汇编语言与接口技术课程实验报告
课内实验

软件 42
学号 2141601026
陈铮

2016 年 11 月 27 日

目录

第一章　汇编语言的上机环境及调试方法

1.1　实验题目

搭建汇编语言运行环境。

在屏幕上打印“hello, world”。

1.2　问题分析与实验设计

1.2.1　搭建环境

本人使用 Windows10 系统，搭建汇编的开发环境需要如下组件：

- 汇编器 Assembler

  Intel x86 汇编的汇编器可以使用 **MASM** [1]，本人使用了代号为 **masm32v11r** 的版本。

- 链接器 Linker

  链接器被包含在上述 SDK 中，针对生成 16 位的可执行文件的要使用 link16.exe 。

- 运行环境 Runtime Environment

  DOSBOX [2]可以模拟 16 位的运行环境，本人使用其 0.74 版本。

- 调试器 Enhanced Debug

  使用 Debug [3]作为汇编程序的调试工具，本人使用其一个升级版本。

安装与配置

将上述组件全部下载并使用默认设置安装：

结果汇编工具的执行目录在 **C:\masm32\bin**，而 DOSBOX 的执行目录在 **C:\Program Files (x86)\DOSBox-0.74**。

并将 DEBUG 压缩包中的文件解压到 **C:\debug**。

然后将汇编工具的执行目录以及 DOSBOX 的执行目录添加到 Windows 系统的环境变量 PATH 中，打开命令行测试 **ml** 与 **dosbox** 命令有效。

由于 DOSBOX 也是一个子系统，拥有自己的环境变量，需要将 DEBUG 的目录配置到 DOSBOX 的环境变量 PATH 中。

运行 DOSBOX 目录下的 **DOSBox 0.74 Options.bat** 批处理文件，将打开一个配置文件，在文件最后的 **autoexec** [4] 域后面附加两行配置：

[1]MASM: Microsoft Macro Assembler 微软宏汇编器，但自 MASM 6 开始，MASM 被集成到 VC 系列的工具链中不单独发布了。分离版本的 MASM SDK 由一群爱好者维护升级，见 http://masm32.com

[2]DOSBOX: 可以在各种系统下模拟 DOS 运行环境，提供 16 位程序的运行环境，见 http://www.dosbox.com。

[3]DEBUG: 著名的程序调试器，具有反汇编、单步调试等功能，见 https://sites.google.com/site/pcdosretro/enhdebug

[4]autoexec 是 DOSBOX 启动后自动运行的脚本

代码 1.1: DOSBOX AutoExec 配置

```
MOUNT D: C:\debug
SET PATH=D:\
```

表示将 Windows 中 DEBUG 所在的目录挂载到 DOSBOX 中的 D 盘分区，然后将 D 盘根目录添加到 DOSBOX 的环境变量 PATH 中。

成功后打开 DOSBOX 可以直接执行命令 **DEBUG** 来启动 DEBUG。

用编辑器编写汇编代码

新建一个文件夹，创建并编辑其中的 main.asm 文件，本人使用 **vim** 作为代码编辑器。

```
$ mkdir 1
$ cd 1
$ vim main.asm
```

向 main.asm 中写入 “hello, world” 程序代码：

代码 1.2: hello, world

```
option casemap: none
.model small
.data
hello db 'hello,␣world$'
.code
main proc far
  mov ax, @data
  mov ds, ax
  mov ah, 09h
  lea dx, hello
  int 21h
  mov ax, 4c00h
  int 21h
  ret
main endp
end main
```

汇编、链接与运行

编译与链接使用两个命令完成：

代码 1.3: 汇编项目 Build 脚本——make.bat

```
$ ml -c *.asm && link16 main.obj,main.exe,nul.map,.lib,nul.def
```

为了以后的模块化编程，也可以封装一个 Build 脚本——**make.bat**

在 DOSBOX 命令后附加可执行文件的名字即可运行：

```
$ dosbox main.exe
```

1.3　结果分析

在 DOSBOX 下运行 MAIN.EXE 得到结果如下：

```
C:\>MAIN.EXE
hello, world
```

查阅 INT 21H / AH = 09H 功能即可了解向控制台输出字符串的详细说明。

第二章　算术逻辑运算程序设计方法

2.1　实验题目

求 100 以内能被 7 整除的所有正数之和。

2.2　问题分析与实验设计

解决这个问题我们有若干种思路：

1. 穷举法

   100 以内的正数是一个有限的集合，因此可以穷举其中的所有元素，验证其是否能被 7 整除，如果是则加到累加器上。

   100 个数要做 100 次除法，过于暴力，效率太低，不予采纳。

2. 递推法

   7 是最小的符合题设条件的数，设为 A_0，以及从当前能被 7 整除的数 A_i 推出下一个能被 7 整除的数 $A_{i+1} = A_i + 7$。因此可以直接从 7 开始每次加 7，将不超过 100 的所有经过的数都加到累加器里即可。

   这个方法只需要涉及加法，但总的来说算法复杂度还是线性的 $O(n)$，不予采纳。

3. 公式法

   由简单的数学推导得到一个更一般的结论：在区间 $[0, a]$ 中，能被 n 整除的整数之和为：

   $$\frac{n\lfloor \frac{a}{n} \rfloor (\lfloor \frac{a}{n} \rfloor + 1)}{2}$$

   代入 $a = 100, n = 7$ 可直接得出答案为 735。

   下取整除法在具体实现中比较简单，而且配合 DX, AX 可以组装出一个 32 位的乘法。

   代码 2.1: 精彩片段

```
1    ; unsigned divide 32-bit result by 2
2    shr dx, 1
3    rcr ax, 1
```

   这里巧妙地利用了循环移位来连接两个寄存器的数据。

2.3　结果分析

代码 2.2: DEBUG: 实验二运行结果

```
-u
06CA:0000 B8CC06            MOV     AX,06CC
06CA:0003 8ED8              MOV     DS,AX
06CA:0005 BA0000            MOV     DX,0000
06CA:0008 A10600            MOV     AX,[0006]
06CA:000B F7360800          DIV     WORD PTR [0008]
06CA:000F BA0000            MOV     DX,0000
06CA:0012 8BD8              MOV     BX,AX
06CA:0014 A10800            MOV     AX,[0008]
06CA:0017 F7E3              MUL     BX
06CA:0019 43                INC     BX
06CA:001A F7E3              MUL     BX
06CA:001C D1EA              SHR     DX,1
06CA:001E D1D8              RCR     AX,1
-g20
AX=02DF BX=000F CX=002A DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=06CC ES=06BA SS=06C9 CS=06CA IP=0020 NV UP EI PL ZR AC PE NC
06CA:0020 B8004C            MOV     AX,4C00
-g

Program terminated (0000)
-q
```

当运行到 06CA:001E 时，程序进入将要结束的状态，此时 (DX AX) 的值是 000002DF，即 16 进制的 735，答案正确。

附录：完整代码清单

代码 2.3: 求 100 以内能被 7 整除的数的和

```
option casemap: none
.model small
.data
a dw 100
n dw 7
.code
main proc far
  mov ax, @data
  mov ds, ax

  mov dx, 0
  mov ax, a
```

```
13    div n
14
15    mov dx, 0
16    mov bx, ax
17    mov ax, n
18    mul bx
19    inc bx
20    mul bx
21    ; unsigned divide 32-bit result by 2
22    shr dx, 1
23    rcr ax, 1
24
25    mov ax, 4c00h
26    int 21h
27    ret
28  main endp
29  end main
```

第三章　DOS/BIOS 中断程序设计

3.1　实验题目

编写一个简单模拟钢琴的程序，在屏幕上显示钢琴键位，通过键盘控制钢琴。

3.2　问题分析与实验设计

1. 显示一段文字

   使用 INT 21H / AH = 09H 功能来输出字符串。

   利用数据区定义变量默认的连续性，可以简洁美观地定义多行字符串输出。

2. 延迟

   反复检查 61H 端口来实现延迟功能。每 15.085μs，从 61H 端口输入的数据中的低 5 位会发生一次改变，监听这个改变来完成计时。

   如延迟 1ms 需要监测到 **约** 66 次这样的变化。

   可以封装一个以毫秒 (ms) 为单位的过程来处理这个延迟。

3. 播放音乐

   通过将频率处理后发送至 42H 来使扬声器发声/禁声。

   首先要先将控制部分（端口号 43H）初始化，然后再与 42H 端口进行通讯。

4. 键盘输入

   使用 INT 16H / AH = 00H 功能来从键盘缓冲区读取一个键，无回显。

3.3　结果分析

代码 3.1: 钢琴模拟屏幕显示

```
1      PIANO
2   2 3    5 6 7
3  q w e r t y u
4    s d    g h j
5  z x c v b n m
6  Press ESC to exit
```

当按下屏幕上所示的键时，会发出对应音调的音符。

附录：完整代码清单

代码 3.2: 钢琴模拟主模块 main.asm

```
option casemap: none
.model small
.stack 200H
.data
welcome db '␣␣␣␣PIANO␣␣␣␣', 13, 10
        db '␣2␣3␣␣␣5␣6␣7␣', 13, 10
        db 'q␣w␣e␣r␣t␣y␣u', 13, 10
        db '␣s␣d␣␣␣g␣h␣j␣', 13, 10
        db 'z␣x␣c␣v␣b␣n␣m', 13, 10
        db 'Press␣ESC␣to␣exit$'

well dw 262, 1000
     dw 0, 500
     dw 262, 1000
     dw 0, 500
     dw 294, 2000
     dw 0, 500
     dw 262, 2000
     dw 0, 500
     dw 349, 2000
     dw 0, 500
     dw 330, 4000
     dw 0, 500
     dw 0, 0

keyno db 'zsxdcvgbhnjmq2w3er5t6y7u'
fre   dw 220, 233, 246, 261, 277, 293, 311, 329, 349, 369, 391, 415
      dw 440, 466, 493, 523, 554, 587, 622, 659, 698, 739, 783, 830

.code
extrn delay: near
extrn init: near
extrn play: near
extrn music: near

main proc far
  mov ax, @data
  mov ds, ax
  mov es, ax
  ; display welcome
  mov ah, 09h
```

```
42    lea dx, welcome
43    int 21h
44    ; init
45    call init
46    ; well
47    lea dx, well
48    push dx
49    call music
50    pop dx
51    ; keyboard
52    L1S:
53      mov ah, 00h
54      int 16h
55      cmp ah, 01h
56      jz L1E
57      lea di, keyno
58      mov cx, lengthof keyno
59      cld
60      repne scasb
61      jnz L1S
62      mov ax, 1000
63      push ax
64      lea si, fre
65      mov bx, lengthof keyno - 1
66      sub bx, cx
67      shl bx, 1
68      mov ax, [si + bx]
69      push ax
70      call play
71      add sp, 4
72      jmp L1S
73    L1E:
74    mov ax, 4c00h
75    int 21h
76    ret
77  main endp
78  end main
```

代码 3.3: 钢琴模拟播放模块 play.asm

```
1  option casemap: none
2  .model small
3  .stack 10
4  .code
5  extrn delay: near
6  ; public void init()
```

```
7  ; init the player
8  public init
9  init proc
10   mov al, 0B6h
11   out 43h, al
12   ret
13 init endp
14
15 ; public void play(int f, int d)
16 ; stack structure
17 ; [bp + 6]: duration dw (ms)
18 ; [bp + 4]: frequency dw (Hz)
19 ; [bp + 2]: caller IP
20 ; [bp + 0]: caller BP
21 public play
22 play proc
23   push bp
24   mov bp, sp
25   ; compute divisor
26   mov dx, 0012H
27   mov ax, 34DEH
28   mov bx, [bp + 4]
29   div bx
30   ; pass divisor
31   out 42h, al
32   mov al, ah
33   out 42h, al
34   ; turn on
35   in al, 61h
36   mov ah, al
37   or al, 11b
38   out 61h, al
39   ;delay
40   push ax
41   mov ax, [bp + 6]
42   push ax
43   call delay
44   pop ax
45   pop ax
46   ; turn off
47   mov al, ah
48   out 61h, al
49   ; exit
50   pop bp
```

```
51    ret
52  play endp
53
54  ; public void music(int *music)
55  ; stack structure
56  ; [bp + 4]: music: the address of music
57  ; [bp + 2]: caller IP
58  ; [bp + 0]: caller BP
59  public music
60  music proc
61    push bp
62    mov bp, sp
63    mov si, [bp + 4]
64    L1S:
65      mov ax, [si + 2]
66      cmp ax, 0
67      jz L1E
68      push ax
69      mov ax, [si]
70      cmp ax, 0
71      jnz T1
72        ; if f == 0
73        call delay
74        add sp, 2
75        jmp T2
76      T1:
77        ; if f > 0
78        push ax
79        call play
80        add sp, 4
81      T2:
82      add si, 4
83      jmp L1S
84    L1E:
85    pop bp
86    ret
87  music endp
88  end
```

代码 3.4: 钢琴模拟延迟模块 delay.asm

```
1  option casemap: none
2  .model small
3  .stack 6
4  .code
5  ; public void delay(times)
```

```
6  ; delay times in ms
7  ; stack structure:
8  ; [bp + 4]: times
9  ; [bp + 2]: caller IP
10 ; [bp + 0]: caller BP
11 public delay
12 delay proc
13   push bp
14   mov bp, sp
15   push ax
16   push cx
17   push dx
18
19   mov ax, [bp + 4]
20   mov cx, 66 ; 1ms = 66 * 15.085 \mu s
21   mul cx
22   mov cx, ax
23   ; (dx cx) cycles, each 15.085 \mu s
24   L1:
25     in al, 61h
26     and al, 10h
27     cmp al, ah
28     je L1
29     mov ah, al
30     loop L1
31
32     cmp dx, 0
33     jz EXIT
34     dec cx
35     dec dx
36     jmp L1
37   EXIT:
38   pop dx
39   pop cx
40   pop ax
41   pop bp
42   ret
43 delay endp
44 end
```

第四章　模块化程序设计及接口应用技术

4.1　实验题目

编写一个迷你射击游戏，有计时、计分功能与子弹飞行动画效果。

4.2　问题分析与实验设计

4.2.1　主模块

主模块要尽量地简短。在游戏类项目中，主模块通常只做初始化 (Initialization) 与注册主循环 (Mainloop)。

初始化

初始化包括**设置视频模式**，**设置光标格式**，**初始化目标**以及**初始化计时器**这四个部分。这些部分都不是需要在每一个回合中处理的。

- 设置视频模式

  使用 INT 10H / AH = 00H 来设置视频模式。这里使用了 AL = 03H 设置了 80×25 16 色文本格式。

- 设置光标格式

  使用 INT 10H / AH = 01H 来设置光标格式。这里使用了 CX = 0100H 来隐藏光标 (满足 CH > CL 即可)。

- 初始化目标

  在屏幕上画出目标。这个功能封装到了目标模块（见章节4.2.3）中。

- 初始化计时器

  计时器清零。这个功能封装到了计时模块（见章节4.2.2) 中。

注册主循环

在游戏中，玩家会不断采取行动。从设计上，这很可能是一个无限的循环。每一次循环都检查处理完整的事务，也称为一个回合。而一个回合中所有要执行的过程要添加到主循环中，这称为**注册主循环**。

代码 4.1: 注册主循环片段

```
1   mainloop:
2     ; display score
3     set_cursor 2, 2
4     PrintString PROMPT1
```

```
5      PrintWordDec score
6      ; display time
7      set_cursor 2, 66
8      PrintString PROMPT2
9      call TimerDisplay
10     ; display player
11     call PlayerDisplay
12     ; call player
13     call PlayerController
14     jmp mainloop
```

可以看到游戏的主逻辑十分清晰：在每一个回合里显示两个板块——**计分**、**计时**；然后显示玩家；然后将控制权移交给玩家的控制器。

4.2.2　计时模块

计时模块给出三个方法：**清空计时器**、**取计时器的值**、打印默认格式的时间字符串。

使用 INT 21H / AH = 2CH 功能可以获取系统的时间戳，将其保存在模块级变量中可以对其他模块隐藏，保证良好的封装。

计时模块只需要保存一个时间戳即可实现一个计时器的功能，取计时器值的时候只要将获得的值与之前保存的值作差即可得到时间间隔。

4.2.3　目标模块

目标模块是保存射击目标的位置、属性，并给出一个方法来绘制它们。

使用 INT 10H / AH = 09H 功能来写带属性的字符。

4.2.4　玩家模块

玩家模块保存玩家的位置、得分，玩家自身与子弹的样式，并给出了绘制玩家的方法与**键盘路由**。

控制器：键盘路由

利用 INT 21H / AH = 06H, DL = FFH 功能获取键盘输入后跑一个分发路由。

代码 4.2: 键盘路由片段

```
1  public PlayerController
2  PlayerController proc
3    ; keyboard
4    mov ah, 06h  ; direct console input [or output]
5    mov dl, 0ffh ; for input
6    int 21h
7    ; input router
8    cmp al, 'e' ; e for exit
9    je InputE
10   cmp al, '␣' ; space for shoot (disappear)
11   je InputSpace
12   cmp al, 4bh ; (left)
```

```
13    je InputLeft
14    cmp al, 4dh ; (right)
15    je InputRight
16    ret
17    ; routines
18    InputE:
19      mov ax, 4c00h
20      int 21h
21      ret
22    InputLeft:
23      set_cursor row, col
24      putchar '␣'
25      dec col
26      ret
27    InputRight:
28      set_cursor row, col
29      putchar '␣'
30      inc col
31      ret
32    InputSpace:
33      call shoot
34      ret
35  PlayerController endp
```

局部读擦写：发射子弹与碰撞检测

利用移动光标和在光标处写字符很容易局部读写屏幕，实现子弹飞行的动画效果。

利用 INT 10H / AH = 08H 来从屏幕上读取字符来做碰撞检测。

代码 4.3: 子弹飞行与碰撞检测

```
1   shoot proc
2     ; generate bullet at (row - 1, col)
3     mov dh, row
4     mov dl, col
5     dec dh
6     set_cursor dh, dl
7     putchar bullet
8     ; let it fly
9     ; do while
10    flyingS:
11      cmp dh, 0
12      jl flyingE ; out of range
13
14
15      set_cursor dh, dl
```

```
16      putchar '␣'; clean old
17      dec dh
18      ; read @bullet judge
19      set_cursor dh, dl
20      mov ah, 08h
21      mov bh, 00h
22      int 10h
23      cmp al, target
24      jnz notHit
25        inc score
26        putchar '␣'
27        ret
28      notHit:
29
30        cmp dh, 0
31        jz flyingT1
32          set_cursor dh, dl
33          putchar bullet
34          push t
35          call delay
36          pop t
37        flyingT1:
38      jmp flyingS
39    flyingE:
40    ret
41  shoot endp
```

4.2.5　延迟模块

复用了实验三用过的延迟模块（见章节2）。

4.3　结果分析

玩家可以用键盘控制：

- 左/右方向键: 控制玩家左/右平移
- 空格: 射击
- e: 退出

左上角显示分数，右上角显示时间。

附录：完整代码清单

代码 4.4: 射击游戏：主模块 main.asm

```
option casemap: none
.model small

.data
PROMPT1 DB 'SCORE:$'
PROMPT2 DB 'TIME:$'

public score
score dw 0
.stack 200H

.code
include macro.inc
extrn TimerClear: near
extrn TimerPeek: near
extrn TimerDisplay: near
extrn PlayerController: near
extrn PlayerDisplay: near
extrn TargetDisplay: near
placeTarget proc
placeTarget endp

main proc far
  mov ax, @data
  mov ds, ax
  ; init
  clear_screen 1, 1, 23, 78
  ; 80 * 25 16 colors text video mode
  mov ah, 0
  mov al, 03h
  int 10h
  ; hide cursor
  mov cx, 0100h
  mov ah, 1
  int 10h
  call TargetDisplay
  call TimerClear
  mainloop:
    ; display score
    set_cursor 2, 2
    PrintString PROMPT1
    PrintWordDec score
    ; display time
```

```
44      set_cursor 2, 66
45      PrintString PROMPT2
46      call TimerDisplay
47      ; display player
48      call PlayerDisplay
49      ; call player
50      call PlayerController
51      jmp mainloop
52    ret
53  main endp
54  end main
```

代码 4.5: 射击游戏：目标模块 target.asm

```
1   option casemap: none
2   .model small
3   .data
4   public target
5   target db 'V'
6   targetCnt dw 7
7   targets db 5, 20, 1
8           db 5, 22, 2
9           db 5, 24, 3
10          db 5, 26, 4
11          db 6, 32, 5
12          db 6, 20, 6
13          db 6, 21, 7
14  .code
15  include macro.inc
16
17  public TargetDisplay
18  TargetDisplay proc
19    lea si, targets
20    mov cx, targetCnt
21    L1:
22      push cx
23      mov dh, [si + 0]
24      mov dl, [si + 1]
25      set_cursor dh, dl
26      mov al, target
27      mov bl, [si + 2]
28      mov bh, 0
29      mov cx, 1
30      mov ah, 9
31      int 10h
32      pop cx
```

```
33      add si, 3
34      dec cx
35      jnz L1
36    ret
37  TargetDisplay endp
38  end
```

代码 4.6: 射击游戏：玩家模块 player.asm

```
1   option casemap: none
2   .model small
3   .data
4   row db 22
5   col db 15
6   t dw 100
7   show db 'A'
8   bullet db 1eh
9   extrn target: byte
10  extrn score: word
11  .code
12  include macro.inc
13
14  public PlayerController
15  PlayerController proc
16    ; keyboard
17    mov ah, 06h  ; direct console input [or output]
18    mov dl, 0ffh ; for input
19    int 21h
20    ; input router
21    cmp al, 'e' ; e for exit
22    je InputE
23    cmp al, '␣' ; space for shoot (disappear)
24    je InputSpace
25    cmp al, 4bh ; (left)
26    je InputLeft
27    cmp al, 4dh ; (right)
28    je InputRight
29    ret
30    ; routines
31    InputE:
32      mov ax, 4c00h
33      int 21h
34      ret
35    InputLeft:
36      set_cursor row, col
37      putchar '␣'
```

```
38      dec col
39      ret
40    InputRight:
41      set_cursor row, col
42      putchar '␣'
43      inc col
44      ret
45    InputSpace:
46      call shoot
47      ret
48  PlayerController endp
49
50  public PlayerDisplay
51  PlayerDisplay proc
52    set_cursor row, col
53    putchar show
54    ret
55  PlayerDisplay endp
56
57  extrn delay: near
58  shoot proc
59    ; generate bullet at (row - 1, col)
60    mov dh, row
61    mov dl, col
62    dec dh
63    set_cursor dh, dl
64    putchar bullet
65    ; let it fly
66    ; do while
67    flyingS:
68      cmp dh, 0
69      jl flyingE ; out of range
70
71
72      set_cursor dh, dl
73      putchar '␣'; clean old
74      dec dh
75      ; read @bullet judge
76      set_cursor dh, dl
77      mov ah, 08h
78      mov bh, 00h
79      int 10h
80      cmp al, target
81      jnz notHit
```

```
82        inc score
83        putchar '␣'
84        ret
85      notHit:
86
87        cmp dh, 0
88        jz flyingT1
89          set_cursor dh, dl
90          putchar bullet
91          push t
92          call delay
93          pop t
94        flyingT1:
95      jmp flyingS
96    flyingE:
97    ret
98  shoot endp
99
100 end
```

代码 4.7: 射击游戏：计时模块 timer.asm

```
1  option casemap: none
2  .model small
3  .data
4  Buffer db 4 dup(?)
5  Format db '00:00:00', '$'
6  Radix db 10
7  .code
8  include macro.inc
9
10 public TimerClear
11 TimerClear proc
12   push ax
13   push cx
14   push dx
15   mov ah, 2ch
16   int 21h
17   mov Buffer[0], ch ; hour
18   mov Buffer[1], cl ; minute
19   mov Buffer[2], dh ; second
20   mov Buffer[3], dl ; (10ms)
21   pop dx
22   pop cx
23   pop ax
24   ret
```

```
25 TimerClear endp
26
27 ; return: CH = hour, CL = minute, DH = second, DL = (10ms)
28 public TimerPeek
29 TimerPeek proc
30   push ax
31   mov ah, 2ch
32   int 21h
33   sub dl, Buffer[3]; 100
34   cmp dl, 0
35   jge T1
36     dec dh
37     add dl, 100
38   T1:
39   sub dh, Buffer[2]; 60
40   cmp dh, 0
41   jge T2
42     dec cl
43     add dh, 60
44   T2:
45   sub cl, Buffer[1]; 60
46   cmp cl, 0
47   jge T3
48     dec ch
49     add cl, 60
50   T3:
51   sub ch, Buffer[0]; 24
52   cmp ch, 0
53   jge T4
54     add ch, 24
55   T4:
56   pop ax
57   ret
58 TimerPeek endp
59 ; display HH:MM:SS format String
60 ; using INT 21H / AH = 09H
61 public TimerDisplay
62 TimerDisplay proc
63   push ax
64   push cx
65   push dx
66   call TimerPeek
67   ; display hour
68   mov ah, 0
```

```
69    mov al, ch
70    div Radix
71    add ax, 3030h
72    mov Format[0], al
73    mov Format[1], ah
74    ; display minute
75    mov ah, 0
76    mov al, cl
77    div Radix
78    add ax, 3030h
79    mov Format[3], al
80    mov Format[4], ah
81    ; display second
82    mov ah, 0
83    mov al, dh
84    div Radix
85    add ax, 3030h
86    mov Format[6], al
87    mov Format[7], ah
88    ; print
89    PrintString Format
90    pop dx
91    pop cx
92    pop ax
93    ret
94  TimerDisplay endp
95
96  end
```

代码 4.8: 射击游戏：延迟模块 delay.asm

```
1  option casemap: none
2  .model small
3  .stack 6
4  .code
5  ; public void delay(times)
6  ; delay times in ms
7  ; stack structure:
8  ; [bp + 4]: times
9  ; [bp + 2]: caller IP
10 ; [bp + 0]: caller BP
11 public delay
12 delay proc
13   push bp
14   mov bp, sp
15   push ax
```

```
16    push cx
17    push dx
18
19    mov ax, [bp + 4]
20    mov cx, 66 ; 1ms = 66 * 15.085 \mu s
21    mul cx
22    mov cx, ax
23    ; (dx cx) cycles, each 15.085 \mu s
24    L1:
25      in al, 61h
26      and al, 10h
27      cmp al, ah
28      je L1
29      mov ah, al
30      loop L1
31
32      cmp dx, 0
33      jz EXIT
34      dec cx
35      dec dx
36      jmp L1
37    EXIT:
38    pop dx
39    pop cx
40    pop ax
41    pop bp
42    ret
43  delay endp
44  end
```

代码 4.9: 射击游戏：宏库 macro.inc

```
1   clear_screen macro op1,op2,op3,op4
2     mov ax,0600h
3     mov bh,7h
4     mov ch,op1
5     mov cl,op2
6     mov dh,op3
7     mov dl,op4
8     int 10h
9     set_cursor 0, 0
10  endm
11
12  ; ah, bh, dx
13  set_cursor macro row, column
14    mov ah, 02h
```

```
15    mov bh, 00h
16    mov dh, row
17    mov dl, column
18    int 10h
19  endm
20
21  ; ah, dx
22  PrintString macro string
23    mov ah, 09h
24    lea dx, string
25    int 21h
26  endm
27
28  ; putchar at cursor
29  putchar macro ch
30    mov ah, 09h
31    mov al, ch
32    mov bh, 0
33    mov bl, 0fh
34    mov cx, 1
35    int 10h
36  endm
37
38  putcharColor macro ch, fg
39    push bx
40    push cx
41    mov ah, 9
42    mov al, ch
43    mov bh, 0
44    mov bl, fg
45    mov cx, 1
46    int 10h
47    pop cx
48    pop bx
49  endm
50
51
52  PrintByteDec macro num
53    mov ah, 0
54    mov al, num
55    _PrintDec
56  endm
57
58  PrintWordDec macro num
```

```
  mov ax, num
  _PrintDec
endm

_PrintDec macro
  local L1, L2
  mov dx, 0
  mov bx, 10
  mov cx, 0
  L1:
    div bx
    push dx
    mov dx, 0
    inc cx
    cmp ax, 0
    jne L1
  L2:
    pop ax
    add al, '0'
    mov dl, al
    mov ah, 2
    int 21h
    dec cx
    jnz L2
endm
```

第五章　实验总结

花在汇编实验上地时间着实多，收获也很大。
学过汇编才知道 C 语言的方便、强大简直是划时代的。
总得来说，汇编是一门很**灵活**的语言。
实验做完之后，我已经想到各种编程范式 (面向过程的、面向对象、函数式等) 在其中的实现方法。
学习与实践汇编以后，对系统底层的编程有了更深刻的认识。

5.1　版权声明

CopyRight ©2016 zccz14(陈铮).
遵循 CC-BY-NC 2.0 协议开源传播。
本文采用 LaTeX 排版，全部源代码已上传至 GitHub。
GitHub: https://github.com/zccz14/x86AssemblyReport

代码索引