МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»



**Контрольна робота №1**

з курсу «Системне програмування»

**Варіант 14**

Виконав студент

групи КНз-2

Чалий Михайло

Перевірив

Мазур В.В.

Львів 2014

# Робота з годинником реального часу (RTC)

## Короткі теоретичнi вiдомостi

Персональні комп’ютери зазвичай мають дві підсистеми таймерів, які паралельно відраховують поточний час.

**Таймер BIOS або системний таймер**, що реалізовано мікросхемою і8253 (вітчизняний аналог – КР580ВН53, в AT – і8254), працює, як і решта вузлів комп’ютера, тільки коли ПК увімкнений. Його канал 0 генерує сигнали з частотою приблизно 18,206 Гц, яка викликає апаратні переривання рівня 0 (вектор 08h). Обробник даного переривання, який входить у систему BIOS, з кожним перериванням інкрементує вміст 4-байтної чарунки за адресою 40h:6Ch, яка розміщена у області даних BIOS.

**Годинник реального часу (RTC)** який розміщено у мікросхемі з низьким споживання енергії (КМОП-мікросхема), що після вимкнення живлення комп’ютера продовжує працювати, отримуючи енергію від вбудованого у комп’ютер аккумулятора (батареї, конденсатора). Складаєьтся з кварцевого генератора, системи керування і має внутрішню пам’ять, у якій зберігається й постійно нарощується значення поточного часу. Значення часу оновлюється щосекунди, причому ця операція виконується на апаратному рівні, не витрачаючи ресурси процесору чи оперативної пам’яті.

Таймер RTC включає в себе контролер та невеликий блок пам’яті об’ємом 64 байт. Перші 14 байт використовються для відліку часу; решта 50 байт зберігають інформацію про конфігурацію системи. Підсистема забезпечує наступні функції:

* відлік поточного часу з точністю до 1 с;
* роботу будильника, який у встановлений час генерує сигнал переривання лінією IRQ8, закріпленним за вектором 70h (може використовуватись для пробудження комп’ютера);
* режим періодичних переривань (лінією IRQ8), частоту яких можна програмно налаштовувати у межах від 2 Гц до 8 КГц;
* зберігання даних про конфігурацію системи (об’єм базової та розширеної пам’яті, типи магнітних дисків і т.д.);
* зберігання даних прикладних програм.

У таблиці 1 наведено призначення окремих байтів КМОП-пам’яті (0-3Fh).

Таблиця 1 – Адресний простір пам’яті КМОП-мікросхеми

|  |  |  |
| --- | --- | --- |
| Адреса  поля | Кількість байт | Призначення |
| 00h | 1 | Секунди в BCD |
| 01h | 1 | Секунди будильника в BCD |
| 02h | 1 | Хвилини в BCD |
| 03h | 1 | Хвилини будильника в BCD |
| 04h | 1 | Години в BCD |
| 05h | 1 | Години будильника в BCD |
| 06h | 1 | День тижня (може бути відсутній) |
| 07h | 1 | Число в BCD |
| 08h | 1 | Місяць (січень – 1, лютий – 2 і т.д.) в BCD |
| 09h | 1 | Рік, молодші дві цифри в BCD |
| 0Ah | 1 | **Регістр A** |
| 0Bh | 1 | **Регістр В** |
| 0Ch | 1 | **Регістр С** |
| 0Dh | 1 | **Регістр D** |
| 0Eh | 1 | Байт діагностування |
| 0Fh | 1 | Байт кода скиду процесора |

Байти з номерами 0Ah, 0Bh, 0Ch та 0Dh виконують функції керуючих регістрів.

**0Ah** – регістр стану RTC

**0Bh** – регістр стану RTC

**0Ch** – регістр стану RTC: біти стану переривання (тільки чатання).

**0Dh** – регістр стану RTC. Біт D7=1, якщо CMOS отримує живлення від автономного джерела; 0 – відсунє живлення від автононого джерела.

**0Eh** – байт байт результатів початкового тестування.

**0Fh** – байт стану перезавантаження. Цей байт зчитується після скиду ЦП, щоб визначити, чи не було скиду, викликаного виведенням 80286 з захищеного режима. Він може мати наступні значення:

0 – гарячий рестарт (Ctrl-Alt-Del) або неочікувана зупинка;

1 – зупинка після визначення розміру ОЗП;

2 – зупинка після тестування пам’яті;

3 – зупинка після виявлення помилки паритета пам’яті;

4 – рестарт за запитом початкового завантажувача;

5 – рестарт за скидом контролера переривань та JMP FAR PTR [0:467h];

6, 7, 8 – зупинка після теста захищеного режиму;

9 – рестарт за JMP FAR PTR [0:467h].

Звертання до байтів КМОП-пам’яті здійснюється за допомогою портів 70h та 71h у два етапи: спочатку в порт 70h записується номер необхідного байта пам’яті; потім через порт 71h виконується читання (in) чи запис (out) байта пам’яті.

Описаним вище способом можна звертатись до всіх чарунок CMOS-пам’яті, за винятком перших десяти. Справа в тому, що КМОП-мікросхема один раз за секунду виконує корекцію поточного часу та перевірку стану будильника. На час корекції ділянка КМОП-пам’яті, яка відноситься до годин, календаря та будильника, відключається від системної магістралі та стає недоступним для програмного звертання. На це й же час встановлюється біт D7 регістра А. Тому перед зверненням до адрес 00h-09h CMOS-пам’яті необхідно спочатку дочекатись скиду біта D7 регістра А, що вказує на завершення процесу корекції і тільки потім здійснювати зверення до пам’яті. В лістингах 1 та 2 продемонстровано читання чарунок КМОП-пам’яті з очікуванням закінчення цикла корекції.

Лістинг 1 – Програма читання з RTC та виведення на екран поточного року

#include <stdio.h>

#include <dos.h>

int main (void)

{

unsigned char YearH, YearL;

do{

outportb(0x70,0x0A);

}while ((inportb(0x71)&0x80)==1); //біт D7 рег. А

// встановлено?

outportb(0x70,0x32); //байт 32h - старші 2 цифри року

YearH=inportb(0x71);

outportb(0x70,0x09); //байт 9h - молодші 2 цифри року

YearL=inportb(0x71);

printf("Поточний рiк: %02x%02x",YearH,YearL);

return 0;

}

Лістинг 2 – Програма читання з RTC та виведення на екран поточної дати

#include <stdio.h>

#include <dos.h>

int main (void)

{

unsigned char Day, Mon;

do{

outportb(0x70,0x0A);

}while ((inportb(0x71)&0x80)==1); //корекцію завершено (D7 Ah)?

outportb(0x70,0x07); //07 – число в ВСD (з табл. 1)

Day=inportb(0x71);

outportb(0x70,0x08); //08 – місяць в ВСD (з табл. 1)

Mon=inportb(0x71);

printf("Поточна дата: %02x.%02x",Day,Mon);

return 0;

}

ГРЧ працює від внутрішнього кварцового генератора, частоту якого підібрано таким чином, щоб сигнали на його виході (після перерахунку) мають частоту точно 1 Гц (щосекунди). Ці сигнали використовуються для відліку поточного часу в байтах годин та календаря КМОП-пам’яті. Окрім постійного перерахунку, який забезпечує частоту 1 Г, в КМОП-схему включено ще вузол настроюваного перерахунку, вихідні сигнали якого поступають на лінію IRQ8, ініціалізуючи перілдичні переривання через вектор 70h. Коефіцієнт перерахунку і, відповідно, частоту періодичних переривань можна програмного налаштовувати, змінюючи біти D0-D3 регістра А. Для дозволу/заборони періодичних переривань використовується біт D6 регістра В.

Встановлення будильника здійснюється записом необхідного часу розбудження в байти секунд, хвилин та годин КМОП-пам’яті (див. таблицю 1). Під час запису в ці чарунки необхідно передбачити очікування кінця циклів корекції, як це показано у лістингу 1. Схеми таймера періодично порівнюють поточний час з часом, який записаний у байтах будильника, і при досягненні рівності збуджує сигнал сигнального переривання на лінії IRQ8. Для дозволу/заборони сигнальних переривань необхідно ініціалізувати біт D5 регістра В.

Якщо відповідні значення встановлено у всіх трьох байтах будильника, сигнальні переривання буде формуватись у вказаний час щодоби. Проте програмно можна встановити в одном або декількох байтах будильника “довільний” код – будь-яке число від C0h до FFh. Якщо довільний код встановлено у байті годин будильника, то сигнальні переривання збуджуються щогодини. Після запису довільного кода у байт годин та хвилин – сигнал генерується щохвилини, а за наявності довільного коду у всіх трьох чарунках – щосекунди. Особливість цього режима заключається в тому, що переривання формуються не просто заданої частоти, а у заданий момент кожної хвилини або кожної години. Наприклад, якщо у байті сукунди записано число 30h, а в байтах хвилин та годин – C0h, то переривання будуть збуджуватись точно на 30-й секунді кожної хвилини.

При обробці переривань від CMOS-мікросхеми необхідно враховувати, що сигнали, призначені для збудження переривань (сигнальних та періодичних), поступають на вхід контролера переривань не безпосередньо, а через розряд D7 регістра С, який виконує функції прапора переривань. Програма обробки переривань повинна скинути прапор переривання, інакше подальші надходження сигналів переривань будуть заблокованими. Скид прапора переривань здійснюється шляхом читання регістру С.

Для читання та зміни показників RTC передбачено переривання BIOS 1Ah, функції якого звертаються безпосередньо до КМОП-пам’яті і дозволяють не тільки отримати чи встановити час та дату, але й керувати будильником.

Основні функції переривання BIOS **1Ah**:

**00h** – отримання системного часу;

**01h** – встановлення системного часу;

**02h** – отримання часу від CMOS-годинника реального часу;

**03h** – встановлення часу у CMOS-годиннику реального часу;

**04h** – отримання дати від CMOS-календаря реального часу;

**05h** – встановлення дати у CMOS-календаря реального часу;

**06h** – встановлення будильника у CMOS-годиннику реального часу;

**07h** – відміна будильника у CMOS-годиннику реального часу.

Є ще переривання 4Ah, яке служить для перехоплення прикладною програмою сигнала від будильника RTC. INT 4Ah включено у системний обробник переривання 70h від будильника в RTC. Системний обробник цього переривання фактично виконує лиш команду IRET; прикладна програма може встановити власний обробник переривання 4Ah, який буде активізуватись сигналом будильника.

Як видно з таблиці 1, адреси СMOS 10h-2Dh захищено контрольною сумою, яка зберігається за адресою 2Eh. Тому зміна вмісту даних чарунок пам’яті необхідно супроводжувати зміною і контрольної суми.

Порт 70h використовується не тільки для індексування адреси CMOS-чарунки, але й для дозволу/заборони NMI (немасковане переривання). Якщо біт D7 скинуто, то NMI дозволено, якщо встановлено – NMI заборонено.

Читати і писати CMOS також можна за допомогою асемблера.

sseg segment stack

db 256 dup(?)

sseg ends

dseg segment

error\_message db 'Unhandled error occured.$'

dseg ends

cseg segment

assume ss:sseg, cs:cseg, ds:dseg

start:

jmp main

main:

mov ax,seg dseg

mov ds,ax

mov cx, 100

;wait\_rtc:

;

; mov al,0Ah

; out 70h,al

; in al,71h

; test al,10000000b ; Test if ready

; jnz read\_rtc

;

; loop wait\_rtc

;

; jmp erop

read\_rtc:

mov al,07h

out 70h,al

in al,71h

mov dl,al

; add dl, '0'

mov ah,2

int 21h

exit:

; Exit all good

mov ax,4c00h

int 21h

erop:

; Show error message

mov dx,offset error\_message

mov ah,9

int 21h

jmp exit

cseg ends

end start

# Задача 1

Написати з використанням циклів і оформити згідно вимог програму на асемблері для рисування вказаним кольором прямокутника заданого розміру у місці екрану, заданому координатами лівої верхньої вершини за допомогою мишки. Ширина 75, висота 45, колір 5(magenta).

Лістінг t1.asm

; Constants

REC\_WIDTH equ 75

REC\_HEIGHT equ 45

REC\_COLOR equ 5

sseg segment stack

db 256 dup(?)

sseg ends

dseg segment

start\_x dw ?

exit\_message db "Press any key to exit...", "$"

dseg ends

cseg segment

assume ss:sseg, cs:cseg, ds:dseg

start:

jmp main

main:

mov ax,seg dseg

mov ds,ax

; VGA Mode

mov ax,0012h

int 10h

; Mouse

mov ax,0

int 33h

; Show cursor

mov ax,1

int 33h

ms:

mov ax,3 ; Read mouse state

int 33h

test bx,001b ; Test if left button not pressed

jz ms

mov ax,2 ; Hide cursor

int 33h

mov start\_x,cx ; X

mov si,cx

add si,REC\_WIDTH

mov di,dx ; Y

add di,REC\_HEIGHT

; Draw rect

mov ah,0ch ; Pixel output mode

mov bh,0 ; Page

mov al,REC\_COLOR

draw\_line:

mov cx,start\_x ; Start from start\_x

draw\_pixel:

int 10h

inc cx ; Next column

cmp cx,si

jl draw\_pixel

inc dx ; Next row

cmp dx,di

jl draw\_line

; Print exit message

mov ah,09

lea dx,exit\_message

int 21h

; Wait user input

mov ah,08h

int 21h

; Clean view port(FreeDOS)

mov ax,0003h ; Text mode

int 10h

; Exit all good

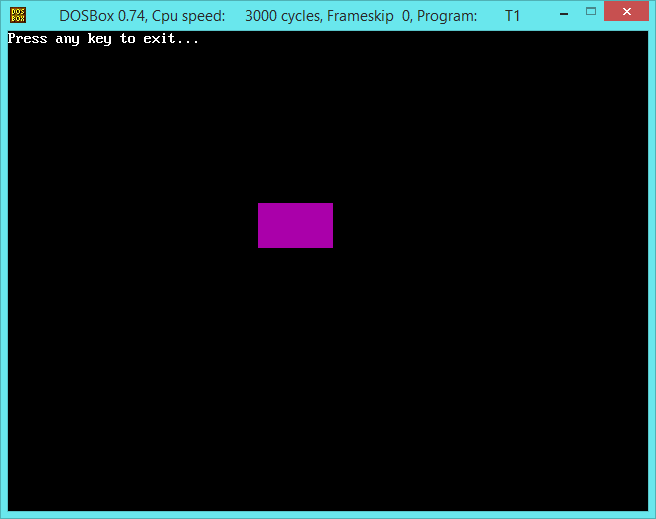
mov ax,4c00h

int 21h

cseg ends

end start

### Результат



## Задача 2

Написати EXE-програму, в якій за допомогою функцій DOS створити каталог CHALIY, створити файл з іменем MIKE, записати VASILIY, закрити файл, перейти в кореневий каталог. Передбачити контроль помилок.

Лістінг t2.asm

sseg segment stack­­­­­

db 256 dup(?)

sseg ends

dseg segment

directory\_name db 'CHALIY',0

file\_name db 'MIKE',0

text db 'VASILIY'

text\_len dw 7

root\_directory db 'C:\',0

error\_message db 'Unhandled error occured.$'

dseg ends

cseg segment

assume ss:sseg, cs:cseg, ds:dseg

start:

jmp main

main:

mov ax,seg dseg

mov ds,ax

; Create new folder

mov dx,offset directory\_name

mov ah,39h

int 21h

jc erop

; Change directory

mov dx,offset directory\_name

mov ah,3bh

int 21h

jc erop

; Create new file

mov dx,offset file\_name

mov cx,0 ; No attributes

mov ah,5bh ; Create and open

int 21h

jc erop

mov bx,ax ; Store file handle

; Write text

mov dx,offset text

mov cx,text\_len

mov ah,40h

int 21h

jc erop

; Close file

mov ah,3eh

int 21h

jc erop

; Chand directory to root

mov dx,offset root\_directory

mov ah,3bh

int 21h

jc erop

exit:

; Exit all good

mov ax,4c00h

int 21h

erop:

; Show error message

mov dx,offset error\_message

mov ah,9

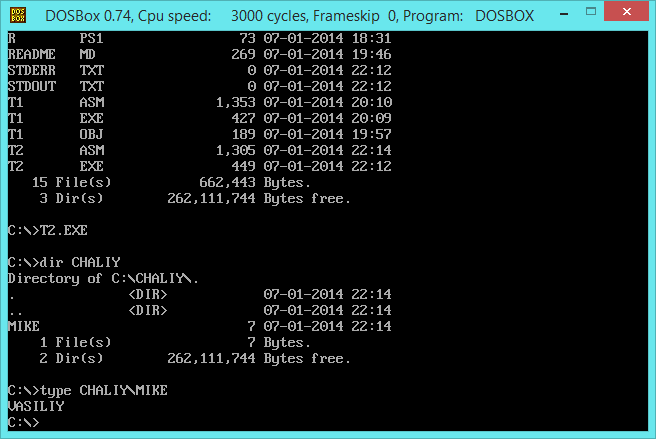
int 21h

jmp exit

cseg ends

### end start

### Результат



Висновки

На основі даної розрахункової роботи я дізнався про структуру програми на асемблері, сегменти, дані та команди, закріпив навички з програмування консольних програм, а також роботи з файлами і каталогами.