《计算机组成原理实验》

实验报告

（数码时钟）

学 院 名 称 ：	数据科学与计算机学院
专业（班级）：	17 软件工程 4 班
学 生 姓 名 ：	郑佳豪
学　　号	16305204
时　　间 ：	2018　年　11　月　04　日

X86汇编语言程序设计实验：数码时钟

一. 实验目的

1. 初步认识和掌握x86汇编语言程序设计的基本方法。
2. 熟悉使用 emu8086 和 DOSBox 等工具。

二. 实验内容

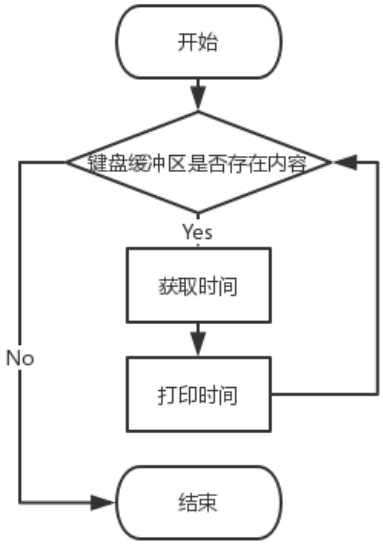
使用 x86 汇编语言编写数码时钟程序，并能在 DOSBox 成功运行：动态显示时钟内容（时、分、秒），并且在接收到任意键盘信号时，退出程序。

三. 实验器材

- 电脑
- Visual Studio Code
- emu8086
- DOSBox

四. 实验过程与结果

程序流程图



设计的思想与方法

通过自顶向下的设计方法，可将程序划分为三大模块：获取时间、打印时间、中断处理。在划分模块后，可进而落实各模块的实现与测试。

实验步骤

由上面的分析可知，实验可分为三大部分，故实验步骤可大致划分为三大部分。

1. 获取时间
2. 打印时间
3. 中断处理

在获取时间模块中，我们通过 int 21h 的 2Ch 指令实现系统时间的读取，读取的时、分、秒分别存储于 CH、CL、DH 寄存器。

在打印时间模块中，我们需要对数值的个位、十位进行拆分，以实现数字字符的打印，故我们使用了 div 指令求取数值与10的商和余数。在得到个位、十位数值后，我们通过逐行定点打印数据段中的数字字符数据，实现了时、分、秒的显示。

在中断处理模块中，我们通过 int 16h 的 01h 指令获取键盘缓冲区的状态，进而根据缓冲区是否存在数据来判断是否中断程序。

实验结果

经过多次实验，实验代码能显示系统时间（时、分、秒），同时能正确响应键盘的中断信号，实验结果满足实验要求内容。下面给出其中一次测试结果。

五. 实验心得

总体来说，本次实验是成功的。经过这次实验，我能基本掌握x86汇编程序设计的基本方法，能熟悉 emu8086 和 DOSBox 工具的使用。由于之前未系统学习x86相关知识，我在实验过程中遇到了不少的问题，这里我一一记录一下。

在实现键盘中断程序的模块时，我走了不少的弯路，一开始想通过 int 21h 的 25h 指令实现预期功能，但始终无法成功。后来，又转而使用了 int 21h 的 0Ah 指令，想通过判断读取的缓冲数据来实现预期功能，但也无法成功。最后，StackOverflow 的一些回答给了我一些启发，我通过在程序的主循环中判断键盘缓冲区的有无数据，来判断用户是否触发键盘事件，进而判断是否中断程序。

在编写代码的过程中，为了使开发流程变得清晰，我先后使用了多种方式组织代码，如注释、空行等，后来发现通过标签和缩进的方式划分代码，是一种很好的方式。

在实验过程中，我感受到了x86相较于优越MIPS的地方，如在寄存器管理和子程序跳转的方面，x86提供了一种对更友好的开发方式。但开发者还是得注意寄存器的管理，要避免寄存器的相互污染。在开始开发 print_digit 子模块的过程中，由于自己开发的不细致，没有注意到存储数值的个位、十位的寄存器与子模块的临时寄存器的冲突，出现了数值打印异常的问题。

这次的课程设计，我学到了不少的底层知识，对计算机的底层原理有了更进一步的了解，希望自己能在日后的计算机组成原理课程中，能够学习与掌握更多相关知识。

【程序代码】

```
.stack 100h
.data
  zero  db "  ___  ", 10
        db " / _ \ ", 10
        db "| | | |", 10
        db "| | | |", 10
        db "| |_| |", 10
        db " \___/ ", 10, "$"

  one   db "  __  ", 10
        db " /_ | ", 10
        db "  | | ", 10
        db "  | | ", 10
        db "  | | ", 10
        db "  |_| ", 10, "$"

  two   db " ___  ", 10
        db "|__ \ ", 10
        db "   ) |", 10
        db "  / / ", 10
        db " / /_ ", 10
        db "|____|", 10, "$"

  three db " ____  ", 10
        db "|___ \ ", 10
        db "  __) |", 10
        db " |__ < ", 10
        db " ___) |", 10
        db "|____/ ", 10, "$"

  four  db " _  _    ", 10
        db "| || |  ", 10
        db "| || |_ ", 10
        db "|__   _|", 10
        db "   | |  ", 10
        db "   |_|  ", 10, "$"

  five  db " _____ ", 10
        db "| ____|", 10
        db "| |__  ", 10
        db "|___ \ ", 10
        db " ___) |", 10
        db "|____/ ", 10, "$"
```

```
six   db "    __  ", 10
      db "  / /  ", 10
      db " / /_  ", 10
      db "| '_ \ ", 10
      db "| (_) |", 10
      db " \___/ ", 10, "$"

seven db " ______ ", 10
      db "|____  |", 10
      db "    / / ", 10
      db "   / /  ", 10
      db "  / /   ", 10
      db " /_/    ", 10, "$"

eight db "  ___  ", 10
      db " / _ \ ", 10
      db "| (_) |", 10
      db " > _ < ", 10
      db "| (_) |", 10
      db " \___/ ", 10, "$"


nine  db "  ___  ", 10
      db " / _ \ ", 10
      db "| (_) |", 10
      db " \__, |", 10
      db "   / / ", 10
      db "  /_/  ", 10, "$"

colon db "     ", 10
      db "   _ ", 10
      db "  (_)", 10
      db "     ", 10
      db "   _ ", 10
      db "  (_)", 10, "$"

line db 0
column db 6
page_number db 0
digit_unit db 0
digit_ten db 0
time db 0
hour db 0
minute db 0
```

```
  second db 0
  current_hour db 0
  current_minute db 0
  current_second db 0
  digit_pointer dw 11 dup(?)

.code
  init:
    mov ax, data
    mov ds, ax

    mov digit_pointer[0], offset zero
    mov digit_pointer[2], offset one
    mov digit_pointer[4], offset two
    mov digit_pointer[6], offset three
    mov digit_pointer[8], offset four
    mov digit_pointer[10], offset five
    mov digit_pointer[12], offset six
    mov digit_pointer[14], offset seven
    mov digit_pointer[16], offset eight
    mov digit_pointer[18], offset nine
    mov digit_pointer[20], offset colon


    ;mov ah, 25h
    ;mov al, 23h
    ;mov dx, fim
    ;int 21h

  main:
    main__read_keyboard_buffer:
      mov ah, 01h
      int 16h
      jnz exit

    main__load_time:
      mov ah, 2Ch
      int 21h

      mov current_hour, ch
      mov current_minute, cl
      mov current_second, dh

    main__print_second:
      mov al, current_second
```

```
    cmp second, al
    jne print

  main__print_minute:
    mov al, current_minute
    cmp minute, al
    jne print

  main__print_hour:
    mov al, current_hour
    cmp hour, al
    jne print

  jmp main

print:
  print__update_hour:
    mov al, current_hour
    mov hour, al

  print__update_minute:
    mov al, current_minute
    mov minute, al

  print__update_second:
    mov al, current_second
    mov second, al

  print__clear_screen:
    call clear_screen

  print__hour:
    mov al, current_hour
    mov time, al
    call parse_time

    call set_digit
    mov column, 1
    call print_digit

    mov al, digit_unit
    call set_digit
    mov column, 11
    call print_digit
```

```
  print__first_colon:
    mov al, 0Ah
    call set_digit
    mov column, 21
    call print_digit

  print__minute:
    mov al, current_minute
    mov time, al
    call parse_time

    call set_digit
    mov column, 31
    call print_digit

    mov al, digit_unit
    call set_digit
    mov column, 41
    call print_digit

  print__second_colon:
    mov al, 0Ah
    call set_digit
    mov column, 51
    call print_digit

  print__second:
    mov al, current_second
    mov time, al
    call parse_time

    call set_digit
    mov column, 61
    call print_digit

    mov al, digit_unit
    call set_digit
    mov column, 71
    call print_digit

  jmp main

parse_time:
```

```
    mov ah, 0
    mov al, time
    mov bl, 10
    div bl
    mov digit_ten, al
    mov digit_unit, ah

    ret

  set_digit:
    mov bl, 2
    mul bl
    mov si, ax
    mov si, digit_pointer[si]

    ret

  print_digit:
    print_digit__reset:
      mov line, 2
      call set_cursor

    print_digit__digit:
      mov dh, 0
      mov dl, ds:[si]

      cmp dx, "$"
      je print_digit__end

      cmp dx, 10
      je print_digit__new_line

      mov ah, 2
      int 21h

      inc si
      jmp print_digit__digit

    print_digit__new_line:
      inc line
      call set_cursor
      inc si
      jmp print_digit__digit
```

```
    print_digit__end:
      ret

  set_cursor:
    mov ah, 2
    mov bh, page_number
    mov dh, line
    mov dl, column
    int 10h

    ret

  clear_screen:
    mov ah, 0fh
    int 10h
    mov ah, 0
    int 10h

    ret

  exit:
    exit__clear_screen:
      call clear_screen

    exit__flush_keyboard_buffer:
      mov ah, 0Ch
      mov al, 0
      int 21h

    exit__signal:
      mov ax, 4C00h
      int 21h

end init
```