ГЛАВА 9

Часы реального времени

Каждый компьютер, как минимум, ассоциируется с точностью. А точность, так или иначе, подразумевает постоянный отсчет времени. Для реализации этой задачи используется специальный модуль, называемый часами реального времени (RTC — Real Time Clock). Именно эти часы позволяют нам отслеживать дату и время, а компьютеру позволяют упорядочить свою работу, и, кроме того, два раза в год предупреждать нас о переводе стрелок часов вперед или назад. Но, как вы заметили, компьютер при необходимости выключается, а после включения все временные (и не только) параметры восстанавливаются в соответствии с текущим временем. Для этого в модуле RTC имеется небольшая микросхема памяти размером от 64 до 128 байт, выполненная по специальной технологии CMOS. Эта микросхема потребляет очень мало энергии (в качестве источника используется литиевая батарейка) и может годами сохранять постоянные значения. Поэтому сюда записывается различная системная информация, в том числе и время. После включения компьютера операционная система считывает данные из CMOS и в соответствии с ними настраивает текущие значения даты и времени.

Не буду вдаваться в детали, скажу только, что аппаратная организация RTC базируется на задающем генераторе синусоидального сигнала с частотой 32 кГц (точнее 32,768 кГц). Для подпитки применяется высококачественная батарейка (Intel рекомендует использовать элементы питания фирмы Duracell) с высокой емкостью заряда (170 мА). Для расчета времени разряда можно емкость батарейки разделить на среднее значение тока разряда (5 мкА), и мы получим значение времени, измеряемое в часах (34,000 часов или 3,88 года). Поскольку точность часов реального времени напрямую зависит от напряжения питания, следует периодически (раз в год) проверять параметры батарейки и при необходимости делать замену. Конечно, для домашних компьютеров небольшая потеря точности часов не принесет никаких проблем, но для производственных систем с компьютерным управлением, а тем более работающих в режиме реального времени, любая неточ-

ность системных часов может привести к сбою всего технологического процесса.

Для программирования часов реального времени можно воспользоваться двумя способами:

1. С помощью функций BIOS.
2. С помощью портов ввода-вывода.

Разберем каждый вариант подробнее.

9.1. Использование функций BIOS

Для поддержки RTC применяется прерывание `int 1Ah` и несколько функций. Эти функции позволяют считывать и устанавливать значения даты и времени для памяти CMOS. Прежде чем перейти к их описанию, я хотел бы предоставить вашему вниманию структуру данных в CMOS. Несмотря на полный размер памяти (от 64 до 128 байт), стандартизированы только первые 51 байт. Остальные зависят от производителя материнской платы и здесь рассматриваться не будут. Каждое смещение байта в памяти CMOS определяет номер регистра, через который можно считывать и записывать информацию. Для RTC выделены первые 13 регистров, а остальные служат для других целей. Формат микросхемы памяти представлен в табл. 9.1.

***Таблица 9.1.** Формат памяти CMOS*

Адрес регистра	Описание
00h	Текущее значение секунды (00h—59h в формате BCD или 00h—3Bh)
01h	Значение секунд будильника (00h—59h в формате BCD или 00h—3Bh)
02h	Текущее значение минуты (00h—59h в формате BCD или 00h—3Bh)
03h	Значение минут будильника (00h—59h в формате BCD или 00h—3Bh)
04h	Текущее значение часа: 24-часовой режим (00h—23h в формате BCD или 00h—17h), 12-часовой режим AM (01h—12h в формате BCD или 00h—0Ch), 12-часовой режим PM (81h—92h в формате BCD или 81h—8Ch)
05h	Значение часа будильника: 24-часовой режим (00h—23h в формате BCD или 00h—17h), 12-часовой режим AM (01h—12h в формате BCD или 00h—0Ch), 12-часовой режим PM (81h—92h в формате BCD или 81h—8Ch)
06h	Текущее значение дня недели (01h—07h в формате BCD или 01h—07h, где 01h соответствует воскресенью)
07h	Текущее значение дня месяца (01h—31h в формате BCD или 01h—1Fh)
08h	Текущее значение месяца (01h—12h в формате BCD или 01h—0Ch)

Таблица 9.1 (продолжение)

Адрес регистра	Описание
09h	Текущее значение года (00h—99h в формате BCD или 00h—63h)
0Ah	Регистр состояния A: бит 7 — обновление времени (1 — идет обновление времени, 0 — доступ разрешен), биты 4—6 — делитель частоты (010b — 32,768 кГц), биты 0—3 — значение пересчета частоты (0110b — 1024 Гц)
0Bh	Регистр состояния B: бит 7 — запрещение обновления часов (1 — идет установка, 0 — обновление разрешено), бит 6 — разрешение периодического прерывания IRQ8 (1 — разрешено, 0 — запрещено), бит 5 — разрешение прерывания от будильника (1 — разрешено, 0 — запрещено), бит 4 — вызов прерывания после цикла обновления (1 — разрешено, 0 — запрещено), бит 3 — разрешение генерации прямоугольных импульсов (1 — разрешено, 0 — запрещено), бит 2 — выбор формата представления даты и времени (1 — двоичный, 0 — BCD), бит 1 — выбор часового режима (1 — 24 часа, 0 — 12 часов), бит 0 — автоматический переход на летнее время (1 — разрешено, 0 — запрещено)
0Ch	Регистр состояния C (только для чтения): бит 7 — признак выполненного прерывания (1 — произошло, 0 — не было), бит 6 — периодическое прерывание (1 — есть, 0 — нет), бит 5 — прерывание от будильника (1 — есть, 0 — нет), бит 4 — прерывание после обновления часов (1 — есть, 0 — нет), биты 0—3 зарезервированы и должны быть равны 0
0Dh	Регистр состояния D: бит 7 — состояние батареи и памяти (1 — память в норме, 0 — батарейка разряжена)
0Eh	Состояние POST после загрузки компьютера: бит 7 — сброс часов по питанию (1 — нет питания), бит 6 — ошибка контрольной суммы (CRT) в CMOS памяти (1 — ошибка), бит 5 — несоответствие конфигурации оборудования (1 — POST обнаружила ошибки конфигурации), бит 4 — ошибка размера памяти (1 — POST обнаружила несоответствие размера памяти с записанным в CMOS), бит 3 — сбой контроллера первого жесткого диска (1 — есть сбой), бит 2 — сбой в работе часов RTC (1 — есть сбой), биты 0 и 1 зарезервированы и должны быть равны 0
0Fh	Состояние компьютера перед последней загрузкой: 00h — был выполнен сброс по питанию (кнопка Reset или <Ctrl>+<Alt>+<Del>), 03h — ошибка тестирования памяти, 04h — POST была завершена и система перезагружена, 05h — переход (jmp dword ptr) на адрес в 0040h:0067h, 07h — ошибка защиты при тестировании, 08h — ошибка размера памяти
10h	Тип дисковода (0—3 для A и 4—7 для B): 0000b — нет, 0001b — 360 Кб, 0010b — 1,2 Мб, 0011b — 720 Кб, 0100b — 1,44 Мб, 0101b — 2,88 Мб
11h	Резерв
12h	Тип жесткого диска (0—3 для D и 4—7 для C): 0000b — нет, 0001b — 1110b — тип дисковода (от 1 до 14), 1111b — диск первый описывается в регистре 19h, а диск второй — в 1Ah

Таблица 9.1 *(окончание)*

Адрес регистра	Описание
13h	Резерв
14h	Состояние оборудования: биты 6—7 — число флоппи-дисководов (00b — один, 01b — два), биты 4—5 — тип дисплея (00b — EGA или VGA, 01b — цветной 40 × 25, 10b — цветной 80 × 25, 11b — монохромный 80 × 25), биты 2 и 3 зарезервированы, бит 1 — наличие сопроцессора (1 — есть), бит 0 — наличие флоппи-дисковода (1 — есть)
15h	Младший байт размера основной памяти в килобайтах (80h)
16h	Старший байт размера основной памяти в килобайтах (02h)
17h	Младший байт размера дополнительной памяти в килобайтах
18h	Старший байт размера дополнительной памяти в килобайтах
19h	Тип первого жесткого диска
1Ah	Тип второго жесткого диска
1Bh—2Dh	Резерв
2Eh	Старший байт контрольной суммы регистров CMOS (10h—2Dh)
2Fh	Младший байт контрольной суммы регистров CMOS (10h—2Dh)
30h	Младший байт размера дополнительной памяти в килобайтах
31h	Старший байт размера дополнительной памяти в килобайтах
32h	Значение века в формате BCD
33h	Дополнительный флаг свойств: бит 7 — размер памяти (1 — больше 1 Мб, 0 — до 1 Мб), бит 6 — используется программой установки, биты 0—5 зарезервированы
34h—3Fh (7Fh)	Зависят от производителя

Используемый формат представления данных BCD (Binary Coded Decimal) представляет собой двоично-десятичный код, где каждый байт содержит два независимых значения. Первое из них кодируется битами 7—4, а второе — битами 3—0. Поддерживаются только положительные значения до 99 включительно.

А теперь рассмотрим функции BIOS для управления часами реального времени.

9.1.1. Функция *00h*

Данная функция позволяет получить текущее значение счетчика RTC.

Использование:

1. В регистр AH следует поместить код функции 00h.
2. Вызвать прерывание int 1Ah.

Выход:

После выполнения функции в регистр AL будет записан код перехода: 00h — произошел переход времени через полночь. В регистры CX и DX будут записаны соответственно старшее и младшее значения счетчика. В случае ошибки флаг переноса CF будет установлен в 1, иначе сброшен. Поскольку частота сигнала прерывания составляет 18,2 Гц (18,2 прерываний в секунду), за сутки набегает немногим более 1 572 480 отсчетов, а после этого счетчик сбрасывается в 0. Например, чтобы получить текущее значение счетчика, можно использовать код из листинга 9.1.

Листинг 9.1. Получение текущего значения счетчика

```
; определяем переменную для хранения полученного значения
Time_Count dd ?
; общий код программы
mov AH, 00h    ; код функции 00h
int 1Ah        ; вызываем прерывание
jc  ERROR_HND ; если произошла ошибка, передаем управление обработчику
; сохраняем текущее значение счетчика
mov word ptr Time_Count, DX; младшее слово
mov word ptr Time_Count + 2, CX; младшее слово
```

9.1.2. Функция *01h*

Функция позволяет установить новое значение счетчика для системного таймера.

Использование:

1. В регистр AH следует поместить код функции 01h.
2. В регистр CX следует записать старшее слово значения счетчика.
3. В регистр DX следует записать младшее слово значения счетчика.
4. Вызвать прерывание int 1Ah.

Выход:

В случае ошибки флаг переноса CF будет установлен в 1, иначе сброшен.

9.1.3. Функция *02h*

Эта функция позволяет получить текущее значение времени в формате BCD.

Использование:

1. В регистр AH следует поместить код функции 02h.
2. Вызвать прерывание int 1Ah.

Выход:

В случае ошибки флаг переноса CF будет установлен в 1, иначе сброшен. При успешном выполнении функции в регистры будут записаны следующие значения: CH — значение часа, CL — значение минуты, DH — значение секунды, DL — значение перехода на летнее время (1 — есть переход, 0 — нет). Посмотрите в листинге 9.2, как можно получить текущее время в формате BCD.

Листинг 9.2. Получение текущего времени в формате BCD

```
; определяем переменные для хранения полученных значений
Current_Hour db ?
Current_Minute db ?
Current_Second db ?
; общий код программы
mov AH, 02h    ; код функции 02h
int 1Ah        ; вызываем прерывание
jc  ERROR_HND ; если произошла ошибка, передаем управление обработчику
; сохраняем текущее значение времени
mov byte ptr Current_Hour, CH; часы
mov byte ptr Current_Minute, CL; минуты
mov byte ptr Current_Second, DH; секунды
```

9.1.4. Функция *03h*

Функция позволяет установить текущее значение времени в формате BCD.

Использование:

1. В регистр AH следует поместить код функции 03h.
2. В регистр CH следует записать часы.
3. В регистр CL следует записать минуты.
4. В регистр DH следует записать секунды.
5. В регистр DL нужно поместить значение перехода на летнее время.
6. Вызвать прерывание int 1Ah.

Выход:

В случае ошибки флаг переноса CF будет установлен в 1, иначе сброшен. Пример для установки нового системного времени приведен в листинге 9.3.

Листинг 9.3. Установка нового системного времени

```
mov CH, 11h    ; 11 часов
mov CL, 15h    ; 15 минут
mov DH, 30h    ; 30 секунд
mov DL, 1      ; с переходом на летнее время
mov AH, 03h    ; код функции 03h
int 1Ah        ; вызываем прерывание
jc  ERROR_HND ; если произошла ошибка, передаем управление обработчику
```

9.1.5. Функция *04h*

Эта функция позволяет получить текущую дату в формате BCD.

Использование:

1. В регистр AH следует поместить код функции 04h.
2. Вызвать прерывание int 1Ah.

Выход:

В случае ошибки флаг переноса CF будет установлен в 1, иначе сброшен. При успешном выполнении функции в регистры будут записаны следующие значения: CH — значение века, CL — значение года, DH — значение месяца, DL — значение дня. Например, для получения текущей даты можно использовать код, представленный в листинге 9.4.

Листинг 9.4. Получение текущей даты в формате BCD

```
; определяем переменные для хранения полученных значений
Current_Vek db ?
Current_Year db ?
Current_Month db ?
Current_Day db ?
; общий код программы
mov AH, 04h    ; код функции 04h
int 1Ah        ; вызываем прерывание
jc  ERROR_HND ; если произошла ошибка, передаем управление обработчику
; сохраняем текущее значение даты
mov byte ptr Current_Vek, CH; век
```

```
mov byte ptr Current_Year, CL; год
mov byte ptr Current_Month, DH; месяц
mov byte ptr Current_Day, DH; день
```

9.1.6. Функция *05h*

Данная функция позволяет установить текущую дату в формате BCD.

Использование:

1. В регистр AH следует поместить код функции 05h.
2. В регистр CH следует записать век.
3. В регистр CL следует записать год.
4. В регистр DH следует записать месяц.
5. В регистр DL нужно поместить значение дня.
6. Вызвать прерывание int 1Ah.

Выход:

В случае ошибки флаг переноса CF будет установлен в 1, иначе сброшен.

9.1.7. Функция *06h*

Функция позволяет установить время срабатывания для будильника.

Использование:

1. В регистр AH следует поместить код функции 06h.
2. В регистр CH следует записать часы.
3. В регистр CL следует записать минуты.
4. В регистр DH следует записать секунды.
5. Вызвать прерывание int 1Ah.

Выход:

В случае ошибки флаг переноса CF будет установлен в 1, иначе сброшен.

9.1.8. Функция *07h*

Эта функция позволяет выключить будильник.

Использование:

1. В регистр AH следует поместить код функции 07h.
2. Вызвать прерывание int 1Ah.

Выход:

В случае ошибки флаг переноса CF будет установлен в 1, иначе сброшен. При успешном выполнении в регистр AL будет записано значение регистра CMOS по адресу 0Bh.

9.2. Использование портов

Для доступа к часам реального времени используются всего два порта: 70h и 71h. Порт 70h используется только для записи и позволяет выбрать адрес регистра в CMOS памяти. Порт 71h применяется как для записи, так и для чтения данных из указанного (через порт 70h) регистра CMOS. Оба порта являются 8-разрядными. Кроме того, бит 7 в порту 70h не относится к работе с RTC, а управляет режимом немаскируемых прерываний (1 — прерывания запрещены). Если у вас будут проблемы с доступом к регистрам CMOS, следует запрещать прерывания (команда cli) перед началом работы и разрешать прерывания (команда sti) после. Но, как правило, этого не требуется. Пример получения числа установленных флоппи-дисководов показан в листинге 9.5.

Листинг 9.5. Определение числа установленных флоппи-дисководов

```
mov AL, 14h  ; регистр CMOS 14h
out 70h, AL  ; записываем значение в порт
jmp $ + 2    ; делаем небольшую паузу
in  AL, 71h  ; читаем значение регистра
test AL, 0   ; проверяем наличие дисковода в системе
jz  NO_FDD   ; дисководов нет
and AL, 0C0h ; выделяем значение в битах 6 и 7
shr AL, 6    ; получаем результат
; сохраняем количество дисководов в переменную
cmp AL, 0    ; один дисковод
jne @dalee
mov byte ptr NUM_FDD, 1
@dalee:
cmp AL, 01   ; два дисковода
jne ERROR_HND; не удалось определить число дисководов
mov byte ptr NUM_FDD, 2
```

Аналогичный пример для C++ показан в листинге 9.6.

Листинг 9.6. Определение числа установленных флоппи-дисководов в C++

```
// напишем функцию для определения числа флоппи-дисководов
int GetNumFDD ()
```

```
{
   DWORD dwResult = 0;
   // записываем значение регистра CMOS 14h в порт
   outPort ( 0x70, 0x14, 1);
   // делаем небольшую паузу
   Sleep ( 1);
   // читаем значение из порта 71h
   inPort ( 0x71, &dwResult, 1);
   // проверяем наличие дисковода в системе
   if ( ( dwResult & 0x01) == 0x00) return 0;
   // выделяем значение в битах 6 и 7
   dwResult &= 0x0C;
   dwResult = dwResult >> 6; // выделяем результат
   // проверяем
   switch ( dwResult)
   {
    case 0:
     return 1; // один флоппи-дисковод
    case 1:
     return 2; // два флоппи-дисковода
   }
   return 0;
}
```

Таким же способом можно получить или записать любое значение в регистры CMOS. Давайте попробуем выполнить очистку памяти CMOS. Хотя эта операция и позволяет осуществить программный сброс памяти, использовать ее стоит очень осторожно. Желательно, чтобы на материнской плате были установлены две микросхемы CMOS. Причина в том, что на старых и некачественных платах затирание ячеек памяти может привести к полному отказу начальной загрузки компьютера. Это может быть связано и с изношенностью микросхемы, и с плохим качеством исполнения, а может зависеть от батарейки. На современных качественных платах таких проблем, как правило, не возникает, но возможность нарушения работы системы все равно существует. Исходя из этого, я приведу пример очистки памяти CMOS, но ответственность за любые проблемы будет лежать целиком на вас. Итак, рассмотрим следующий пример, показанный в листинге 9.7.

Листинг 9.7. Очистка памяти CMOS

```
Clear_CMOS proc near
xor CX, CX    ; обнуляем регистр CX
mov CL, 3Fh   ; предполагаем, что размер CMOS равен 64 байта
```

```
@repeat:
mov AL, CL    ; передаем номер регистра CMOS
out 70h, AL   ; записываем значение в порт
nop           ; делаем небольшую паузу
out 71h, AL   ; записываем значение в порт
loop @repeat ; повторяем для следующего регистра
Clear_CMOS endp
```

Для очистки памяти CMOS в C++ используйте код из листинга 9.8.

Листинг 9.8. Очистка памяти CMOS в C++

```
// напишем функцию для очистки CMOS
void Clear_CMOS ( unsigned int uSizeCMOS)
{
   if ( uSizeCMOS < 64) return;
   for ( int i = 0; i < uSizeCMOS; i++)
   {
     outPort ( 0x70, i, 1); // записываем номер регистра в CMOS
     outPort ( 0x71, i, 1); // записываем значение в регистр
   }
}
// воспользуемся нашей функцией для очистки памяти CMOS размером 64 байта
Clear_CMOS ( 64);
```

И напоследок я хочу привести примеры функций для C++, позволяющие писать (листинг 9.9) и читать (листинг 9.10) память CMOS в файл.

Листинг 9.9. Сохранение памяти CMOS в файл

```
// пишем функцию сохранения CMOS в файл
 bool Save_CMOS ( char* FileName)
{
   FILE* outFile;
   DWORD dwResult = 0;
   BYTE buffer[64];
   // открываем новый файл
   if ( ( outFile = fopen ( FileName, "wb")) == NULL)
      return false; // не удалось создать файл
   for ( int i = 0; i < 64; i++)
   {
     outPort ( 0x70, i, 1); // записываем номер регистра в CMOS
     Sleep ( 1); // пауза
```

```
      inPort ( 0x71, &dwResult, 1); // читаем байт из CMOS
      buffer[i] = dwResult;
   }
   // записываем данные в файл
   fwrite ( buffer, 1, 64, outFile);
   // закрываем файл
   fclose( outFile);
   return true;
}
```

Листинг 9.10. Загрузка памяти CMOS из файла

```
// пишем функцию загрузки CMOS из файла
bool Load_CMOS ( char* FileName)
{
   FILE* inFile;
   DWORD dwResult = 0;
   BYTE buffer[64];
   // открываем новый файл
   if ( ( inFile = fopen ( FileName, "wb")) == NULL)
      return false; // не удалось создать файл
   // читаем файл в буфер
   fread( buffer, 1, 64, inFile);
   // закрываем файл
   fclose( inFile);
   // сохраняем данные в CMOS
   for ( int i = 0; i < 64; i++)
   {
     dwResult = buffer[i];
     outPort ( 0x70, i, 1); // записываем номер регистра в CMOS
     outPort ( 0x71, dwResult, 1); // записываем значение в регистр
   }
   return true;
}
```

На этом можно завершить программирование часов реального времени и памяти CMOS.

ГЛАВА 10

Таймер

В любом компьютере наряду с часами реального времени существует устройство системного таймера, для реализации которого применяется специальная микросхема (например, 8254). Он помогает организовать всевозможные временные задержки, счетчики и управляющие сигналы. Из всех предоставляемых таймером функций, можно выделить несколько основных:

1. Организация часов реального времени.
2. Программируемый генератор прямоугольных и синусоидальных импульсов.
3. Счетчик событий таймера.
4. Управление двигателями флоппи-дисководов.

Таймер предоставляет три независимых канала, каждый из которых имеет свое назначение. В первом канале (0) отслеживается текущее значение времени от момента включения компьютера, для хранения которого используется область памяти BIOS (`0040:006C`). Каждое изменение значения (18,2 раз в секунду) в этом канале генерирует прерывание `IRQ0` (`int 8h`). Данное прерывание будет обработано процессором в первую очередь, но при этом должны быть разрешены аппаратные прерывания. При программировании первого канала всегда следует после выполнения задачи восстанавливать его первоначальное состояние. Второй канал (1) используется системой для работы с контроллером прямого доступа к памяти (DMA). Третий канал (2) позволяет управлять системным динамиком.

Генератор сигналов таймера вырабатывает импульсы с частотой 1 193 180 Гц. Поскольку максимальное значение 16-битного регистра ограничено значением 65 535, используется делитель частоты. В итоге, результирующее значение равно 18,2 Гц. Именно с такой частотой выдается прерывание `IRQ0`.

Для работы с системным таймером используются порты от `40h` до `43h`. Все они имеют размер 8 бит. Порты с номерами `40h`, `41h` и `42h` связаны соответ-

ственно с первым, вторым и третьим каналами, а порт 43h работает с управляющим регистром таймера. Принцип работы с портами очень простой: в порт 43h записывается управляющая команда, а после этого данные записываются или считываются из 40h, 41h и 42h, в зависимости от решаемой задачи. Формат командного регистра представлен в табл. 10.1.

Таблица 10.1. *Формат управляющего регистра таймера*

Биты	7	6	5	4	3	2	1	0
Описание	Номер канала		Тип операции		Режим		Формат	

Приведем краткое описание табл. 10.1.

- Бит 0 определяет формат представления данных: 0 — двоичный (16-битное значение от 0000h до FFFFh), 1 — двоично-десятичный (BCD от 0000 до 9999).
- Биты 1—3 определяют режим работы таймера. Существуют шесть режимов работы (от 0 до 5). Возможные значения перечислены в табл. 10.2.
- Биты 4—5 определяют тип операции. Имеются четыре возможных значения, которые перечислены в табл. 10.3.
- Биты 6—7 позволяют выбрать номер канала или управляющий регистр (только для операций чтения). Возможные значения этого поля перечислены в табл. 10.4.

Таблица 10.2. *Режимы работы таймера*

Бит 3	Бит 2	Бит 1	Описание режима
0	0	0	Генерация прерывания IRQ0 при установке счетчика в 0
0	0	1	Установка в режим ждущего мультивибратора
0	1	0	Установка в режим генератора импульсов
0	1	1	Установка в режим генератора прямоугольных импульсов
1	0	0	Установка в режим программно-зависимого одновибратора
1	0	1	Установка в режим аппаратно-зависимого одновибратора

Таблица 10.3. *Тип операции*

Бит 5	Бит 4	Тип операции
0	0	Команда блокировки счетчика
0	1	Чтение/запись только младшего байта

Таблица 10.3 *(окончание)*

Бит 5	Бит 4	Тип операции
1	0	Чтение/запись только старшего байта
1	1	Чтение/запись младшего, а за ним старшего байта

Таблица 10.4. *Возможные значения для поля 6–7*

Бит 7	Бит 6	Описание
0	0	Выбор первого канала (0)
0	1	Выбор второго канала (1)
1	0	Выбор третьего канала (2)
1	1	Команда считывания значений из регистров каналов

При установке битов 6 и 7 управляющего регистра в 1 будет использована команда считывания данных. При этом формат определения этого регистра меняется согласно табл. 10.5.

Таблица 10.5. *Формат управляющего регистра в режиме считывания*

Биты	7	6	5	4	3	2	1	0
Описание	1	1	Б	Статус	Канал 2	Канал 1	Канал 0	0

Приведем краткое описание таблицы.

- Бит 0 не используется и должен быть установлен в 0.
- Биты 1—3 позволяют установить номера каналов. Установка бита в 1 выбирает соответствующий номер канала.
- Бит 4 позволяет получить состояние для выбранного канала (каналов) при установке в 0.
- Бит 5 позволяет зафиксировать значение счетчика для выбранного канала (каналов) при установке в 0.
- Биты 6 и 7 определяют команду чтения и должны быть установлены в 1.

При выполнении команды блокировки счетчика (биты 4 и 5 установлены в 0) можно получить значение (состояние) выбранного счетчика без остановки самого таймера. Результат считывается из указанного канала (биты 6 и 7). При этом формат команды будет таким, как показано в табл. 10.6.

Таблица 10.6. *Формат команды блокировки*

Биты	7	6	5	4	3	2	1	0
Описание	Номер канала		0		0		Не используются	

Приведем краткий комментарий к таблице.

- Биты 0—3 не используются и должны игнорироваться.
- Биты 4 и 5 определяют команду блокировки и должны быть установлены в 0.
- Биты 6 и 7 определяют номер канала (см. табл. 10.4), для которого будет выполнена команда блокировки.

Полученный байт состояния имеет определенный формат (табл. 10.7).

Таблица 10.7. *Формат байта состояния канала*

Биты	7	6	5	4	3	2	1	0
Описание	OUT	Готов	Тип операции		Режим работы			Формат

Приведем краткое пояснение к таблице 10.7.

- Бит 0 определяет формат представления данных: 0 — двоичный (16-битное значение от `0000h` до `FFFFh`), 1 — двоично-десятичный (BCD от `0000` до `9999`).
- Биты 1—3 определяют режим работы таймера. Возможные значения перечислены в табл. 10.2.
- Биты 4—5 определяют тип операции. Возможные значения перечислены в табл. 10.3.
- Бит 6 определяет готовность счетчика для считывания данных (1 — готов, 0 — счетчик обнулен).
- Бит 7 определяет состояние выходного сигнала на канале в момент блокировки счетчика импульсов.

Рассмотрим стандартный пример для установки начального значения счетчика для второго канала (управляет динамиком) равным 5 120 Гц (листинг 10.1).

Листинг 10.1. Установка начального значения счетчика для второго канала

```
mov AL, 10110110b   ; канал 2, операция 4, режим 3, формат 0
out 43h, AL . ; записываем значение в порт
mov AX, 0E9  ; определяем делитель частоты для получения 5120 Гц
```

```
out 42h, AL  ; посылаем младший байт делителя
mov AL, AH   ; копируем значение
out 42h, AL  ; посылаем старший байт делителя
```

Аналогичный пример для C++ показан в листинге 10.2.

Листинг 10.2. Установка начального значения счетчика для второго канала в C++

```
// пишем функцию установки значения счетчика
void SetCount ( int iDivider)
{
   int iValue = 0;
   outPort ( 0x43, 0xB6, 1); // канал 2, операция 4, режим 3, формат 0
   iValue = iDivider & 0x0F;
   outPort ( 0x42, iValue, 1); // младший байт делителя
   outPort ( 0x42, ( iDivider >> 4), 1); // старший байт делителя
}
```

А теперь рассмотрим пример для получения случайного значения в установленном диапазоне (листинг 10.3).

Листинг 10.3. Получение случайного значения в диапазоне *от* 0 *до* 99

```
; выделяем место для переменной
Random_Value db ?
; сначала пишем процедуру для установки режима работы таймера
TimerInit proc near
mov AL, 10110111b   ; канал 2, операция 4, режим 3, формат 1
out 43h, AL  ; записываем значение в порт
mov AX, 11931; определяем делитель частоты
out 42h, AL  ; посылаем младший байт делителя
mov AL, AH   ; копируем значение
out 42h, AL  ; посылаем старший байт делителя
ret
TimerInit endp
; пишем процедуру получения случайного значения
GetRandomValue proc near
mov AL, 10000000b   ; канал 2, получить значение, биты 3-0 игнорируем
out 43h, AL  ; записываем значение в порт
in  AL, 42h  ; читаем младший байт значения счетчика
mov AH, AL   ; копируем его в AH
in  AL, 42h  ; читаем старший байт значения счетчика
```

```
call TimerInit      ; устанавливаем таймер заново
xchg AH, AL  ; меняем местами младший и старший байты
; сохраняем полученное значение в переменную
mov byte ptr Random_Value, AX
ret
GetRandomValue endp
```

Реализация генератора случайных чисел на C++ представлена в листинге 10.4.

Листинг 10.4. Получение случайного значения в C++

```
// пишем функцию установки таймера
void InitCount ( int iMaximum)
{
   int iValue = 0;
   int iDiv = 1193180 / iMaximum;
   outPort ( 0x43, 0xB7, 1); // канал 2, операция 4, режим 3, формат 1
   iValue = iDiv & 0x0F;
   outPort ( 0x42, iValue, 1); // младший байт делителя
   outPort ( 0x42, ( iDiv >> 4), 1); // старший байт делителя
}
// пишем функцию генерации случайного значения
int GetRandomValue ( int iMaximum)
{
   DWORD dwLSB = 0, dwMSB = 0;
   // канал 2, получить значение, биты 3-0 игнорируем
   outPort ( 0x43, 0x80, 1);
   // читаем младший байт значения счетчика
   inPort ( 0x42, &dwLSB, 1);
   // устанавливаем таймер заново
   InitCount ( iMaximum);
   // читаем старший байт значения счетчика
   inPort ( 0x42, &dwMSB, 1);
   return ( int) ( ( WORD) ( ( ( BYTE) ( dwLSB)) |
          ( ( ( WORD) ( ( BYTE) ( dwMSB))) << 8)));
}
```

На этом можно считать тему программирования системного таймера завершенной.