

# 华中科技大学

## 课程实验报告

课程名称： 汇编语言程序设计实验

实验时间： 2017-5-20, 19:00-21:30      实验地点： 南一楼 804 室 08 号实验台

指导教师： 张 勇

专业班级： 七校联合二学位 计算机科学与技术 201502 班

学 号： U201516431      姓 名： 刘云中

同组学生： 无      报告日期： 2017 年 5 月 23 日

### 原创性声明

本人郑重声明：本报告的内容由本人独立完成，有关观点、方法、数据和文献等的引用已经在文中指出。除文中已经注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品或成果，不存在剽窃、抄袭行为。

特此声明！

学生签名：

日期：2017.5.14

### 成绩评定

实验完成质量得分(70分) (实验步骤清晰详细深入,实验记录真实完整等)	报告撰写质量得分(30分) (报告规范、完整、通顺、详实等)	总成绩(100分)

指导教师签字：

日期：

# 目录

1	实验目的与要求.....	1
2	实验内容.....	1
3	实验过程.....	2
3.1	任务 3 .....	2
3.1.1	设计思想及存储单元分配 .....	2
3.1.2	流程图 .....	3
3.1.3	源程序 .....	8
3.1.4	实验步骤 .....	14
3.1.5	实验记录与分析 .....	15
4	总结与体会.....	18
4.1	对实验所涉及知识点的主要认识与收获 .....	18
4.1.1	如何编写一个有一定规模的汇编程序 .....	18
4.2	经验教训与发现的问题 .....	19
4.2.1	字符串输入功能（INT 10H）的缓冲区要求 .....	19
	参考文献 .....	21

# 汇编语言程序设计实验报告

---

## 1 实验目的与要求

本次实验的主要目的与要求有下面 6 点，所有的任务都会围绕这 6 点进行，希望大家事后检查自己是否达到这些目的与要求。

- (1) 掌握汇编源程序编辑工具、汇编程序、连接程序、调试工具 TD 的使用；
- (2) 理解数、符号、寻址方式等在计算机内的表现形式；
- (3) 理解指令执行与标志位改变之间的关系；
- (4) 熟悉常用的 DOS 功能调用；
- (5) 熟悉分支、循环程序的结构及控制方法，掌握分支、循环程序的调试方法；
- (6) 加深对转移指令及一些常用的汇编指令的理解。

## 2 实验内容

任务 3：设计实现一个学生成绩查询的程序。

### 1、实验背景

在以 BUF 为首址的字节数据存储区中，存放着 n 个学生的课程成绩表（百分制），每个学生的相关信息包括：姓名（占 10 个字节，结束符为数值 0），语文成绩（1 个字节），数学成绩（1 个字节），英语成绩（1 个字节），平均成绩（1 个字节）。

例如：

```
N    EQU    30
BUF  DB    'zhangsan',0,0    ;学生姓名，不足 10 个字节的部分用 0 填充
DB    100, 85, 80, ?        ; 平均成绩还未计算
      DB    'lisi',6 DUP(0)
DB    80, 100, 70, ?
DB    N-3 DUP( 'TempValue',0,80,90,95,?) ;除了 3 个已经具体定义了学生信息的成绩表以外，其他学生的信息暂时假定为一样的。
DB    'wangwu', 0, 0, 0, 0; 最后一个必须是自己名字的拼音
DB    85, 85, 100, ?
```

### 2、功能一：提示并输入待查询成绩的学生姓名

(1) 使用 9 号 DOS 系统功能调用，提示用户输入学生姓名。

(2) 使用 10 号 DOS 系统功能调用，输入学生姓名。输入的姓名字符串放在以 in\_name 为首址的存储区中。

# 汇编语言程序设计实验报告

---

(3) 若只是输入了回车,则回到“(1)”处重新提示与输入;若仅仅输入字符q,则程序退出,否则,准备进入下一步处理。

## 3、功能二:以学生姓名查询有无该学生

(1) 使用循环程序结构,在成绩表中查找该学生。

(2) 若未找到,就提示用户该学生不存在,并回到“功能一(1)”的位置,提示并重新输入姓名。

(3) 若找到,则将该学生课程成绩表的起始偏移地址保存到POIN字变量中。

提示:字符串比较时,当采用输入串的长度作为循环次数时,若因循环次数减为0而终止循环,则还要去判断成绩表中名字串的下一个字符是否是结束符0,若是,才能确定找到了(这样做是为了避免输入的名字仅仅是数据段中所定义名字的子集的误判情况)。

## 4、功能三:计算所有学生的平均成绩

使用算数运算相关指令计算并保存每一个学生的平均成绩。

平均成绩计算公式:  $(A*2+B+C/2)/3.5$ ,即将语文成绩A乘以权重2、英语成绩C除以权重2后,与数学成绩B一起求和,再计算该生的平均成绩。要求避免溢出。

提示: 整个功能三使用循环程序结构,但其中的平均成绩计算的过程采用子程序实现。注意寻址方式的灵活使用。把小数3.5转换成分数后再运算避免使用浮点数指令。

## 5、功能四:将功能二查到的学生的平均成绩进行等级判断,并显示判断结果。

(1) 平均成绩等级显示方式:若平均成绩大于等于90分,显示“A”;大于等于80分,显示“B”;大于等于70分,显示“C”;大于等于60分,显示“D”;小于60分,显示“F”。

提示:使用分支程序结构,采用2号DOS系统功能调用显示结果。

(2) 使用转移指令回到“功能一(1)”处(提示并输入姓名)。

## 3 实验过程

### 3.1 任务3

#### 3.1.1 设计思想及存储单元分配

##### (一) 设计思想

在功能设计上,本程序是一个交互式命令程序,由用户输入与程序的提示组成。程序会在屏幕

# 汇编语言程序设计实验报告

上给予必要而充分的提示，并即时对用户的输入进行解析、判定，给予用户提示。交互式命令的设计有利于在只能显示文字的运行环境里保证用户体验，因此这类程序的体验性要明显强于只接受传入命令行参数的程序。

在设计方法上，本程序采用的是类模块化设计方法。即借鉴模块化程序设计思想，相对独立地开发程序的主要功能，并将各个功能有机组合在一起，形成完整的程序。（但是这里的“模块化”仅仅指功能开发过程中的相对独立，并不是真正意义上的模块化开发，因为本程序所有功能都写在同一个代码段中。）

## （二）存储单元分配

表格 3.1 1 存储单元分配表

变量名	类型	作用
BUF	DB	存放学生成绩信息，总大小为 14×30 字节。 每个学生的资料占 14 字节，其中包括： 一、学生名字，占 10 个字节，不足 10 个字节的用 0 填充； 二、四个分数，每个分数占 1 个字节。 整个 BUF 变量最多存储 30 个学生的资料。
BUF_END	DB	标志变量，紧接 BUF 定义，用以标志 BUF 的结束。 使用该变量，可以在程序中便捷地定位到 BUF 的末端。
STR_*	DB	一系列字符串变量，存放屏幕提示信息。
CRLF	DB	字符串变量，存放回车换行符。
IN_NAME	DB	接收用户输入的学生名字，有 3 个域。 第一个域，占用 1 个字节，存放允许的最大字符串长度； 第二个域，占用 1 个字节，存放用户实际输入的字符串长度； 第三个域为数据域，存放用户输入的字符串，长度同第一个域的值。
POIN	DW	存放检索的目标学生记录的首地址。

### 3.1.2 流程图

四个核心功能的流程图如下：

# 汇编语言程序设计实验报告

---

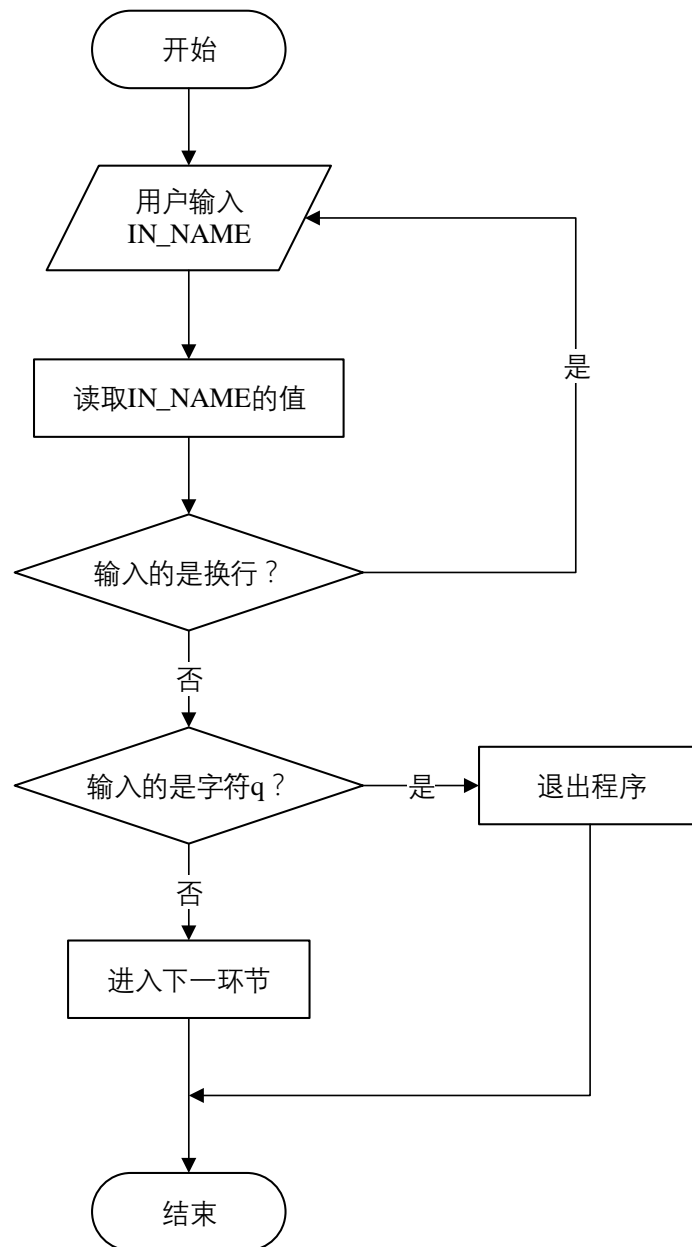


图 3.1 1 功能一（判定用户输入）流程图

# 汇编语言程序设计实验报告

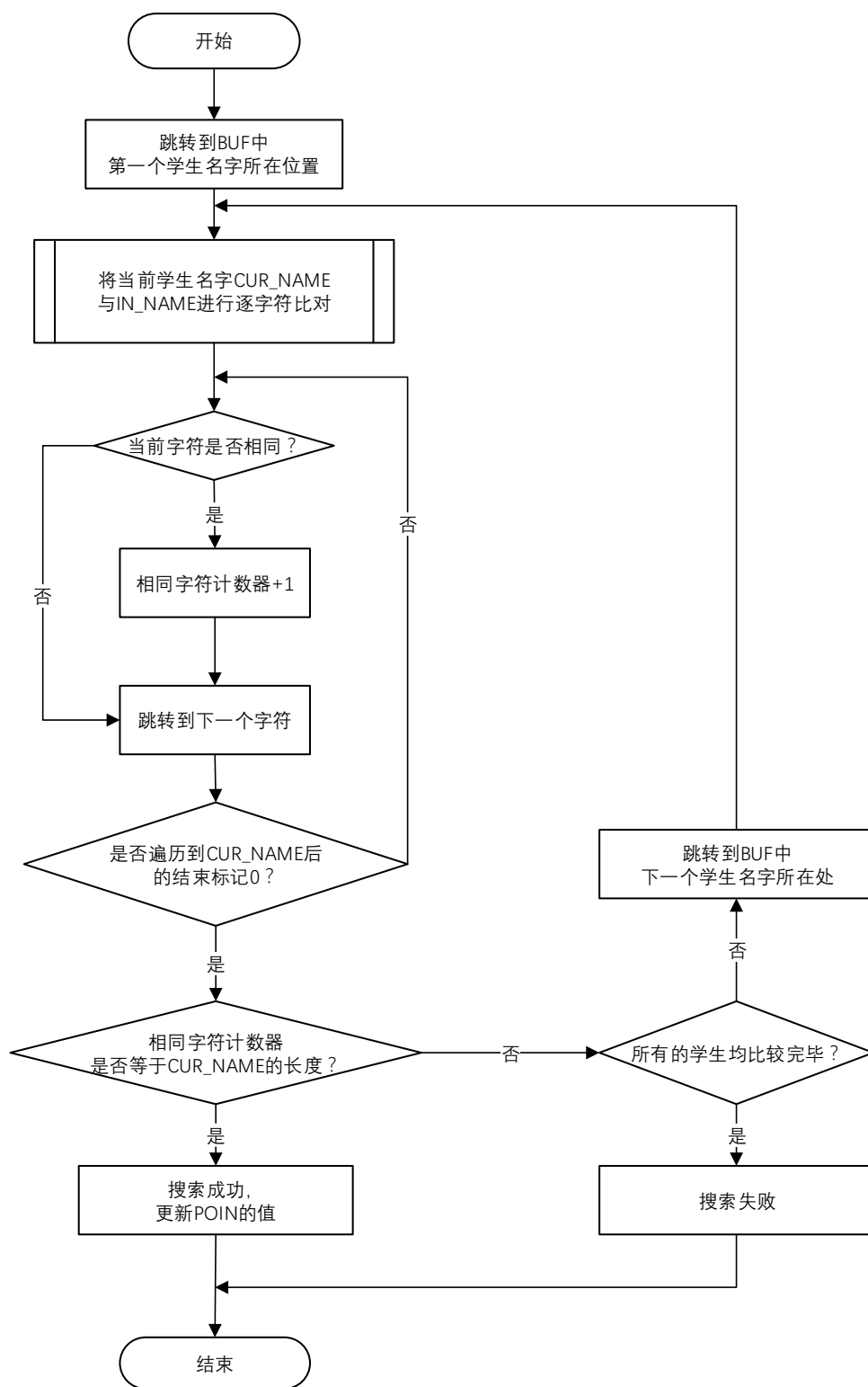


图 3.1.2 功能二（以姓名查询学生）流程图

# 汇编语言程序设计实验报告

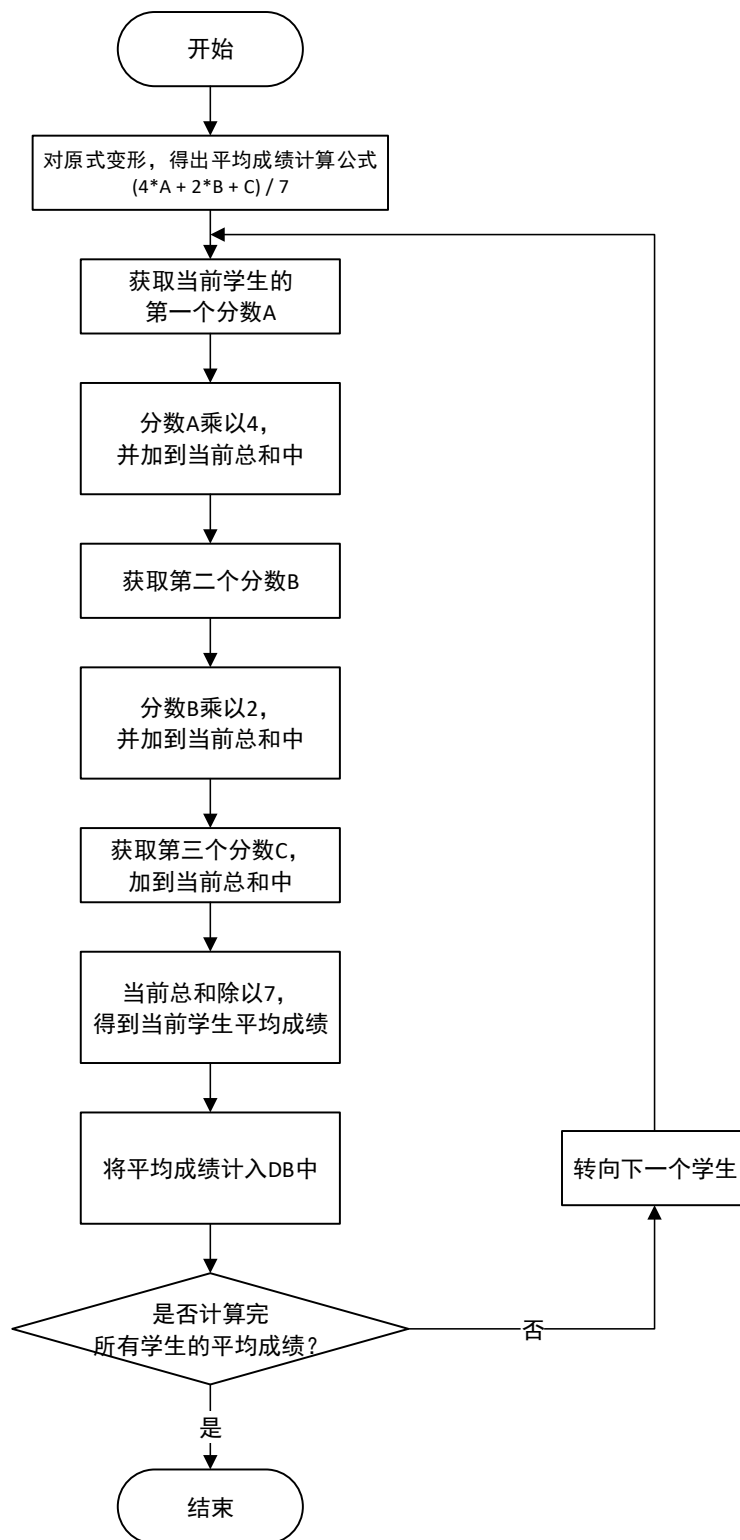


图 3.13 功能三（计算平均成绩）流程图



# 汇编语言程序设计实验报告

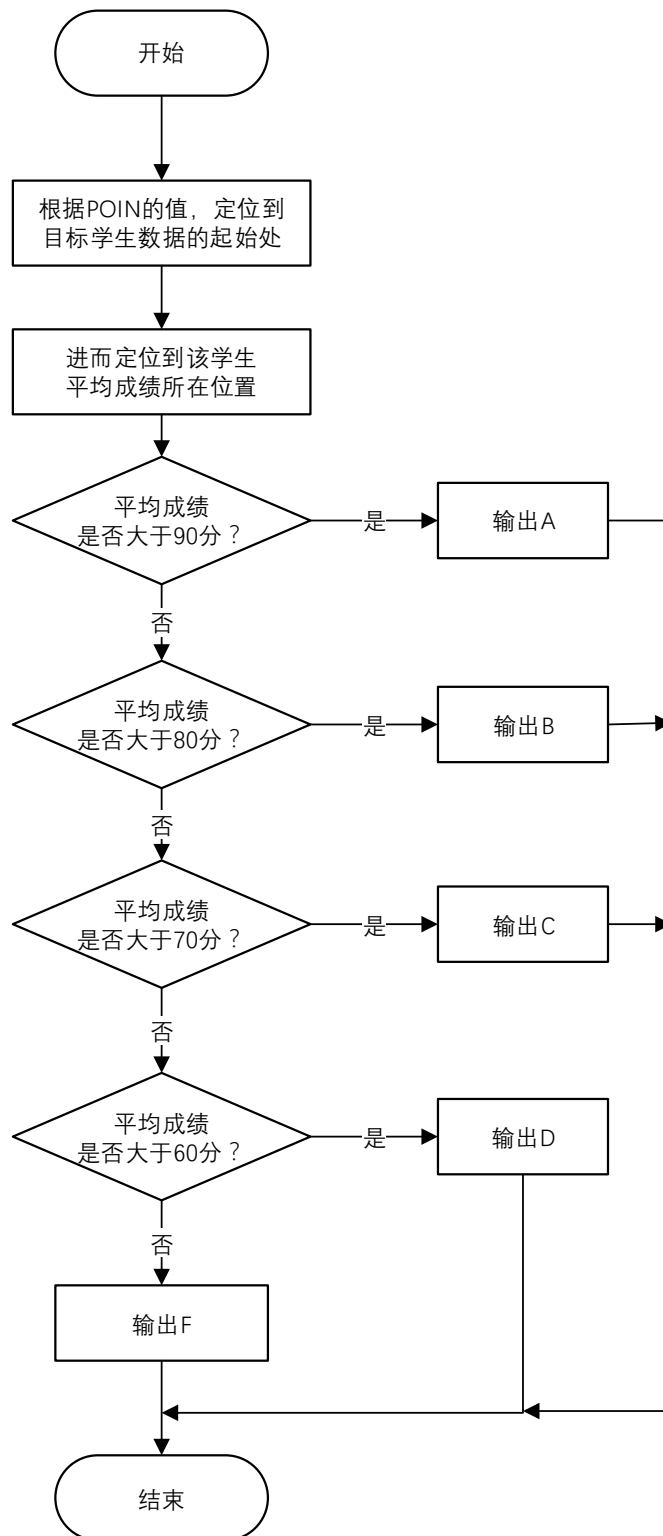


图 3.14 功能四（判定学生平均成绩等级）流程图

# 汇编语言程序设计实验报告

---

## 3.1.3 源程序

```
; Task 3: Write a simple scholar grade querier.

; ----- MACRO DEFINITIONS -----
; Screen output: print screen info
PRINT_MACRO MACRO A
    LEA DX, A
    MOV AH, 9
    INT 21H
ENDM

; Screen output: Go to a new line
PRINT_CRLF_MACRO MACRO
    LEA DX, CRLF
    MOV AH, 9
    INT 21H
ENDM

STACK SEGMENT USE16 STACK
    DB 512 DUP(0)
STACK ENDS

DATA SEGMENT USE16
; ===== Student Data =====
N EQU 30
BUF DB 'Satoshi', 0, 0, 0
    DB 100, 85, 80, ?          ; AVE = 92 (A)
    DB 'Kasumi', 0, 0, 0, 0
    DB 80, 100, 70, ?          ; AVE = 84 (B)
    DB 'AsukaRyo', 0, 0
    DB 77, 83, 60, ?           ; AVE = 76 (C)
    DB 'Kagome', 0, 0, 0, 0
    DB 62, 87, 70, ?           ; AVE = 70 (C)
    DB 'EdSheeran', 0
    DB 60, 80, 70, ?           ; AVE = 67 (D)
    DB 'NoisyGuy', 0, 0
    DB 55, 62, 33, ?           ; AVE = 53 (F)

    DB N-7 DUP('TEMP_VALUE', 60, 60, 60, ?) ; AVE = 60 (D)

    DB 'AkuraRyu', 0           ; 这是我名字的日文拼读
    DB 95, 95, 100, ?          ; AVE = 95 (A)

; BUF's ending flag, defined in order to mark the end of BUF CORRECTLY.
; NOTICE: This is essential for compatibility among different machines!
BUF_END DB 0
```

# 汇编语言程序设计实验报告

---

```
; ===== Screen Info =====
STR_TITLE DB '----- GRADE QUERY ----- $'
STR_INPUT_NAME DB 'Please input student name: $'
STR_NOT_FOUND DB ODH, OAH, 'Student not found! $'
STR_FOUND DB ODH, OAH, 'Student FOUND! $'
STR_QUIT DB ODH, OAH, ODH, OAH, 'Quitting program...', ODH, OAH, '$'
STR_PRINT_GRADE DB ODH, OAH, 'Grade is: $'
CRLF DB ODH, OAH, '$'

; ===== Vars for query =====
IN_NAME DB 11
        DB ?
        DB 11 DUP(0)
POIN DW ?

DATA ENDS

CODE SEGMENT USE16
    ASSUME CS:CODE, DS:DATA, SS:STACK
START:
; ===== INITIALIZE =====
; Initialize data segment
MOV AX, DATA
MOV DS, AX
;=====

; Show program title
PRINT STR_TITLE

PROMPT:
PRINT_CRLF
PRINT_CRLF

; Prompt for user input
PRINT STR_INPUT_NAME

; Receive user input
LEA DX, IN_NAME
MOV AH, 10
INT 21H

; Append dollar mark to the end of string
; NOTICE: If you input nothing, the string will only contain '$'.
MOV BL, IN_NAME+1          ; Get actual length of input
MOV BH, 0                  ; Clear BH
MOV BYTE PTR IN_NAME+2[BX], '$' ; Locate to the end of string, then append

; ===== FUNCTION 1: Check user input =====
```

---

# 汇编语言程序设计实验报告

---

```
; First: get value of your input
LEA SI, IN_NAME+2
MOV BL, [SI]

; SITUATION 1: Just input an CRLF, aka. input nothing
; At this time, the buffer only contains a '$'.
CMP BL, '$'
JE PROMPT

; SITUATION 2: Just input 'q' for quit.
; TODO: In future, using 'JE' may raise an exception!
CMP BL, 'q'
JE QUIT

; SITUATION 3: Default --- input a valid student name
; Just continue!

; ===== FUNCTION 3: Calculate all students' average =====
CALL CALCULATE_AVERAGE

; ===== FUNCTION 2: Find student =====
; METHOD: Check character one by one, and analyze how many chars are same.

; Get heading addr of BUF.
; In this segment, POIN points to current position. It will stop on target (if found) or ending
(if not found).
MOV POIN, OFFSET BUF

; -----
LOOP_FIND_STUDENT:
; Set cursor to strings we compare
MOV SI, POIN          ; Student name in BUF
LEA DI, IN_NAME+2     ; Our input

; Use DL to store the number of same chars, DH stores the length of STR1.
XOR DX, DX            ; Clear DX first
XOR BX, BX            ; BX plays role of buffer

LOOP_CHECK_SAME_STUDENT_NAME:: Count how many characters are same.
; Get current characters
MOV BL, [SI]          ; Char in student name in BUF
MOV BH, [DI]          ; Char in our input
INC DH                ; Calculate length of STR1

; Compare if same
CMP BH, BL
JE SAME_CHAR
JMP NOT_SAME_CHAR

SAME_CHAR:
```

---

# 汇编语言程序设计实验报告

---

```
INC DL

NOT_SAME_CHAR:
; Point to the next char
INC SI
INC DI

; --- LOOPBACK JUDGEMENT
CMP BYTE PTR [SI], 0 ; Check if we arrived the end flag
JNE LOOP_CHECK_SAME_STUDENT_NAME

; Check if found by checking if DH=DL.
; If found: Give a tip, then TODO: continue.
CMP DH, DL
JE FOUND
JMP TRY_NEXT_STUDENT

TRY_NEXT_STUDENT:
; If not found this time: Try next student
ADD POIN, 14 ; Cursor to the next student

; --- LOOPBACK JUDGEMENT
LEA DI, BUF_END
CMP POIN, DI ; Check if we arrived the end of BUF

JL LOOP_FIND_STUDENT
; -----

; FINAL CHECK: The worth situation is that we traversed all those students, but still not found.
LEA DI, BUF_END
CMP POIN, DI

JGE NOT_FOUND
JMP FOUND

NOT_FOUND:
PRINT STR_NOT_FOUND
JMP PROMPT ; Return to prompt

FOUND:
PRINT STR_FOUND

; When found, go to the next stage - getting grade!
JMP GET_GRADE

; ===== FUNCTION 4: Get grade =====
GET_GRADE:
; --- DEBUG: This is a mark for locating code in TD.
XOR DX, DX
XOR DX, DX
```

---

# 汇编语言程序设计实验报告

---

```
XOR DX, DX

; Locate to the chosen student's average
LEA SI, BUF          ; Get BUF's base address
ADD SI, POIN         ; Forward to the target student
ADD SI, 13           ; Forward to the student's average

; Print screen tip
PRINT STR_PRINT_GRADE

; Now analyse and give a result
CMP BYTE PTR [SI], 90
JGE LE_90

CMP BYTE PTR [SI], 80
JGE LE_80

CMP BYTE PTR [SI], 70
JGE LE_70

CMP BYTE PTR [SI], 60
JGE LE_60
JNGE NLE_60

LE_90:
MOV DX, 'A'
MOV AH, 2
INT 21H
JMP END_GET_GRADE
LE_80:
MOV DX, 'B'
MOV AH, 2
INT 21H
JMP END_GET_GRADE
LE_70:
MOV DX, 'C'
MOV AH, 2
INT 21H
JMP END_GET_GRADE
LE_60:
MOV DX, 'D'
MOV AH, 2
INT 21H
JMP END_GET_GRADE
NLE_60:
MOV DX, 'F'
MOV AH, 2
INT 21H
JMP END_GET_GRADE

END_GET_GRADE:
```

# 汇编语言程序设计实验报告

---

```
JMP PROMPT

QUIT:
; Print quit tip
PRINT STR_QUIT

; Exit program
MOV AH, 4CH
INT 21H

CALCULATE_AVERAGE PROC NEAR
; Calculate all students' average
; FORMULA:
;     - Source:  $(A*2+B+C)/2$  / 3.5
;     - Exported:  $(4*A + 2*B + C) / 7$ 
; REGISTERS:
;     - AX: Occupied by MUL & IMUL
;     - BX: Contains number to multiply/divide with
;     - CX: Current result of calculation of current student
;
; STORAGE:
;     - POIN

; NOTICE: I don't keep registers here, or my program will behave unexpectedly.

; Get ready
; Get the heading address FOR MARKS of BUF
LEA SI, BUF
ADD SI, 10

LOOP_FIND_STUDENT_MARKS:
; Clear registers
XOR AX, AX
XOR BX, BX
XOR CX, CX
XOR DX, DX

; NOTICE!
; How much indirect addr picker picks are depended on the first argument!
; Here, each mark takes only 1 Byte. If we use AX instead of AL when getting values of A,
; B, C,
; 2 Bytes of data will be read. The only consequence is that we get a totally wrong
; result.

; Read, calculate mark A, then accumulate
MOV AL, [SI] ; Get A
MOV BX, 4 ; Set multiplier 4
MUL BX ; 4*A
ADD CX, AX ; Accumulate
```

---

# 汇编语言程序设计实验报告

---

```
; Read and calculate mark B, then accumulate
XOR AH, AH          ; Clear AH to prevent pollution
MOV AL, [SI]+1       ; Get B
MOV BX, 2            ; Set multiplier 2
MUL BX               ; 2*B
ADD CX, AX           ; Accumulate

; Read and calculate mark C, then accumulate
XOR AH, AH          ; Clear AH to prevent pollution
MOV AL, [SI]+2       ; Get C
ADD CX, AX           ; Accumulate

; Now let's give it a final division!
MOV AX, CX           ; AX stores the number for divide
MOV BX, 7            ; Set divider 7
DIV BX               ; Divide 7
MOV CX, AX           ; Get result back

; Everything is done. Put result to where it should be.
MOV [SI]+3, CL

; Cursor to the next student
ADD SI, 14

; --- LOOPBACK JUDGEMENT
LEA DI, BUF_END
CMP SI, DI
JL LOOP_FIND_STUDENT_MARKS

RET
CALCULATE_AVERAGE ENDP

CODE ENDS
END START
```

## 3.1.4 实验步骤

1. 仔细阅读实验题目文档，熟知题目中对成绩查询程序的各项要求。在之后编写代码的过程中，保持文档打开，以备随时查阅。
2. 先写好汇编程序的基本框架。基本框架主要包括三个段定义（SS、DS、CS）、代码段中的初始语句（START:标识符、给数据段寄存器 DS 赋值的语句）。
3. 根据题目文档，编写数据段的内容，主要包括 BUF 变量、IN\_NAME 变量、屏幕提示文本（以“STR\_”开头的一类字符串），以及实际运行中为保证功能正确实现而应具备的其他各种变量（如 BUF\_END）。

这里的 BUF\_END 紧接着 BUF 来定义，值为空，用以标志 BUF 变量的结束。运用这个小技巧，就



# 汇编语言程序设计实验报告

可以快速定位 BUF 变量的尾部，而无需经过其他运算。

为了测试程序的功能，须在 BUF 变量中安排一些测试性的学生成绩记录。至少要保证每个记录具有唯一性，同时保证所安排的记录中，平均成绩在各个分数档次内均有分布。

4. 根据题目文档，开始编写代码段内容。由于文档中明确地规定了程序的四个主要功能，因此四个功能分别编写，然后再组合到一起。

值得注意的是，每个功能都相当于汇编语言的一道“应用题”，因此为了更好地“解题”，构思功能的实现，有必要先把这些功能单独拿出来，作为独立的程序进行测试，成功后再写入本实验的总代码中。

5. 将写好的功能组合到实验的总代码当中，并进行调试。

6. 在学习了宏的使用后，有必要将一些复用程度高的代码整理成宏，从而减少代码的复杂度，并降低出错率。如最常用的打印输出字符串指令，在这样一个交互式程序中会经常用到。若每次都输入那三行代码，必然会使程序繁复冗长；同时在大型的汇编程序中，指令一旦输错，程序无法正常运行，就很难找出错误的代码所在。

## 3.1.5 实验记录与分析

1. 实践证明，在编写功能代码的过程当中，确有必要用单独的程序来“打草稿”，便于构思程序。如果直接在总代码中进行测试，则很容易与代码的其他部分混淆，且总代码中复杂的变量关系与输入输出关系会给功能的单独调试添加障碍。

2. 为了更好地调试程序，我准备了以下学生资料，均写入了我定义的 BUF 变量里：

表格 3.12 BUF 中的备测试数据

姓名	科目 A 成绩	科目 B 成绩	科目 C 成绩	平均成绩	理论等级
Satoshi	100	85	80	92	A
Kasumi	80	100	70	84	B
AsukaRyo	77	83	60	76	C
Kagome	62	87	70	70	C
EdSheeran	60	80	70	67	D
NoisyGuy	55	62	33	53	F
AkuraRyuu	95	95	100	95	A

3. 程序运行效果如下所示。

首先是程序初运行时的画面：

## 汇编语言程序设计实验报告

---

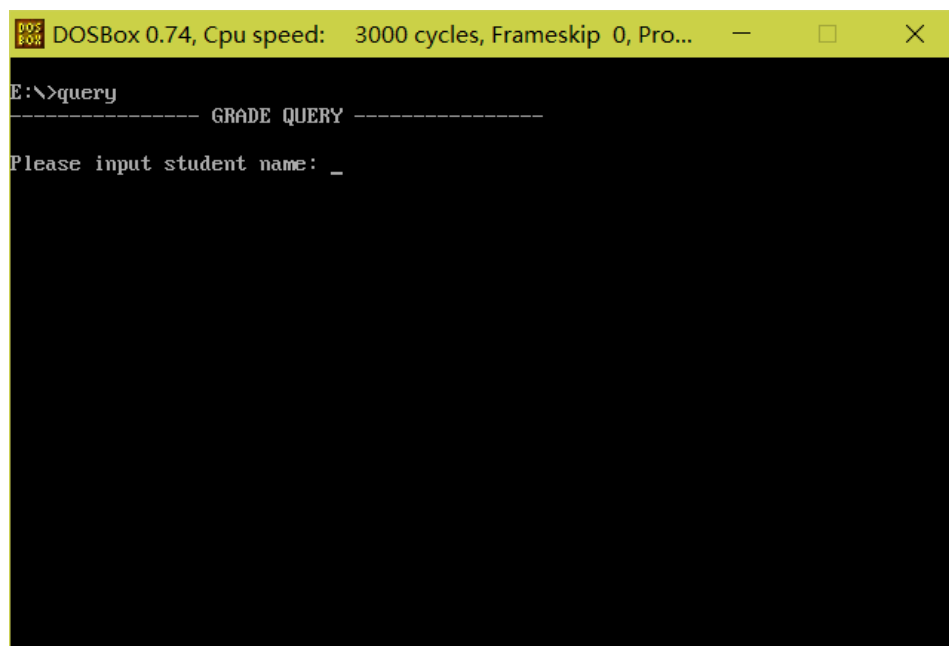


图 3.15 程序初运行画面

未输入任何字符而直接回车时，程序的提示符可重新弹出，让用户重输；输入字符“q”，可退出程序，并给出退出程序的提示。

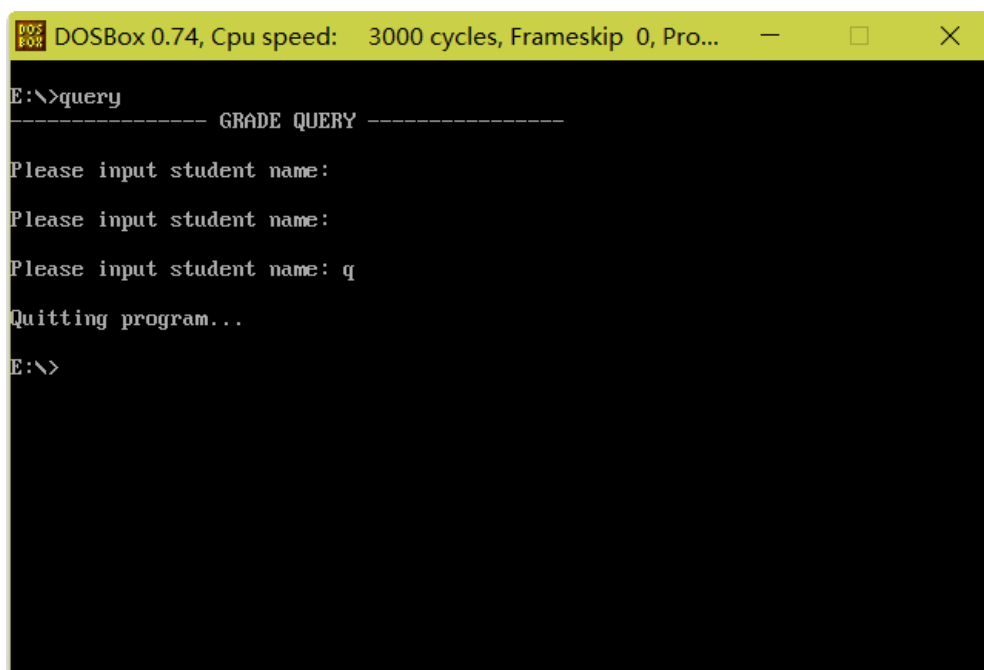
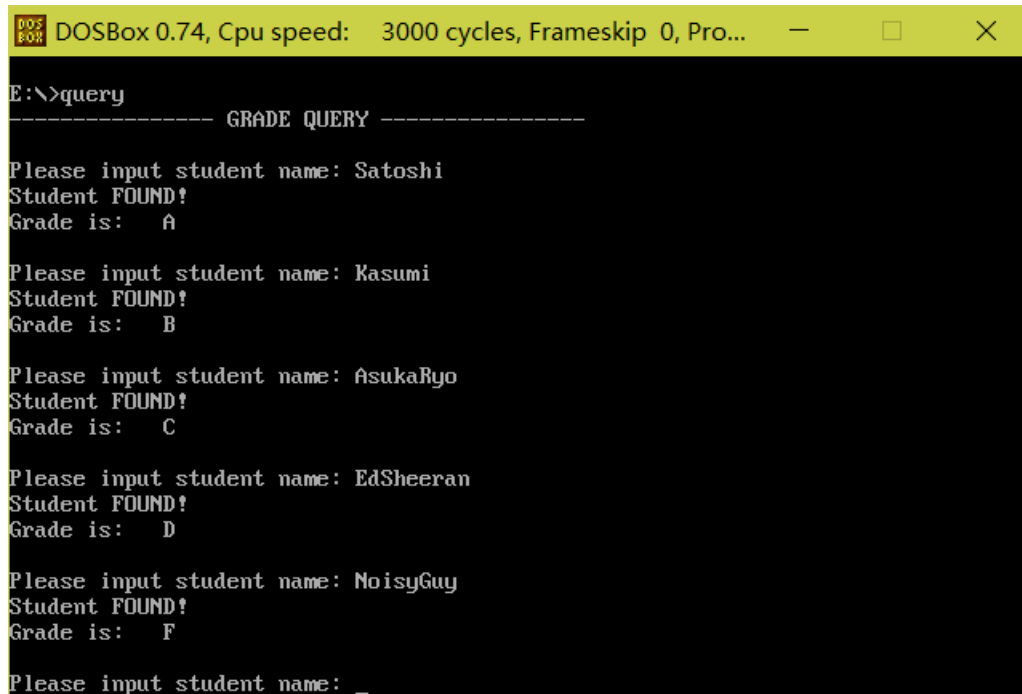


图 3.16 功能一实现示意（回车键提示重输，以及输入字符“q”退出程序）

输入第一个名字并回车后，程序即可计算出 BUF 中所有学生的平均成绩。直接输入 BUF 中记载

## 汇编语言程序设计实验报告

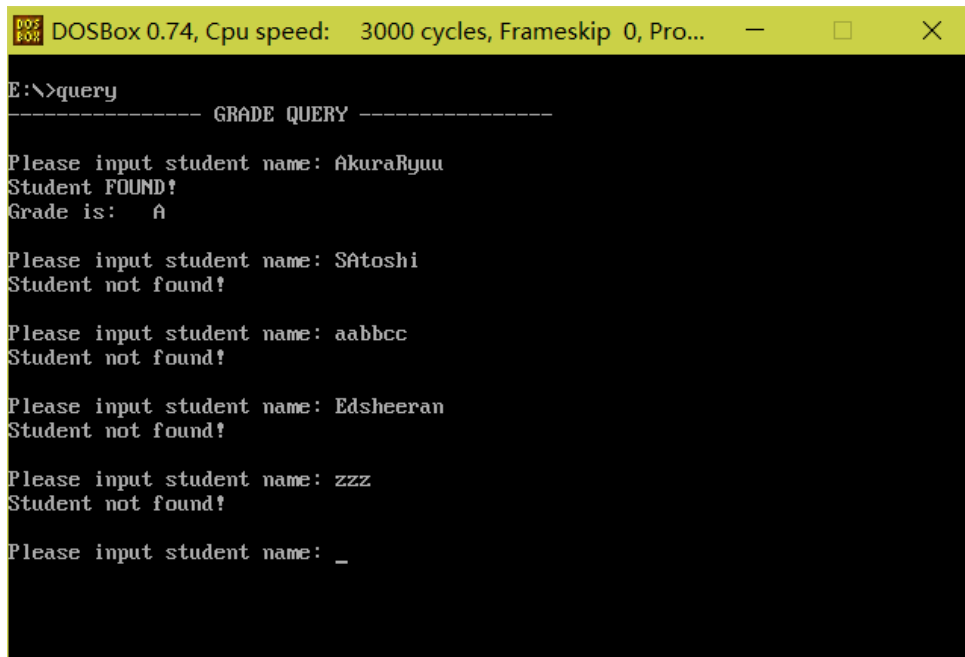
的学生名字，即可立刻查询到该学生，并根据该学生的平均成绩而输出对应的等级评价。



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...  
E:\>query  
----- GRADE QUERY -----  
  
Please input student name: Satoshi  
Student FOUND!  
Grade is: A  
  
Please input student name: Kasumi  
Student FOUND!  
Grade is: B  
  
Please input student name: AsukaRyo  
Student FOUND!  
Grade is: C  
  
Please input student name: EdSheeran  
Student FOUND!  
Grade is: D  
  
Please input student name: NoisyGuy  
Student FOUND!  
Grade is: F  
  
Please input student name: _
```

图 3.17 其余功能综合演示（查询学生信息，并根据运算所得平均成绩给出等级评定）

如果输入的学生不存在，则输出错误提示：



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...  
E:\>query  
----- GRADE QUERY -----  
  
Please input student name: AkuraRyu  
Student FOUND!  
Grade is: A  
  
Please input student name: Satoshi  
Student not found!  
  
Please input student name: aabbcc  
Student not found!  
  
Please input student name: Edsheeran  
Student not found!  
  
Please input student name: zzz  
Student not found!  
  
Please input student name: _
```

图 3.18 输入不存在的学生名字时，给出错误提示

# 汇编语言程序设计实验报告

---

综上测试可知，程序正常实现了所要求的全部功能。

## 4 总结与体会

### 4.1 对实验所涉及知识点的主要认识与收获

#### 4.1.1 如何编写一个有一定规模的汇编程序

任务三中所要求编写的学生成绩查询程序，从规模上看已经可以算是一个“大型”的汇编程序了。一是功能较为复杂，四个主打功能——用户输入解析、学生查询、平均成绩计算、等级评定输出，每个功能都是一个含金量高的部分；二是结合我们的课程为二学位课程这一事实，如此代码量的确可观，毕竟我们与主修本专业的同学相比，要求是有所不同的。

不过，不管代码如何复杂，本质上它们终究是汇编语言的应用。掌握强有力的程序设计思想、必要的指令用法，以及程序结构的编写，再大的程序都能玩转于键盘之间。

##### 第一，模块化与类模块化设计思想。

大程序的共同特点，就是有若干项各自相对独立的功能，且每个功能都具有一定的复杂度。因此，采用模块化设计思想与类模块化设计思想是有必要的。

模块化设计思想是高级语言中的基本概念，它将每个程序功能作为一个独立的模块，各模块各自独立地进行开发，并最终有机组合成一个完整的程序。而类模块化设计思想则是脱胎于模块化设计思想的另一个概念，它同样用“各自独立”的处理思想来编写每一个功能，只是没有用到编程语言中的模块化编程支持特性，如多文件编程。所以，前者适用于大型应用程序，而后者则适用于规模较小、功能划分相对简单的程序，本任务即是。

以本任务为例，本任务的四个功能其实可以在同一个代码段中完成，若按照模块化思想拆分成多个文件，势必会使简单的问题复杂化，因此用类模块化思想即可。类模块化思想的实现相当简单，只需在同一个代码段中分区块编写各功能的代码（或将功能编写在子程序中），并用明显而格式统一的注释相隔。这样，既可以使各功能处在相对独立的地位中，更充分发挥线性单过程程序简洁明了的优势。

##### 第二，程序结构的编写。

不同于高级语言，汇编语言的两大基本程序结构——条件结构和循环结构，都需要我们自己用比较指令（CMP）、跳转指令（J（Jump）开头的系列语句）和循环指令（LOOP）来部署，而不能用“关键字+表达式+语句块”这样我们早已熟知的调用方式。（实际上，对于条件结构，汇编语言也提供了条件判断伪指令，但该指令只是对有关跳转指令的封装，不是我们学习的重点。）

但是，从另一个角度来说，汇编语言最能帮助我们掌握程序结构的工作原理。不管是条件结构

# 汇编语言程序设计实验报告

---

还是循环结构，本质上都是条件判断和跳转的组合，只是部署上有一定的差异——跳转标志位的设置。因此，熟练掌握这两种指令，加上精妙的跳转标志位的安排，再复杂的程序编写都不在话下。

## 第三，宏与子过程的运用。

编写大程序时，往往会不可避免地用到重复的语句块，最典型的莫过于屏幕输出字符串的那三行指令。大程序自然免不了经常打印屏幕提示，若每次输出屏幕提示时都打上那三行语句，势必会啰嗦繁复，还会提高出错的可能。

如果将同样的指令封装在一起，需要的时候直接调用之，整个程序就会立刻清爽起来。汇编语言提供了两种封装语句的方式：宏与子过程。二者都可以把若干代码封装在一起，并给用户提供调用的接口：对于宏，直接使用其名字，加上要传入的参数；对于子过程，则使用 CALL 语句，并通过寄存器、存储单元等途径传参。

当然，二者的原理是有本质的区别的，我们在使用前免不了“三思”。宏的原理是汇编（编译）时进行代码替换，即汇编过程中，代码中调用了宏的地方会被自动替换成宏体，无需专门分配存储空间；而子程序本身就是代码的一部分，占用代码段的空间，还要为其分配入口地址。

而原理不同，也就决定了其效能的不同：宏对时间的消耗仅集中在汇编过程中，对代码的效率无影响；而子程序的每一次调用都要占用系统资源，若采用嵌套调用甚至递归，还会大幅降低其效率。因此在做决定之前，一定要三思，根据待封装代码的特点来决定使用宏还是子程序。我个人建议，像屏幕打印输出这样纯机械性、重复率高的语句，用宏比较合适；对于功能相对独立，或逻辑性强的语句，还是建议选择子过程。另外，由于子过程的参数传递比较难实现，因此对于需要传递大量参数的语句，用宏也是不错的选择，因为宏支持参数表。

## 4.2 经验教训与发现的问题

### 4.2.1 字符串输入功能（INT 10H）的缓冲区要求

在编写接收用户键入字符串的那一部分时，总是逃不开这样一个问题：在弹出“Please input student name:”的提示符后，程序立即退出，根本不给我输入的机会。检查系统功能调用的语句，发现与课本上的完全一致，没有显著错误；检查缓冲区变量 IN\_NAME 的定义时，一时半会也没发现错误。那会是哪里出现问题呢？

后来结合课本“系统功能调用”部分发现，问题正出在我对缓冲区变量 IN\_NAME 的定义上。我的 IN\_NAME 定义只有区区一行：

```
IN_NAME    DB 11
```

然而，字符串输入功能对缓冲区的要求是：

```
IN_NAME    DB 11
            DB ?
            DB 11 DUP(0)
```

# 汇编语言程序设计实验报告

---

从要求中可以看出，缓冲区至少要有三个数据域：最大字符串长度、实际输入字符串长度，以及字符串数据域。如果只定义一个数据域，系统当然会无法识别，视之为错误，就强行退出程序了。

## 参考文献

- [1] 王元珍, 曹忠升, 韩宗芬. 《80X86 汇编语言程序设计》. 版本(2005 版). 武汉: 华中科技大学出版社, 2005.