中南大學

CENTRAL SOUTH UNIVERSITY

计算机组成原理与汇编语言

课程实验报告

题　　目 计算机组成原理与汇编语言课程实验

学生姓名 孙重豪

班级学号 8207191520

指导教师 李仪

设计时间 2021 年 7 月

目 录

第 1 章　实验概述

1.1 实验目的

熟悉 8086 指令系统：数据传送指令、算数指令、逻辑指令、串处理指令、控制转移指令等，掌握子程序的设计方法，熟悉递归程序的设计，了解循环程序的设计方法。

1.2 实验要求

(1) 编写一个累计加法，从 1 加到 5，将结果保存至 AX 中。

(2) 编写一个累计减法，被减数是 10011000B，减数是 01000000B，连续减 5 次， 观察 FLAGS 的变化

(3) 编写一个 16 位的乘法，被乘数是 100H，乘数是 100H，观察 Flags 的变化，编写一个 32 位的乘法，被乘数是 0F0FH，乘数是 FF00H，观察 Flags 的变化。

(4) 编写一个 16 位的除法，被除数是 100H，除数是 100H，观察 Flags 的变化，编写一个 32 位的除法，被除数是 0F0FH，除数是 00FFH，观察 Flags 的变化。

(5) 编写一个累计加法，被加数是 0FH，加数是 01H，观察 Flags 的变化，被加数是 0FFH，加数是 01H，观察 Flags 的变化，被加数是 0FFFH，加数是 01H，观察 Flags 的变化，被加数是 FFFFH，加数是 01H，观察 Flags 的变化，被加数是 FFFFFFFFH 加数是 01H，观察 Flags 的变化。

(6) 编写一个移位运算，将 8F1DH 存至 AX，然后用指令右移 1 位然后左移 1 位， 显示结果并观察 Flags 的变化。将 8F1DH 存至 AX 中，然后带 CF 位左移 5 位，并右移 7 位，观察 Flags 的变化，并给出结果。

(7) 将 71D2H 存至 AX 中，5DF1H 存至 CX 中，DST 为 AX，REG 为 AX，实现双精度右移 2 次，交换 DST 与 REG，然后左移 4 次，分别查看结果.

(8) 实现压缩 BCD 码的加减法，用压缩 BCD 码实现（21+71），（12+49），（65+82），（46-33），（74-58），（43-54）的十进制加减法。然后又用非压缩 BCD 实现上述 6 个式子。

(9) 实现 KMP 算法，输入两个字符串（可以直接保存在内存中），实现快速匹配 。

(10)斐波纳契数列：1，1，2，3，5，8，13。通常可以使用递归函数实现，现用汇编实现该过程。

第 2 章 实验设计与源码

2.1 实验一

2.1.1 实验简述

由于 cx 可以作为通用寄存器保存计数值，可以用做隐含的计数器，因此将 cx 赋值为 5，实现五次相加操作，ax 初始赋值为 0，每次进行循环时，将 cx 的值加在 ax 中。

2.1.2 实验源码

```
code segment
start:
    mov ax,0
    mov cx,5
    next: add ax ,cx
    loop next ;循环 5 次
code ends
end start
```

2.1.3 实验结果

实现五次循环操作达到目的，依次得到结果 ax 为 0005、0009、000C、000E、000F，最终结果为 000F=15。

2.2 实验二

2.2.1 实验简述

将 cx 赋值为 5 用做隐含的计数器，实现五次相减操作，将 ax 赋值为 10011000B，将 bx 赋值为 01000000B，实现循环相减操作。

2.2.2 实验源码

```
code segment
start:
    mov ax ,10011000B
    mov bx ,01000000B
```

```
    mov cx ,5   ;循环次数
    next: sub ax,bx
    loop next
code ends
end start
```

2.2.3 实验结果

进行连减，第一次相减得到 ax=00000058,第二次相减得到 ax=00000018，第三次相减得到 ax=FFFFFFD8，且 PL=1，第四次相减得到 ax=FFFFFF98，且 PL=1，第五次相减得到 ax=FFFFFF58，且 PL=1。

2.3 实验三

2.3.1 实验简述

16 位乘法中直接将 ax,bx 分别赋值为 0100H,实现相乘并存在 ax 中；32 位乘法中将定义在数据段的 00000f0fH 和 0000ff00H 的低位放入 ax，dx 中，并相乘，结果保存在 ax 中,将 ax,dx 入栈，再将 00000f0fH 的高位放入 ax 中并与 dx 相乘，将 ax,dx 入栈，再将 00000f0fH 的低位放入 ax，0000ff00H 的高位放入 dx,相乘将 ax,dx 入栈，00000f0fH 和 0000ff00H 的高位放入 ax，dx 中并相乘将 ax,dx 入栈。

2.3.2 实验源码

```
3_1
code segment
start:
    mov ax ,100h
    mov bx ,100h
    mul bx ;16 位乘法
code ends
end start

3_2
assume cs:code, ds:data
data segment
    x1 dw 0f0fH
    x2 dw 0000H
    y1 dw 0ff00H
    y2 dw 0000H
    xy dw 4 dup (?)
data ends
code segment
```

```
start:
    mov ax,data
    mov ds,ax
    mov ax,x1
    mov dx,y1
    mul dx
    mov [xy],ax
    mov [xy+2],dx      ;被乘数低位 4 字符 x1 和乘数低位 4 字符 y1 相乘结果低位存入 xy,高位存入 xy+2

    mov ax,x2
    mov dx,y1
    mul dx
    add [xy+2],ax
    adc [xy+4],dx       ; 被乘数高位 4 字符 x2 和乘数低位 4 字符 y1 相乘结果低位存入 xy+2,高位存入 xy+4

    mov ax,x1
    mov dx,y2
    mul dx
    add [xy+2],ax
    adc [xy+4],dx
    adc [xy+6],0        ; 被乘数低位 4 个字符 x1 和乘数高位 4 个字符 y2 相乘结果低位存入 xy+2,高位存入 xy+4

    mov ax,x2
    mov dx,y2
    mul dx
    add [xy+4],ax
    adc [xy+6],dx       ; 被乘数高位 4 个字符 x2 和乘数高位 4 个字符 y2 相乘结果低位存入 xy+4,高位存入 xy+6

    mov ah,4ch
    int 21h

code ends
end start
```

2. 3. 3 实验结果

0100H*0100H 得到的结果应为 10000，发生了溢出，观察到 OV=1,CY=1；0f0fh*ff00h 得到的结果为 0EFFF100。

2.4 实验四

2.4.1 实验简述

将被除数 0100H 赋值在 ax 中，除数 0100H 赋值在 cx 中，实现 16 位除法运算，结果除数保存在 ax 中，余数保存在 dx 中，再将 0f0fh 保存在 ax 中，然后 cwd 扩展 ax 到 dx,将 0000h 保存在 dx,ff00h 赋值于 cx 中，实现 32 位除法，除数保存在 ax 中，余数保存在 dx 中。

2.4.2 实验源码

```
4_1
code segment
start:
    mov dx,0000h ; 高位
    mov ax ,0100h ;低位
    mov cx ,0100h
    div cx  ;商在 ax,余数在 dx
code ends
end start
4_2
code segment
start:
    mov ax ,0f0fh
    cwd
    mov dx, 0f000h
    mov cx ,00ffh
    div cx
code ends
end start
```

2.4.3 实验结果

0100h/0100h 得到结果商为 1，余数为 0，ax 得到结果为 1，dx 得到结果为 0，0f0fh/ff00h 得到的结果商为 0，余数为 ff00h，故 ax 为 0，dx 为 ff00h。

2.5 实验五

2.5.1 实验简述

实现连续相加，对于不同数据都加 01h，有些可能加完会产生溢出，有些则不会产生溢出，如果溢出，则直接将溢出位舍弃。把最终结果存在 ax 或者 dx：ax 中。

2. 5. 2 实验源码

```
5_1
code segment
start:
    mov ah,0fh
    mov al ,01h
    mov cx,241
    next: add ah,al
    loop next
code ends
end start
5_2
code segment
start:
    mov ax,0ffh
    mov bx ,01h
    mov cx,65281
    next: add ax,bx
    loop next
code ends
end start
5_3
code segment
start:
    mov ax,0fffh
    mov bx ,01h
    mov cx,61441
    next: add ax,bx
    loop next
code ends
end start
5_4
code segment
start:
    mov ax,0ffffh
    mov bx ,01h
    mov cx,5
    next: add ax,bx
    loop next
code ends
end start
```

```
5_5
code segment
start:
    mov ax,0ffffh
    mov bx ,0ffffh
    mov cx,0000h
    mov dx,0001h
    add bx,dx
    adc ax,cx
    ;loop next
code ends
end start
```

2.5.3 实验结果

第一次相加得到结果为 10h,第二次相加得到结果为 100h,第三次相加得到结果为 1000h, 第四次相加得到结果为 0000h,且 CY=1，产生进位，第五次相加得到结果为 00000000，且 CY=1，产生进位。

2.6 实验六

2.6.1 实验简述

实现移位运算，将 8f1dh 存入 ax，使用 shr 和 shl 进行移位操作，重新进行赋值，再用 sal 和 sar 实现带符号位的移位操作。

2.6.2 实验源码

```
code segment
start:
    mov ax,8f1dh
    shr ax,1
    shl ax,1
    mov ax,8f1dh
    mov cl,5
    sal ax,cl
    mov cl,7
    sar ax,cl
code ends
end start
```

2.6.3 实验结果

指令右移 1 位，得到结果为 478E，产生溢出，OV=1,CY=1。然后左移 1 位，得到结果 8F1C，仍然 OV=1,PL=1,CY=0。将 8F1DH 存至 AX 中，然后带 CF

位左移 5 位，得到结果为 E3A0，产生溢出，OV=1,CY=1。右移 7 位，得到结果为 FFC7,未产生溢出，OV=0,CY=0。

2.7 实验七

2.7.1 实验简述

将 71d2h 保存在 ax 中，5df1h 保存在 bx 中，cx 保存移位的次数 2，用循环+shr+rcr 实现双精度右移，得到结果保存在 ax 中，再次赋值，并将 ax 与 bx 中的值互换，cx 保存移位的次数 4，用循环+shl+rcl 实现双精度左移，得到结果保存在 bx 中。

2.7.2 实验源码

```
code segment
start:
    mov ax, 71d2h
    mov bx, 5df1h
    mov cx, 2
    next1:
    shr ax, 1
    rcr bx,1
    loop next1
    mov cx, 4
    xchg ax,bx
    next2:
    shl ax, 1
    rcl bx,1
    loop next2
code ends
end start
```

2.7.3 实验结果

将 71D2H 存至 ax 中，5DF1H 存至 bx 中，DST 为 ax，REG 为 bx，实现双精度右移 2 次，得到结果为 5C74,OV=1。交换 DST 与 REG，然后左移 4 次，得到的结果为 DF17，OV=1,PL=1。

2.8 实验八

2.8.1 实验简述

实现压缩 BCD 码的加减法，用压缩 BCD 码实现（21+71），（12+49），（65+82），（46-33），（74-58），（43-54）的十进制加减法。使用 al,bl 进

行操作，并用压缩 BCD 调整指令进行调整，得到的第一个结果即为非压缩 BCD 码的结果，经过调整后寄存器数值为压缩 BCD 码结果。

2.8.2 实验源码

```
8_1_1
code segment
start:
    mov al,21h
    mov bl,71h
    add al,bl
    daa
code ends
end start
8_1_2
code segment
start:
    mov al,12h
    mov bl,49h
    add al,bl
    daa
code ends
end start
8_1_3
code segment
start:
    mov ax,65h
    mov bx,82h
    add ax,bx
    add ax, 60h
code ends
end start
8_1_4
code segment
start:
    mov al,46h
    mov bl,33h
    sub al, bl
    das
code ends
end start
8_1_5
code segment
```

```
start:
    mov al,74h
    mov bl,58h
    sub al, bl
    das
code ends
end start
8_1_6
code segment
start:
    mov al,43h
    mov bl,54h
    sub al, bl
    neg al
code ends
end start
8_2_1
code segment
start:
    mov ax, 0201h
    mov bx, 0701h
    add ax, bx
    aaa
code ends
end start
8_2_2
code segment
start:
    mov ax, 0102h
    mov bx, 0409h
    add ax, bx
    aaa
code ends
end start
8_2_3
code segment
start:
    mov bx, 0605h
    mov dx, 0802h
    add bx, dx
    mov al, bh
    cbw
```

```
    aaa ;ax:bl 010407
code ends
end start
8_2_4
code segment
start:
    mov ax, 0406h
    mov bx, 0303h
    sub ax, bx
    aas
code ends
end start
8_2_5
code segment
start:
    mov ax, 0704h
    mov bx, 0508h
    sub al, bl
    aas
    sub ah, bh
    aas
code ends
end start
8_2_6
code segment
start:
    mov ax, 0403h
    mov bx, 0504h
    sub ax, bx
    neg ax ;求补
code ends
end start
```

2.8.3 实验结果

得到的压缩 BCD 结果分别为 92、5B、E7、13、1C、EF，调整后得到的压缩 BCD 码为 92、61、47、13、16、83，并在 E7 调整为 47 时有 CY=1，即进位。

2.9 实验九

2.9.1 实验简述

实现 KMP 算法,将需要匹配的两个字符串置于数据段并用$结尾，并设置一个 next 数组，先求得子串的 next 数组，即第 j 个元素前 j-1 个元素首尾重合部分个

数加一。再依次比较两个字符串中的字符，当不相同时，S 串的索引 i 不动，P 串的索引 j 定位到某个数。T(n)=O(n+m)。若索引值 j 大于等于子串长度，说明匹配成功，用此时的 i 减去子串长度得到匹配时对应的字符所在位置。

2.9.2 实验源码

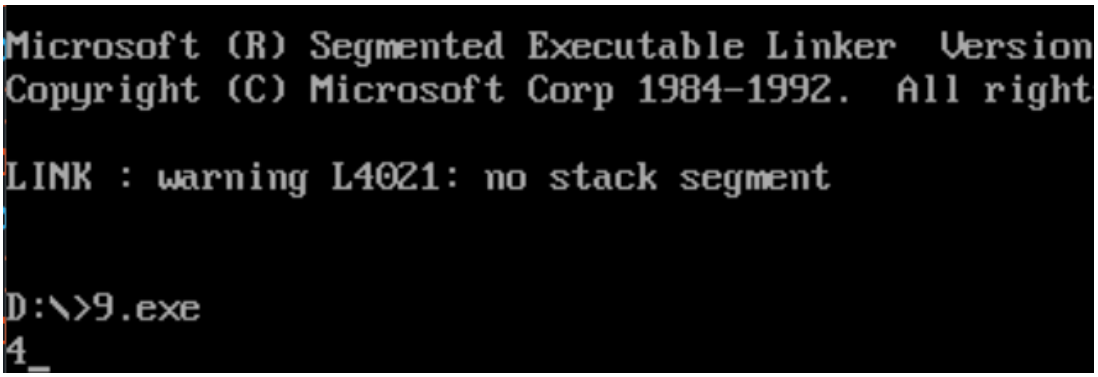
```
data segment
    string1 db 'abcdabcdefghi$'
    string2 db 'abcde$'
    array db  100 dup(?);定义 100 个字节的数组
data ends
code segment
main proc
    mov ax,data
    mov ds,ax
    mov cx,5  ;CX 存放子串数目
    mov al,1 ;Q 主串索引
    mov bl,0 ;K 子串索引
next:    ;跳到 NEXT[Q]=K，先求 next 数组
    mov ah,0
    mov si,offset string2;偏移地址，便于存储数组
    add si,ax
    mov ah,[si]  ;P[Q]，依次保存串
    mov bh,0
    mov di,offset string2
    add di,bx
    mov bh,[di]  ;P[K]
    cmp ah,bh;对比计数，确定 next 的值
    je DENG;跳跃
    cmp bl,0
    je then
    mov bh,0
    mov di,offset array
    add di,bx
    dec di
    mov bl,[di]  ;NEXT[K-1]
    jnz then
DENG:
    inc bl;继续存下一个
    dec cx;
    jmp then
then:
```

```
    mov ah,0
    mov di,offset array;引入数组保存
    add di,ax ;next Q 实际地址
    mov [di],bl ;next[Q]=K，保存上述求的结果
    inc al;不断往后索引
    dec cx;减少
    cmp cx,0;若相等则进入 kmp_before
    je kmp_before
    jmp next
kmp_before:
    mov al,0 ;i 指针
    mov bl,0 ;j 指针
    mov di,offset array
    add di,4 ;next Q 实际地址
    mov [di],al ;next[Q]=K
    inc al
    mov [di],al
    dec al
    jmp kmp
kmp:
    cmp al,13;主串长度为 13，可以任意更改，如果 i 索引到最后，则结束
    je over
    cmp bl,5;j 索引长度大于子串长度，即找到
    je OK;输出结果
    mov ah,0
    mov di,offset string1
    add di,ax  ;S[i]
    mov ah,[di];保存当前指针 i 所指
    mov bh,0
    mov si,offset string2  ;T[j]
    add si,bx
    mov bh,[si];保存当前指针 j 所指
    cmp ah,bh;依次比较
    je next_one
    mov bh,0
    mov si,offset array  ;NEXT[j]，不相同，则往后走
    add si,bx
    mov bl,[si]
    jmp kmp
next_one:
    inc al;往后
    inc bl
```

```
    jmp kmp
OK:
    sub al,5;将 j 指针所指的地方回溯 5
    add al,48
    mov ah,02h;显示输出
    mov dl,al;要读出的内容
    int 21h
    ret
over:
    ret
main endp
code ends
end main
```

2.9.3 实验结果

进行 KMP 算法实现字符串匹配后得到匹配位置为 4，如下图所示。



2.10 实验十

2.10.1 实验简述

使用汇编实现斐波纳契数列，展示形式为输入数字 n,然后输出第 n 个斐波那契数，程序首先调用 1 号功能进行输入，然后判断输入的数字是否小于等于 2，若是则直接输出 1，若不是则调用递归 fact，然后将最终结果存入 ax 中，最后调用输出将结果展示在 dos 窗口。

2.10.2 实验源码

```
.MODEL SMALL
.DATA
STRING1 DB 'INPUT:$'    ;输入
STRING2 DB 'OUTPUT:$'   ;输出
.CODE
main proc far
    mov ax,@data;        为数据段开辟地址
```

```
    mov ds,ax
    lea dx,STRING1;      输出第一句话
    mov ah,09h
    int 21h
    call input;      读入数，了解使用者需要知道第几个斐波那契数
    lea dx,STRING2;      输出第二句话
    mov ah,09h
    int 21h
    mov cx,bx
    dec cx
    dec cx;              进行两次减减，方便分输入数是否大于 2 进行不
同的运算输出
    cmp cx,0
    jle shuchu1;             若输入的数小于 2（进行两次减减小于 0）
则直接输出 1 结束程序
    mov ax,1;
    mov bx,1
    push ax
    push dx;
    push dx;             此处连续压入两个 dx 仅仅为了方便与后续情况
保持相同规律
    push bx
    call fact;           若输入的数大于 2（进行两次减减大于 0）则进
行 fact 运算
    call output;
shuchu1:
    mov al,31h;          直接输出 1 结束程序
    mov dl,al
    mov ah,02h
    int 21h
    MOV AX,4C00H;终止程序
    INT 21H
main endp
;-------------------------------
; 开始读入一个数
input proc near
    mov bx,0
newchar:
    mov ah,1 ;输入指令，存入 AL
    int 21h
    sub al,30h;          如果小于 0
    jl exit  ;小于转移
```

```
    cmp al,9;              如果大于 9
    jg exit ;大于退出
    cbw;                   扩展，al->ax
    ; xchg ax,bx;
    ; mov cx,10
    ; mul cx
    ; xchg ax,bx
    ; add bx,ax;               最后的结果放入 bx
    mov bx,ax
    jmp newchar
exit:
    ret
input endp
;--------------------
;开始计算斐波那契数列
fact proc near
    push bp ;保存 bp 指针
    mov bp,sp ;将 sp 指针传给 bp，此时 bp 指向 sp 的基地址。
    mov ax,[bp+4];         取上个数
    mov bx,[bp+10];      取上上个数
    add ax,bx
    push ax
    dec cx;                此处 cx 为前面输入的标记，控制输出
    cmp cx,0
    je fact1
    call fact
fact1:
    pop ax;                出栈所寻找的数
fact endp
;--------------------------
;开始输出结果
output proc near
OUTPUT5:
    MOV DX,0;
    MOV BX,10
    DIV BX;                ax 除以 10 余数放入 dx 中
    ADD DX,30H
    PUSH DX ;                  将 dx（个位、十位、百位、等）压入栈
    INC SI;                做出栈标记
    CWD;                       扩展
    CMP AX,0
    JZ OUTPUT4         ;Z 标志为 0 跳转
```

```
    JMP OUTPUT5
OUTPUT4:
    POP AX;              逐个出栈（。。。。百位、十位、个位）
    DEC SI
    MOV DL,AL;           逐个输出数的。。。。百位、十位、个位
    MOV AH,02H
    INT 21H
    CMP SI,0
    JNZ OUTPUT4     ;Z 标志不为 0 跳转
output endp
ENDING:
    MOV AX,4C00H;        终止程序
    INT 21H
END
```

2.10.3 实验结果

```
LINK : warning L4021: no stack segment
LINK : warning L4038: program has no starting address


D:\>10.exe
INPUT:9
OUTPUT:34
(END)Here is the end of the program's output
```

得到最终结果多次除 10 得到单个字符，然后调用 dos2 号功能逐个输出，即在屏幕上展示最终结果。

第 3 章　实验心得

本次实验是在 8086 环境下使用汇编语言编写程序，由于时间太紧张并且汇编知识比较难懂，所以刚开始的编写过程是比较艰难的，出错有时找不到合适的解决方案，不同的编译器得到的结果也不相同。在进行实验汇报时，老师问了我几个问题，但是我回答的都不是太好，可能是因为太紧张了，或者是知识点确实掌握的不太好。同时我也在实验中发现了自己的不足之处，在进行代码编写的过程中，并没有深究其中的原理，而只是想着去如何实现，或者这个寄存器怎么用，导致我很多问题都似懂非懂，我之后会尽量改正的。

不过我也有许多收获，我更加深入地了解了计算机组成及内部程序执行的原理。观察每一条命令对计算机变化，也让我知道了各命令的用途和对计算机的影响。同时，还有对于各寻址方式的了解，要正确书写各个值，尤其要注意细节，如数据传送时要注意目的操作数和源操作数的类型匹配，长度不同时要进行转化。对于分支结构，对于跳转指令要时刻清楚自己的思路，防止编写过程中出现逻辑混乱。我认为汇编语言入门是一件比较困难的事情，因为他与我们平时学习到的高级语言的侧重点有些差别，汇编语言需要我们更加具有计算机的思维，要更加考虑代码对于计算机的影响，不同于我们以往所学的高级语言，更偏重于算法本身空间和计算的时间影响，并不考虑对计算机本身的影响。并且汇编语言有很多细小且不可更改的指令规定，要按照严格的规则编写，本身无法使用函数模块，比起高级语言更加繁琐。不过学透了汇编会使我们真正理解计算机，还有很多问题是高级语言解决不了的，在分析问题等方面，汇编语句能帮我们大忙，这也说明了学习汇编语言的重要性。

本次实验结束，虽然不是完全掌握汇编的全部知识，不过在此过程中我还是学到了一些基本的操作，后期想要深入了解其中的知识，必然还需要投入时间和精力进行学习，不断提升个人能力。