))+exp(x*cos(x+ln(y*z)))
27. f:=ln(x*exp(x+sin(y*z)))+ln(x/exp(x*sin(y+z)))
28. f:=exp(x*ln(x+sin(y*z)))-ln(x/tg(x*sin(y+z)))
29. f:=sin(x+exp(x*ln(y+z)))+th(x+exp(z*tg(y+z)))
30. f:=sin(x+cos(x*ln(y+z)))+exp(x+ln(z*sin(y+z)))
31. f:=sin(x/exp(x*ln(y+z)))+cos(x+ln(x+exp(y/z)))
32. f:=exp(x+sin(x*ln(y+z)))+sin(x/exp(x*ln(y+z)))
33. f:=ln(x+exp(z*sin(y+z)))+sin(x*exp(x+ln(y*z)))
34. f:=sin(x*exp(x+ln(y*z)))+exp(x+sin(x*ln(y+z)))
35. f:=ln(x/exp(x*sin(y+z)))+exp(x+ln(x+cos(y/z)))
36. f:=ln(x*exp(x+sin(y*z)))+ln(x/exp(x*sin(y+z)))
37. f:=exp(x+lg(z*sin(y+z)))+sin(x+cos(y*exp(y+z)))
38. f:=tg(x+exp(y*ln(x+z)))+exp(x+sin(x*ln(y+z)))
39. f:=sin(x+cos(y*exp(y+z)))+exp(x*cos(x+ln(y*z)))
40. f:=exp(x+sin(x*ln(y+z)))+ln(x+exp(z*sin(y+z)))
41. f:=cos(x+sin(y*exp(y+z)))+sin(x*exp(x+ln(y*z)))
42. f:=sin(x*exp(x+ln(y*z)))+exp(x+sin(x*ln(y+z)))
43. f:=exp(x+tg(y*ln(x+z)))+sin(x+cos(x*ln(y+z)))
44. f:=exp(x+tg(z*sin(y*z)))+cos(x*ln(x+exp(y*z)))
45. f:=sin(x+exp(x*ln(y+z)))+cos(x+ln(x+exp(y/z)))
46. f:=exp(x+lg(z*sin(y+z)))+sin(x+cos(y*exp(y+z)))
47. f:=exp(x+ln(x+cos(y/z)))+exp(x+tg(y*ln(x+z)))
48. f:=ln(x+exp(z*sin(y+z)))+sin(x*exp(x+ln(y*z)))
49. f:=tg(x+exp(y*ln(x+z)))+exp(x+sin(x*ln(y+z)))
```

4.5 Лабораторная работа № 5

Тема: Накопители FDD и HDD IBM PC

Продолжительность — 6 часов.

Максимальный рейтинг — 10 баллов.

Теоретическая часть

В современных персональных компьютерах наиболее распространена дисковая подсистема IDE/ATA. Стандарт SCSI и его последующие реализации Fast-SCSI и Wide-SCSI, также широко распространенные в современных системах (файловые серверы и

другие высокопроизводительные устройства), требуют отдельного рассмотрения.

• ***IDE/ATA.*** Спецификация IDE/ATA была предложена в качестве недорогой альтернативы интерфейсам ESDI и SCSI для персональных компьютеров семейств IBM PC XT/AT. Большинство функций контроллера реализовано непосредственно на плате дискового накопителя.

Функции контроллера перенесены на плату управления приводом диска и головок винчестера. Информация о геометрии диска (число головок, цилиндров и секторов) хранится в самом устройстве.

Как правило диски IDE имеют небольшую встроенную кэш-память (до 256Кб) и позволяют работать с фактором чередования 1:1 (дорожка может быть прочитана целиком за один оборот диска).

Хост-адаптер для подключения дисков IDE зачастую устанавливается на системной плате. Подключение устройств к хост-адаптеру осуществляется с помощью 40-проводного плоского кабеля, к которому можно присоединить два винчестера. Для корректной адресации устройств один из винчестеров должен быть установлен в режим Master (ведущий), другой — в режим Slave (ведомый). Режим работы диска задается с помощью перемычек, расположенных, как правило, около сигнального разъема винчестера.

• ***SCSI*** не накладывает никаких ограничений на связь между контроллером и периферийным устройством. Шину SCSI можно использовать для связи компьютера с несколькими периферийными устройствами (как внешними, так и внутренними). Более того, допускается совместное использование одного периферийного устройства несколькими компьютерами, подключенными к общей шине SCSI. Подключаемые к шине SCSI устройства могут играть роль ведущих (Initiator) или ведомых (Target), при этом одно и то же устройство может быть ведомым в одних случаях и ведущим — в других. Обмен между устройствами по магистрали SCSI происходит в соответствии с протоколом высокого уровня и адресация осуществляется на уровне логических блоков. Программы для работы со SCSI-устройствами не используют физические характеристики конкретного устройства (число головок,

цилиндров и т.п.), а имеют дело с логическими блоками, что дает возможность работы фактически со всеми блочными устройствами.

Для подключения устройств SCSI используется кабель с 50-контактными разъемами. Возможны как синфазная, так и дифференциальная (с помощью "токовой петли") передача данных по кабелю; при синфазной передаче длина кабеля может достигать 6 м, при дифференциальной — 25 м. Для гарантированной передачи сигналов по магистрали SCSI линию требуется согласовывать с помощью терминаторов, устанавливаемых по обоим концам шины SCSI.

Спецификация SCSI предусматривает подключение к шине до восьми устройств, однако с учетом того, что каждое устройство может содержать 8 логических блоков, а каждый блок — 256 подблоков, возможности расширения являются фактически неограниченными. Каждое подключаемое к шине SCSI устройство имеет свой идентификатор, устанавливаемый с помощью перемычек или переключателей непосредственно в устройстве. Идентификаторы позволяют адресовать устройства и задают их приоритет (чем больше значение идентификатора, тем выше приоритет устройства).

На протяжении последних лет интерфейс SCSI был существенно расширен — появились спецификации Fast-SCSI и Wide-SCSI, обеспечивающие более высокую скорость обмена данными с устройствами SCSI. В настоящее время интерфейс SCSI используется в основном в высокопроизводительных системах, предназначенных для коллективного использования (диски файловых серверов, сканеры и т.д.).

• ***ATA66, ATA100 и ATA133.*** На смену устаревшему ATA33 пришел сначала интерфейс ATA66 с максимальной пропускной способностью 66 Мбайт/с, а затем и интерфейс ATA100 с пропускной способностью 100 Мбайт/с. Несколько позднее был выпущен интерфейс ATA133 с максимальной пропускной способностью 133 Мбайт/с.

Стандарт ATA100 допускает использование максимального объема жесткого диска до 137 Гбайт, что связано с 28-битной адресацией сектора. Однако современные диски вплотную подошли к этому пределу — их максимальный объем составляет 120 Гбайт, поэтому дальнейшее увеличение емкости дисков в стан-

дарте ATA100 просто невозможно. В стандарте же ATA133 (и в этом его главное достоинство) используется 48-битная адресация сектора, что позволяет адресовать диски с невероятно большим объемом — 144 Пбайт (петабайт).

Казалось бы, стандарт ATA133 решил все проблемы и можно на этом успокоиться, но, увы, параллельная передача данных имеет свои ограничения. В этом смысле скорость интерфейса в 133 Мбайт/с является предельной, а значит, и у самого интерфейса отсутствует перспектива дальнейшего развития, то есть он не имеет потенциальной возможности для масштабирования.

Serial ATA. В первой версии стандарта Serial ATA (SATA 1.0) предусмотрена максимальная пропускная способность 150 Мбайт/с, а об ограничениях на размеры дисков можно просто забыть на ближайшие лет десять. В следующих версиях SATA предусматривается удвоение скорости передачи, то есть сначала будет 300, а затем и 600 Мбайт/с.

Как уже отмечалось, стандарт SATA подразумевает последовательную передачу данных, а потому в кабелях передачи данных используются всего две дифференциальные пары. Одна из них работает на передачу, а другая — на прием. Всего же в кабеле SATA допускается (опционально) использование семи проводников, три из которых «земля». Максимальная длина кабеля при этом составляет 1 м.

По сравнению с традиционным параллельным интерфейсом интерфейс Serial ATA имеет большую помехозащищенность и мало восприимчив к электромагнитным помехам благодаря использованию низкоуровневых дифференциальных сигналов.

На физическом уровне для передачи данных используется двухэтапное логическое кодирование 8b/10b. При логическом кодировании 8b/10b каждые 8 бит исходной последовательности заменяются на 10 бит в соответствии с определенными правилами. В результате для 256 возможных комбинаций из 8 входных бит получаем 1024 возможные комбинации для 10 выходных бит. Но разрешенными из этих 1024 комбинаций являются только 256, а остальные — запрещенными. Как правило, такая избыточность используется для того, чтобы повысить помехоустойчивость кодирования (если при приеме обнаруживается запрещенная последовательность, то распознается ошибка передачи). Кроме того,

незначительная избыточность улучшает спектральные характеристики сигнала, поскольку исключает возможность появления в цепочке передаваемых бит длинных последовательностей нулей и единиц. Также повышаются и самосинхронизирующие свойства кода.

Кроме логического двухэтапного кодирования, при передаче данных используется метод циклического избыточного контроля CRC-32 (Cyclic Redundancy Check). На физическом уровне используется потенциальный код NRZ (Non Return to Zero).

Другой особенностью стандарта SATA является организация взаимодействия между контроллером и диском по принципу «точка-точка» (peer-to-peer): к одному контроллеру можно подключить только один жесткий диск, поэтому каждому устройству стандарта SATA предоставляется вся полоса пропускания целиком.

К тому же в стандарте SATA предусмотрена поддержка технологии «hot swap» (использование дисков с горячей заменой).

Задание

1. Разобраться с принципами хранения информации во внешней памяти (на магнитных и оптических носителях, лентах и пр.).
2. Изучить структуру накопителей Floppy Disk Driver (FDD) и Hard Disk Driver (HDD), принципы их работы и хранения информации.
3. Освоить команды прерываний BIOS и ОС MS-DOS для работы с магнитными носителями.
4. Исследовать структуру файловой системы FAT16, MBR, Boot sector, Boot record, FAT.
5. Изучить утилиты для работы с дисковыми накопителями и FAT (NDD, DE, SD, FORMAT, SF).
6. Изучить при помощи утилиты DE структуру дискеты 1.44 МБ (BootRecord, FAT, Dir), его физические и логические параметры.
7. Смоделировать при помощи программы DE нарушения файловой структуры (lost cluster, cross-link, invalid name, invalid boot record, invalid dir и пр.), затем исправить их с помощью программы NDD.

8. Изучить возможности утилит работы с FDD и HDD (редактирование диска — DE, форматирование — format и SF, варианты «сжатия» (дефрагментации) диска — SD, «лечение» — NDD).

9. Сделать вывод по проделанной работе.

Примечания

1. В отчете по лабораторной работе необходимо, помимо ответа на вопросы, привести результаты работы (файлы-протоколы, информацию, выводимую на экран и т.п.) исследуемых утилит.

4.6 Лабораторная работа № 6

Тема: Файловая система FAT16. Работа с файлами и каталогами

Продолжительность — 6 часов.

Максимальный рейтинг — 10 баллов.

Теоретическая часть

Файловая система определяет принципы хранения данных на физическом носителе. Например, файловая система определяет, как должны сохраняться данные файла, какая информация (например, имя, дата создания и т.п.) о файле должна храниться и каким образом. Формат хранения данных определяет основные характеристики файловой системы.

При рассмотрении характеристик файловых систем важным понятием является понятие кластера. Кластер — это минимальный блок данных, размещаемый на носителе. Файловая система использует кластеры для более эффективного управления дисковым пространством. Размер кластера всегда кратен размеру сектора диска. Потенциальный недостаток кластеров большого размера — это менее эффективное использование дискового пространства, поскольку данные одного файла и каталога всегда выделяется целое число кластеров. Например, если размер кластера составляет 32 Кб, то файл размером 100 байт все равно займет на диске 32 Кб.

В настоящее время существует большое количество файловых систем, отличающихся друг от друга целевым использовани-

ем (например, ориентацией только на конкретный вид носителей) и различными характеристиками. В Windows XP, также как и в Windows Server 2003, поддерживаются следующие файловые системы:

• ***FAT*** (*File Allocation Table*) — файловая система, разработанная для MS-DOS и являющаяся основной для Windows 3.x и 9x. Windows XP и Windows Server 2003 поддерживают три разновидности FAT: FAT12, FAT16 и FAT32. Первые две обеспечивают совместимость со старыми ОС Microsoft. Кроме того, FAT12 используется как формат хранения данных на гибких дисках. FAT 32 — модифицированная версия FAT, используемая в Windows 95 OSR2, Windows 98 и Windows Millennium.

• ***NTFS*** (*Windows NT file system*) — файловая система, разработанная специально для Windows NT и унаследованная Windows 2000, Windows XP, Windows 2003.

• ***CDFS*** (*Compact Disk File System*) — файловая система компакт-дисков.

• ***UDF*** (*Universal Disk Format*) — универсальный формат дисков, используемый современными магнитооптическими накопителями и, прежде всего, технологией DVD.

У каждой системы есть свои полезные свойства, но возможности защиты и аудита систем различны. На выбор файловой системы оказывают влияние следующие факторы: цель, для которой предполагается использовать компьютер, аппаратная платформа, количество жестких дисков и их объем, требования к безопасности, используемые в системе приложения.

Файловые системы FAT12 и FAT16. Файловая система FAT (File Allocation Table) получила свое имя в соответствии с названием метода организации данных — таблицы распределения файлов. FAT (или FAT16) первоначально была ориентирована на небольшие диски и простые структуры каталога. Затем ее усовершенствовали для обеспечения работы с большими дисками и мощными персональными компьютерами.

Windows XP и Windows Server 2003 поддерживают файловую систему FAT по трем причинам:

• для возможности обновления операционной системы с прежних версий Windows;

• для совместимости с другими операционными системами при многовариантной загрузке;

• как формат гибких дисков.

В название каждой версии FAT входит число, которое указывает разрядность, применяемую для идентификации кластеров на диске. 12-разрядный идентификатор кластеров в FAT12 ограничивает размер дискового раздела 212 (4096) кластерами. В Windows используются кластеры размером от 512 байт до 8 Кб, так что размер тома FAT12 ограничен 32 Мб. Поэтому Windows использует FAT12 как формат 5,25- и 3,5-дюймовых дискет, способных хранить до 1,44 Мб данных.

FAT16 — за счет 16-разрядных идентификаторов кластеров — может адресовать до 216 (65 536) кластеров. В Windows размер кластера FAT16 варьируется от 512 байт до 64 Кб, поэтому размер тома FAT16 ограничен 4 Гб. Размер кластеров, используемых Windows, зависит от размера тома.

Размеры кластеров в FAT16 по умолчанию (в Windows):

Размер тома (Мб)	***Размер кластера***
0—32	512 байт
33—64	1 Кб
65—128	2 Кб
129—256	4 Кб
257—511	8 Кб
512—1023	16 Кб
1024—2047	32Кб
2048—4095	64Кб

Файловая система FAT не обеспечивает функций защиты данных и автоматического восстановления. Поэтому она используется, только если альтернативной системой на компьютере является MS-DOS или Windows 95/98, а также для передачи данных на гибких дисках. В остальном использовать FAT не рекомендуется.

Файловая система FAT32. Модифицированная версия FAT — FAT32 — позволяет создавать разделы большие, чем в FAT16, и применять кластеры меньшего размера, что приводит к более эффективному использованию дискового пространства. Впервые FAT32 появилась в Windows 95 OSR2. Она также поддерживается в Windows 98 и Windows Millennium.

FAT32 использует 32-разрядные идентификаторы кластеров, но при этом резервирует старшие 4 бита, так что эффективный размер идентификатора кластера составляет 28 бит. Поскольку максимальный размер кластеров FAT32 равен 32 Кб, теоретически FAT32 может работать с 8-терабайтными томами. Однако реализация FAT32 в Windows XP / Windows 2003 не позволяет создавать тома, превышающие 32 Гб, но ОС может задействовать существующие тома FAT32 любого размера.

Размер кластеров на томах FAT32 (по умолчанию):

Размер раздела	***Размер кластера (Кб)***
От 32 Мб до 8 Гб	4
8—16 Гб	8
16—32 Гб	16
32Гб	32

Помимо большего предельного числа кластеров преимуществом FAT32 перед FAT12 и FAT16 является следующее:

• место хранения корневого каталога FAT32 не ограничено предопределенной областью тома, поэтому его размер не ограничен;

• для большей надежности FAT32 хранит вторую копию загрузочного сектора.

Файловая система NTFS. Файловая система NTFS — наиболее надежная система, специально разработанная для Windows NT и усовершенствованная в более поздних версиях Windows.

NTFS использует 64-разрядные индексы кластеров. Это позволяет NTFS адресовать тома размером до 1б экзабайт (1б миллиардов Гб). Однако Windows XP ограничивает размеры томов NTFS до значений, при которых возможна адресация 32-разрядными кластерами, т. е. до 128 Тб (с использованием кластеров по 64 Кб).

Размер кластеров на томах NTFS:

Размер раздела	***Размер кластера***
512 Мб и менее	512 байт
513—1024 Мб	1 Кб
1025—2048 Мб	2 Кб
более 2048 Мб (2 Гб)	4 Кб

Одно из важнейших свойств NTFS — восстанавливаемость. При неожиданном сбое системы информация о структуре папок и файлов на томе FAT может быть утеряна. NTFS протоколирует все вносимые изменения. Это позволяет избежать разрушения данных о структуре тома (Однако при этом данные файлов в некоторых случаях могут быть утеряны). Благодаря возможности шифровать файлы и папки, и устанавливать запрет на доступ к ним использование файловой системы NTFS повышает безопасность компьютера. NTFS поддерживает ряд дополнительных, по сравнению с FAT возможностей. Основные из них перечислены ниже:

- защита файлов и каталогов;
- сжатие файлов;
- поддержка многопоточных файлов;
- отслеживание связей;
- дисковые квоты;
- шифрование;
- точки повторной обработки;
- точки соединения;
- теневые копии.

Группа дисковых функций MS-DOS

Классические функции для работы с дисками:

- Int 21h, функция 0Eh: сменить текущий логический диск.
- Int 21h, функция 19h: определить номер текущего дисковода.
- Int 21h, функция 1Ah: изменить адрес области обмена с диском.
- Int 21h, функция 2Fh: получить адрес области обмена с диском.
- Int 21h, функция 36h: определить объем свободного места на диске.

Улучшенные функции для работы с дисками.

- Int 21h, функция 39h: создать подкаталог.
- Int 21h, функция 3Ah: удалить подкаталог.
- Int 21h, функция 3Bh: перейти в другой каталог.
- Int 21h, функция 3Ch: создать файл.
- Int 21h, функция 3Dh: открыть существующий файл.
- Int 21h, функция 3Eh: закрыть файл.

• Int 21h, функция 3Fh: чтение информации из файла.

• Int 21h, функция 40h: запись информации в файл.

• Int 21h, функция 41h: удалить файл.

• Int 21h, функция 42h: изменить положение указателя файла.

• Int 21h, функция 43h, подфункция 00h: получить атрибуты файла.

• Int 21h, функция 43h, подфункция 01h: изменить атрибуты файла.

• Int 21h, функция 47h: определить имя текущего каталога на указанном устройстве.

• Int 21h, функция 4Eh: найти первый файл заданного типа.

• Int 21h, функция 4Fh: найти следующий файл.

• Int 21h, функция 56h: переименовать или переместить файл.

• Int 21h, функция 57h, подфункция 00h: получить время и дату создания файла.

• Int 21h, функция 57h, подфункция 01h: изменить время и дату создания файла.

• Int 21h, функция 59h: получить дополнительную информацию об ошибке.

• Int 21h, функция 5Ah: открыть существующий файл.

• Int 21h, функция 5Bh: создать новый файл.

Низкоуровневые дисковые функции DOS:

• Int 25h: абсолютное чтение секторов из разделов малого объема.

• Int 25h, функция FFFFh: абсолютное чтение секторов из разделов большого объема.

• Int 26h: абсолютная запись секторов в разделы малого объема.

• Int 26h, функция FFFFh: абсолютная запись секторов в разделы большого объема.

Прерывания BIOS для работы с дисками на низком уровне:

• Int 13h, функция 00h: сброс дисковой системы.

• Int 13h, функция 01h: определить текущее состояние дисковой системы.

• Int 13h, функция 02h: читать сектор.

• Int 13h, функция 03h: записать сектор.

• Int 13h, функция 04h: проверить правильность записи.

• Int 13h, функция 05h: форматировать дорожку гибкого диска.

• Int 13h, функция 08h: получить параметры дисковода.

• Int 13h, функция 0Dh: сброс контроллера жесткого диска.

• Int 13h, функция 10h: проверить готовность жесткого диска к работе.

• Int 13h, функция 11h: рекалибровка жесткого диска.

• Int 13h, функция 16h: проконтролировать смену гибкого диска.

• Int 13h, функция 18h: установить тип носителя для форматирования.

Задание

1. Разобраться с принципами хранения информации во внешней памяти (на магнитных и оптических носителях, лентах и пр.).

2. Изучить структуру накопителей Floppy disk driver (FDD) и Hard disk driver (HDD), принципы их работы и хранения информации.

3. Освоить команды прерываний BIOS и ОС MS-DOS для работы с магнитными носителями.

4. Исследовать структуру файловой системы FAT16, MBR, Boot sector, Boot record, FAT.

5. Используя встроенный ассемблер языка Turbo Pascal v7.0 написать программу, реализующую работу с файлами и каталогами FAT16 на FDD дискете в соответствии с индивидуальным заданием.

6. Сделать вывод по проделанной работе.

Примечания

1. При создании, редактировании, форматировании и пр. файлов, необходимых для иллюстрации работы программы пользоваться утилитами, изученными в Лабораторной работе №5.

2. Отразить в отчете алгоритм (в виде блок-схемы) работы программы.

Пример

Рассмотрим, в качестве примера, фрагменты программы, представленные на листинге (Listing 22), в которых производится создание файла и запись в него строковой информации.

Listing 22

```
;================================================
;...
;Создание нового файла
LEA   DX, File_Name     ; имя файла
MOV   AH, 3CH           ; функция создания файла
XOR   CX, CX            ; атрибуты файла
INT   21h               ; вызов прерывания
JC   Disk_Not_Found ; проверка на наличие дисковода
;...
;================================================
;Запись строки в файл
MOV   AH, 40h    ; ошибка операции поиска на диске
LEA   DX, Simple_Text  ; заносим информацию в файл
MOV   CX, StrLen       ; запись конца строки
INT   21h              ; вызов прерывания
JC    Error_Open_File  ; проверка на наличие файла
;...
```

В случае, когда встречается ошибка, DOS и BIOS устанавливают кери-флаг (CF) — флаг переноса в единицу (CF=1), таким образом, для проверки правильности выполняемых операций уместно использовать подпрограммы, эквивалентные приведенным ниже.

```
;================================================
;...
Disk_Not_Found:
MOV   AH, 9
LEA   DX, Not_Found      ; проверка кери флага (CF)
INT   21h
Exit
;================================================
```

```
;...
Error_Open_File:
MOV   AH, 9
LEA   DX, Error_File   ; проверка кери флага (CF)
INT   21h
.Exit
;...
```

Варианты индивидуальных заданий к лабораторной работе № 6

1. Создать программным способом файл на диске A:\ с именем «LAB1.TXT» и занести информацию (год вашего рождения и дату в двоично-десятичной форме).

2. Написать программу определения времени создания файла «LAB2.TXT», заранее созданного и расположенного на диске A:\ (с определением наличия файла).

3. Создать программным способом файл на диске A:\ в каталоге A:\LAB3 с расширением «.002» и записать фразу «Hello friend» в прямом и обратном порядке.

4. Изменить программно атрибуты файла «LAB4.TXT», расположенного в каталоге A:\LAB4\ и вывести их на экран (с определением наличия файла).

5. Создать программным способом файл на диске A:\ с именем «LAB3.TXT» и осуществить запись чисел от 1 до 9, каждое число пишется с новой строки.

6. Создать программным путем файл на диске A:\ в каталоге A:\LAB6 с именем «LAB6.16H» и записать свою фамилию и номер группы в шестнадцатеричной форме.

7. Создать программным способом файл на диске A:\ с именем «LAB7.02H» и записать свою фамилию и номер группы в двоичной форме.

8. Написать программу «обновления» всех файлов на диске A:\ в каталоге A:\LAB8 путем изменения времени создания файла.

9. Создать программным способом два файла на диске A:\ с именем «LAB9.NGR» и «LAB9.NAM», в первый файл занести номер группы, а во второй файл занести фамилию и дату своего рождения.

10. Создать программным способом файл на диске A:\ с именем «LAB10.STR» и занести в него 5 строк любой текстовой информации.

11. Создать программным способом пять файлов на диске A:\ и записать в каждом из них по одному числу.

12. Создать программным способом файл с именем «LAB12.MAT» и занести в него двухмерный массив вещественных чисел размером 10 ×10.

13. Создать программным способом файл с именем «LAB13.MAT» в каталоге A:\LAB13 и занести в него одномерный массив любых вещественных чисел размером 100 элементов.

14. Добавить программным путем строковую информацию в файл, уже существующий на диске A:\ в каталоге A:\LAB14 (с проверкой наличия файла).

15. Удалить программным путем существующий на диске A:\ файл из каталога A:\LAB15 (с проверкой наличия файла).

16. Создать программным способом файл на диске A:\ в каталоге A:\LAB16 с именем «LAB16.STR» и занести в него 16 строк любой символьной информации.

17. Заменить программным путем файл уже существующий на диске A:\ (с проверкой наличия файла) на вновь созданный файл с таким же именем, но иным содержанием.

18. Скопировать программным путем файл с диска A:\ (с проверкой наличия файла) на диск A:\ в каталог A:\LAB18\ в файл с именем «LAB18.TXT»/

19. Скопировать программным путем файл с диска A:\ из каталога A:\LAB19\ (с проверкой наличия файла) в корневой каталог диска A:\ в файл «LAB19.TXT».

20. Программным способом считать шестнадцатеричную информацию из файла, расположенного на диске A:\ и вывести на экран в двоичном виде.

21. Программным способом считать двоичную информацию из файла, расположенного на диске A:\ в каталоге A:\LAB21\ и вывести ее на экран в шестнадцатеричном виде.

22. Скопировать программным путем 3 файла (находящиеся на диске A:\) на диск A:\ в каталог A:\LAB22, после чего удалить исходные 3 файла с диска.

23. Осуществить программный поиск файла «LAB23.TXT» на диске A:\. Результат вывести на экран.

24. Программным путем открыть на диске A:\ файл «LAB24.TXT» и найти в нем определенное слово.

25. Программным способом отредактировать на диске A:\ файл «LAB25.TXT», путем удаление всех букв «А» из текста.

26. Программным путем отредактировать на диске A:\ файл «LAB26.TXT», путем замены буквы «А» на букву «О» в тексте.

27. Отредактировать программно на диске A:\ файл «LAB27.TXT», записывая все слова в обратном порядке.

28. Написать программу «форматирования» диска A:\ путем удаления всех файлов и обнуления содержимого кластеров.

29. Поменять метку диска A:\ программным способом.

30. Создать программным способом файл в каталоге A:\LAB30\ с именем «LAB30.TXT» и занести в него какую-либо текстовую информацию.

31. Написать программу, выводящую список всех файлов в уже существующем каталоге A:\LAB31\.

32. Записать в файл «LAB32.TXT» программным способом список всех файлов каталога A:\LAB32\ с указанием атрибутов.

33. Записать в файл «LAB33.TXT», программным способом создаваемый в каталоге A:\LAB33.txt список всех файлов корневого каталога диска D:\ (выдавать сообщение в случае отсутствия данного диска).

34. Записать в файл «LAB34.TXT», программным способом создаваемый в каталоге A:\LAB34.txt список всех файлов корневого каталога диска C:\ с указанием времени создания.

4.7 Лабораторная работа № 7

Тема: Подготовка компьютера к работе. Setup BIOS

Продолжительность — 6 часов.

Максимальный рейтинг — 10 баллов.

Теоретическая часть

BIOS — Базовая система ввода-вывода (Basic Input Output System) называется так потому, что включает в себя обширный набор программ ввода-вывода, благодаря которым операционная система и прикладные программы могут взаимодействовать с различными устройствами как самого компьютера, так и подключенными к нему. С одной стороны, BIOS можно рассматривать как составную часть аппаратных средств, с другой стороны, она является как бы одним из программных модулей операционной системы.

Большинство современных видеоадаптеров, а также некоторые контроллеры накопителей имеют собственную систему BIOS, которая обычно дополняет системную. Во многих случаях программы, входящие в конкретную BIOS, заменяют соответствующие программные модули основной BIOS. Вызов программ BIOS, как правило, осуществляется через программные или аппаратные прерывания.

BIOS, помимо программ взаимодействия с аппаратными средствами на физическом уровне, содержит программу тестирования при включении питания компьютера POST (Power–On-Self-Test, Самотестирование при включении питания компьютера). Тестируются основные компоненты, такие как процессор, память, вспомогательные микросхемы, приводы дисков, клавиатуру и видеоподсистему.

Система BIOS в компьютерах, неразрывно связана с SMOS RAM. Под этим понимается «неизменяемая» память, в которой хранится информация о текущих показаниях часов, значении времени для будильника, конфигурации компьютера: количестве памяти, типах накопителей и т.д. Именно в этой информации нуждаются программные модули системы BIOS. Своим названием SMOS RAM обязана тому, что эта память выполнена на основе КМОП-струкгур которые, как известно, отличаются малым энергопотреблением. Заметим, что КМОП-память энергонезависима только постольку, поскольку постоянно подпитывается, например, от аккумулятора, расположенного на системной плате, или батареи гальванических элементов, как правило, смонтированной на корпусе системного блока. Заметим, что большинство систем-

ных плат допускают питание КМОП-память как от встроенного, так и от внешнего источника.

Изменение установок в CMOS осуществляется через программу SETUP и замыканием (размыканием) соответствующих перемычек на системной плате. Назначение каждой из них указано в соответствующей документации.

Задание

1. Изучить конфигурационные параметры BIOS Setup.
2. Включая (Enabled) и отключая (Disabled) различные параметры Setup BIOS проиллюстрировать различия в работе ПК при тех или иных настройках Setup BIOS.
4. Сделать вывод по проделанной работе.

Примечание

1. В отчете описать все настроечные параметры Setup BIOS вашего компьютера.

4.8 Лабораторная работа № 8

Тема: Программирование порта последовательной передачи данных (COM-порт) персонального компьютера

Продолжительность — 6 часов.

Максимальный рейтинг — 10 баллов.

Теоретическая часть

Последовательный интерфейс для передачи данных в одну сторону использует одну сигнальную линию, по которой информационные биты передаются друг за другом последовательно. Такой способ передачи определяет название интерфейса и порта, его реализующего (***Serial Interface*** и ***Serial Port***). Последовательная передача данных может осуществляться в синхронном и асинхронном режимах.

При ***асинхронной передаче*** каждому байту предшествует старт-бит, сигнализирующий приемнику о начале очередной посылки, за которой следуют биты данных или бит паритета (контроля четности). Завершает посылку стоп-бит. Старт-бит (имеющий значение лог."0") следующего посланного байта может по-

сылаться в любой момент после окончания стоп-бита. Старт-бит обеспечивает механизм синхронизации приемника по сигналу от передатчика. Внутренний генератор синхронизации приемника использует счетчик-делитель опорной частоты, обнуляемый в момент приема начала старт-бита. Этот счетчик генерирует внутренние стробы, по которым приемник фиксирует последующие принимаемые биты. Формат асинхронной посылки позволяют выявить возможные ошибки передачи. Для асинхронного режима принят ряд стандартных скоростей обмена: 50, 75, 110, 150, 300, 600, 1200, 2400, 4800, 19200, 38400, 57600, 115200 бит/сек. Количество бит данных может составлять 5,6,7,8 бит. Количество стоп битов может быть 1, 1.5 или 2 бита. Асинхронный последовательный порт в PC реализуется с помощью COM-порта с использованием протокола RS-232C.

Синхронный режим передачи предполагает постоянную активность канала связи. Посылка начинается с синхробайта, за которым плотно следует поток информационных бит. Если у передатчика нет данных для передачи, он заполняет паузу непрерывной посылкой байтов синхронизации. При передаче больших массивов данных накладные расходы на синхронизацию в данном режиме будут ниже, чем в асинхронном. Однако в синхронном режиме необходима внешняя синхронизация приемника с передатчиком, поскольку даже малое отклонение частот приведет к быстро накапливающейся ошибке и искажению принимаемых данных. Внешняя синхронизация возможна либо с помощью отдельной линии передачи для передачи сигнала синхронизации, либо с использованием самосинхронизирующего кодирования данных, при котором на приемной стороне из принятого сигнала могут быть и импульсы синхронизации. В любом случае синхронный режим требует либо дорогих линий связи, либо дорогого оконеченного оборудования. Из синхронных адаптеров в настоящее время чаще всего применяются адаптеры интерфейса V.35.

Последовательный интерфейс на физическом уровне может иметь различные реализации, различающиеся способом передачи электрических сигналов. Существует ряд родственных международных стандартов: ***RS-232C, RS-432A, RS-422A, RS-485***. Несимметричные линии интерфейсов RS-232C, RS-432A имеют са-

мую низкую защищенность от синфазной помехи. Лучшие параметры имеет двухточечный интерфейс RS-422A и его магистральный (шинный) родственник RS-485, работающие на симметричных линиях связи. В них для каждого сигнала используются дифференциальные сигналы с отдельной (витой) парой проводников.

Электрический интерфейс. Стандарт RS-232C использует несимметричные передатчики и приемники — сигнал передается относительно общего провода. Интерфейс не обеспечивает гальванической развязки устройств. Логической единице соответствует уровень напряжения на входе приемника в диапазоне –12...–3 В, для линий управляющих сигналов это состояние называется ON ("включено"), для линий последовательных данных называется MARK. Логическому нулю соответствует напряжение в диапазоне +3...+12 В, для линий управляющих сигналов это состояние называется OFF ("выключено"), для линий последовательных данных называется SPACE. Между уровнями –3...+3 В имеется зона нечувствительности, обусловливающая гистерезис приемника: состояние линии будет считаться измененным только после пересечения соответствующего порога . Уровни сигналов на выходах передатчиков должны быть в диапазонах –12...–5 В и +5...+12 В для представления единицы и нуля соответственно. Разность потенциалов между схемными землями (SG) соединяемых устройств должна быть менее 2 В, при более высокой разности потенциалов возможно неверное восприятие сигналов. Интерфейс предполагает наличие защитного заземления для соединяемых устройств, если они оба питаются от сети переменного тока и имеют сетевые фильтры. Подключение и отключение интерфейсных кабелей устройств с автономным питанием (не питающихся от интерфейса, таких как, например, мышь) должно производиться при отключении питания. В противном случае разность не выровненных потенциалов устройств в момент коммутации (присоединения или отсоединения разъема) может оказаться приложенной к выходным или входным (что опаснее) цепям интерфейса и вывести из строя микросхемы.

COM-порт (*Communications Port*) обеспечивает асинхронный обмен по стандарту RS-232C. Компьютер может иметь до четырех последовательных портов COM 1-COM4. COM-порты

имеют внешние разъемы DB25P или DB9P, выведенные на заднюю панель компьютера.

COM-порты реализуются на микросхемах UART, совместимых с семейством 18250. Они занимают в пространстве ввода/вывода по 8 смежных 8-битных регистров и могут располагаться по стандартным базовым адресам 3F8h (COM1), 2F8h (COM2), 3E8h (COM3), 2E8h (COM4). Для портов COM3 и COM4 возможны альтернативные адреса 3EOh, 338h и 2EOh, 238h соответственно. Для PS/2 стандартными для портов COM3-COM8 являются адреса 3220h, 3228h,4220h, 4228h, 5220h и 5228h соответственно.

Порты могут вырабатывать аппаратные прерывания IRQ4 (обычно используются для COM1 и COM3) и IRQ3 (для COM2 и COM4).

Режим DMA при работе с COM-портами используют редко, поэтому в большинстве случаев каналы DMA порту не назначают.

Задание

1. Изучить принципы ввода-вывода информации при помощи последовательного COM-порта ПК.

2. Изучить формат последовательной асинхронной передачи данных, физический и электрический интерфейс COM-порта, программные интерфейсы COM-порта (RS-232C, RS-423, RS-422, RS-485).

3. Исследовать интерфейс RS-232C, команды и разъемы последовательного интерфейса, интерфейс "токовая петля". Функции BIOS для COM-портов. Методы конфигурирования, настройки и тестирования COM-портов.

4. На встроенном ассемблере языка Turbo Pascal v7.0 написать программу, иллюстрирующую передачу информации между ПК и любым периферийным устройством (мышь, принтер, модем и пр.), подключенным к COM-порту. Допускается реализация нуль-модемного соединения двух ПК через COM-порт.

5. Изменяя настройки соответствующего COM-порта в BIOS Setup продемонстрировать работоспособность (или неработоспособность) вашей программы при различных настройках BIOS Setup.

6. Сделать вывод по проделанной работе.

Примечания

1. Результаты работы программы при различных настройках BIOS Setup привести в отчете. В отчете привести так же алгоритм работы (блок-схему) программы.

2. Если никакого периферийного устройства, подключаемого к COM-порту нет в наличии, то допускается написание программы автотестирования COM-порта при помощи заглушки, на порт DB9 со следующей схемой соединения выводов:

2	–	3
1	–	6 + 4
7	–	8

Заглушка в этом случае выполняется студентом самостоятельно. Автотестирование в данном случае состоит в передаче данных в COM-порт и немедленном приеме этих же данных из него.

3. При реализации программы, имитирующей нуль-модемное соединение двух ПК через COM-порт, соединительный кабель выполняется студентом самостоятельно по схеме:

1+7	–	8
2	–	3
3	–	2
4	–	6
5	–	5
6	–	4
7+1	–	8
8	-	1+7

(1 и 7 контакты на разъемах соединены между собой; 9 не используется; экраны соединяются).

Пример

Приведенный на листинге Listing 23 фрагмент программного кода иллюстрирует работу с COM-портом ПК при помощи команд прерывания INT 14h.

Listing 23

```
{================================================}
{ получить один символ от внешнего устройства,    }
{ подключенного к COM-порту                       }
asm
   MOV   AX, 0
   MOV   AH, 02
   MOV   DX, 0
   MOV   DL, com_number   { номер COM-порта 0...3 }
   INT   14h
   MOV   sim, AL        { прочитана переменная sim }
   MOV   err, AH       { ошибка - в переменной err }
end;
...
{================================================}
{ получить состояние последовательного порта      }
asm
   MOV   AX, 0
   MOV   AH, 03
   MOV   DX, 0
   MOV   DL, com_number   { номер COM-порта 0...3 }
   INT   14h
   MOV   lin, AL          { состояние линии }
   MOV   dev, AH          { состояние устройства }
end;
...
{================================================}
```

4.9 Лабораторная работа № 9

Тема: Программирование параллельного порта (LPT-порт) персонального компьютера

Продолжительность — 6 часов.

Максимальный рейтинг — 10 баллов.

Теоретическая часть

Параллельные интерфейсы характеризуются тем, что в них для передачи каждого бита в слове используются отдельные сигнальные линии, и байт (слово) передается за один цикл. Параллельные интерфейсы используют логические уровни ТТЛ (транзисторно-транзисторной логики), что ограничивает длину кабеля из-за невысокой помехозащищенности ТТЛ-интерфейса. Гальваническая развязка отсутствует. Передача данных может быть как однонаправленной, так и двунаправленной.

К параллельным интерфейсам, помимо шин, относят LPT-порт, используемый, как правило, для подключения принтеров, отсюда и название LPT-порт (line printer — построчный принтер).

Стандарт IEEE 1284-1994

Стандарт на параллельный интерфейс IEEE 1284, принятый в 1994 году, определяет 5 режимов обмена данными, метод согласования режима, физический (кабели, соединители) и электрический интерфейсы (драйверы, приемники, окончание линии, импеданс). Согласно IEEE 1284, возможны следующие режимы обмена данными через параллельный порт:

• ***Режим совместимости*** (*Compatibility Mode*) — однонаправленный (вывод) по протоколу ***Centronics***. Этот режим соответствует стандартному порту ***SPP*** (*Standard Parallel Port*).

• ***Полубайтный режим*** (*Nibble Mode*) — ввод байта в два цикла (по 4 бита), используя для приема линии состояния. Этот режим обмена может использоваться на любых адаптерах.

• ***Байтный режим*** (*Byte Mode*) — ввод байта целиком, используя для приема линии данных. Этот режим работает только на портах, допускающих чтение выходных данных (Bi-Directional или PS/2 Type 1).

• ***Режим EPP*** (*Enhanced Parallel Port, EPP Mode*) — двунаправленный обмен данными. Управляющие сигналы интерфейса генерируются аппаратно во время цикла обращения к порту.

• ***Режим ECP*** (*Extended Capability Port*, ***ECP Mode***) — двунаправленный обмен данными с возможностью аппаратного сжатия данных по методу RLE (Run Length Encoding) и использования FIFO-буферов и DMA. Управляющие сигналы интерфейса генерируются аппаратно.

В компьютерах с LPT-портом на системной плате режим — SPP, EPP, ECP или их комбинация — задается в BIOS Setup. Режим совместимости полностью соответствует стандартному порту SPP.

Все параллельные порты в режимах Compatibility Mode, Nibble Mode и Byte Mode используют только программное управление передачей данных. Драйвер должен устанавливать данные, проверять сигнал (Busy#), устанавливать соответствующие сигналы управления (-Strobe#) и затем переходить к следующему байту, что ограничивает эффективную скорость передачи данных на уровне от 50 до 100 КБайт/с.

В режимах EPP и ECP для передачи данных используются аппаратные средства. Например, в режиме EPP байт данных может быть передан периферии простой инструкцией Out. Контроллер ввода-вывода самостоятельно выполняет операции подтверждения связи и передачи данных на периферию.

Режимы нестандартных портов, реализующих протокол обмена Centronics аппаратно (Fast Centronics, Parallel Port FIFO Mode), могут и не являться режимами IEEE 1284, несмотря на наличие в них черт EPP и ECP.

Адаптер параллельного интерфейса представляет собой набор регистров, расположенных в пространстве ввода/вывода. Регистры порта адресуются относительно базового адреса порта, стандартными значениями которого являются 3BCh, 378h и 278h. Порт может использовать линию запроса аппаратного прерывания, обычно IRQ7 или IRQ5. Порт имеет внешнюю 8-битную шину данных, 5-битную шину сигналов состояния и 4-битную шину управляющих сигналов,

BIOS поддерживает до четырех (иногда до трех) LPT-портов (LPT1-LPT4) своим сервисом — прерыванием INT 17h, обеспе-

чивающим через них связь с принтером по интерфейсу Centronics. Этим сервисом BIOS осуществляет вывод символа (по опросу готовности, не используя аппаратных прерываний), инициализацию интерфейса и принтера, а также опрос состояния принтера.

Задание

1. Изучить принципы ввода-вывода информации посредством параллельного порта персонального компьютера.
2. Изучить стандарт IEEE-1284-1994, физический и электрический LPT-интерфейс, программные интерфейсы LPT-порта (Centronics, Nibble Mode, Byte Mode, EPP, ECP) и процедуру согласования режимов.
3. Изучить структуру и принцип работы адаптера параллельного интерфейса Centronics.
4. Разработать программы демонстрирующие взаимодействие с адаптером на уровнях:

– прерывание DOS (INT 21H Fn05H),

– прерывание BIOS (INT 17H),

– портов ввода-вывода (378H, 278H или 3BCH).

5. Сделать вывод по проделанной работе.

Примечания

1. Для отладки и проверки работы программ необходим принтер!
2. Дополнительно 100 % баллов начисляется за реализацию программы нуль-модемного соединения двух ПК через LPT-порт. Соединительный кабель выполняется студентом самостоятельно по одной из схем, приведенных ниже

XI, разъем PC#1		X2, разъем PC#2	
Бит	Контакт	Контакт	Бит
DR.O	2	15	SR.3
DR.1	3	13	SR.4
DR.2	4	12	SR.5
DR.3	5	10	SR.6
DR.4	6	11	SR.7\
SR.6	10	5	DR.3
SR.7\	11	6	DR.4

XI, разъем PC#1		X2, разъем PC#2	
Бит	Контакт	Контакт	Бит
SR.5	12	4	DR.2
SR.4	13	3	DR.1
SR.3	15	2	DR.0
GND	18—25	18—25	GND

Кабель для нуль-модемного соединения двух ПК через LPT-порт в режиме Nibble Mode (4-битный). Разъемы X1 и X2 — DB25-P (вилки).

Разъем X1, разъем PC#1		Разъем X2, разъем PC#2	
Контакт	Имя в ECP	Имя в ECP	Контакт
1	HostClk	PeriphClk	10
14	HostAck	PeriphAck	11
17	1284Active	+PeriphRequest#	15
16	ReverseRequest#	AckReverse#	12
10	PeriphCLk	HostClk	1
11	PeriphAck	HostAck	14
12	AckReverse#	ReverseRequest#	16
13	Xflag	–	–
15	PeriphRequest#	1284Active	17
2—9	Data[0:7]	Data[0:7]	2—9

Кабель для нуль-модемного соединения двух ПК через LPT-порт в режиме ECP и Byte Mode

3. Если принтер отсутствует, то допускается написание программы автотестирования LPT-порта при помощи заглушки. Заглушка в этом случае выполняется студентом самостоятельно. Автотестирование в данном случае состоит в передаче данных в LPT-порт и немедленном приеме этих же данных из него.

2	–	15
3	–	14
4	–	12
5	–	10
6	–	11

Заглушка для автотестирования LPT-порта

Пример

Приведенный на листинге Listing 24 фрагмент программного кода иллюстрирует вывод символа simbol на принтер (по умолчанию LPT1) при помощи прерывания DOS (INT 21H, Fn05H).

```
Listing 24.
{================================}
code_symb:=Ord(symbol);
asm
   MOV   AH, 05H
   MOV   DL, code_symb
   INT   21H
end;
{================================}
```

5 МЕТОДИКА ФОРМИРОВАНИЯ ТЕКУЩЕГО РЕЙТИНГА

Лабораторные занятия выполняются согласно расписанию занятий. Собеседование, прием экзаменов и зачетов проводится во время экзаменационной сессии.

Максимальный рейтинг по дисциплине составляет 120 баллов и определяется по таблице 1. Для получения оценки «отлично» требуется набрать не менее 100 баллов, «хорошо» — 80 баллов.

Таблица 1 — **Распределение рейтинга по элементам контроля**

Отчетные этапы	Элементы контроля	Рейтинг
1 контрольная неделя	Контрольная работа № 1	9
	Лабораторная работа № 1	7
	Лабораторная работа № 2	7
	Лабораторная работа № 3	7
2 контрольная неделя	Контрольная работа № 2	9
	Лабораторная работа № 4	7
	Лабораторная работа № 5	7
	Лабораторная работа № 6	7
Зачетная неделя	Контрольная работа № 3	9
	Лабораторная работа № 7	7
	Лабораторная работа № 8	7
	Лабораторная работа № 9	7
Посещение лекций		10
Творческое задание		20
Всего		120 мах

6 СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. ОС ТУСУР 6.1-97. Работы студенческие учебные и выпускные квалификационные. Общие требования к правилам оформления.

2. Бондарь А.В. Эксплуатация и развитие компьютерных сетей и систем. Часть 1: Программно-аппаратная реализация персонального компьютера: Учебное пособие. — ТУСУР, 1999.

3. Брябин В.М. Программное обеспечение персональных ЭВМ. — М.: Наука. Гл. ред. физ.-мат. лит., 1988. — 272 с.

4. Фролов А.В., Фролов Г.В. Аппаратное обеспечение IBM PC. В 2-х ч. — М.: ДИАЛОГ-МИФИ, 1992. — 208 с.

5. Воробьев Н.И. Проектирование электронных устройств: Учебное пособие для вузов по спец. "Автоматика и управление в технических системах". — М.: Высш. шк., 1989. — 223 с.

6. Левкин Г.Н., Левкина В.Е. Введение в схемотехнику IBM PC/AT. — М.: Изд-во МПИ, 1991. — 96 с.

7. Сопряжение датчиков и устройств ввода данных с компьютерами IBM PC: Пер. с англ. / Под ред. У. Томпкинса, Дж. Уэстера. — М.: Мир, 1992. — 592 с.

8. Шарапов А.В. Микропроцессорные устройства и системы: Методические указания к выполнению курсового проекта. — Томск: ТУСУР, 1998. — 38 с.

9. Мячев А.А. Системы ввода-вывода ЭВМ. — М.: Энергоатомиздат, 1983. — 168 с.

10. Гибсон Г.Ю. Аппаратные и программные средства микро-ЭВМ. — М.: Финансы и статистика, 1983. — 304 с.

11. Хвощ С.Т. и др. Организация последовательных мультиплексных каналов систем автоматизированного управления. — Л.: Машиностроение, 1988. — 120 с.

12. Якимов О.П. Газоразрядные матричные индикатроные панели. — М.: Сов. радио, 1980. — 72 с.

13. Однокристальные микроЭВМ: Справочник / Под ред. А.В. Боборыкина, А.А. Сергеева и др. — М.: МИКАП, 1994. — 400 с.