

汇编语言字符串处理完整模板

第一部分：大小写转换模板

1. 核心位运算转换

```

; =====
; 1.1 一键大小写转换（无检查）
; =====
; 转小写
OR AL, 20H      ; AL = AL | 00100000

; 转大写
AND AL, 0DFH    ; AL = AL & 11011111

; 大小写互换
XOR AL, 20H     ; AL = AL ^ 00100000

; =====
; 1.2 安全大小写转换（带检查）
; =====

; 安全转小写（仅大写字母转换）
SAFE_TO_LOWER:
    CMP AL, 'A'
    JB  NOT_UPPER
    CMP AL, 'Z'
    JA  NOT_UPPER
    OR  AL, 20H
NOT_UPPER:
    RET

; 安全转大写（仅小写字母转换）
SAFE_TO_UPPER:
    CMP AL, 'a'
    JB  NOT_LOWER
    CMP AL, 'z'
    JA  NOT_LOWER
    AND AL, 0DFH
NOT_LOWER:
    RET

```

第二部分：字符串基本操作模板

2. 字符串长度计算

```

; =====
; 2.1 计算以'$'结尾的字符串长度
; =====
; 输入: DS:SI = 字符串地址
; 输出: CX = 字符串长度
STRLEN PROC NEAR

```

```

    PUSH SI
    MOV CX, 0

COUNT_LOOP:
    CMP BYTE PTR [SI], '$'
    JE  COUNT_DONE
    INC CX
    INC SI
    JMP COUNT_LOOP

COUNT_DONE:
    POP SI
    RET
STRLEN ENDP

; =====
; 2.2 计算0结尾的字符串长度（C风格）
; =====
STRLEN_ZERO PROC NEAR
    PUSH SI
    MOV CX, 0

COUNT_ZERO_LOOP:
    CMP BYTE PTR [SI], 0
    JE  COUNT_ZERO_DONE
    INC CX
    INC SI
    JMP COUNT_ZERO_LOOP

COUNT_ZERO_DONE:
    POP SI
    RET
STRLEN_ZERO ENDP

```

3. 字符串复制

```

; =====
; 3.1 复制以'$'结尾的字符串
; =====
; 输入: DS:SI = 源字符串, ES:DI = 目标地址
STRCOPY PROC NEAR
    PUSH SI
    PUSH DI
    PUSH AX

COPY_LOOP:
    MOV AL, [SI]
    MOV ES:[DI], AL
    CMP AL, '$'
    JE  COPY_DONE
    INC SI
    INC DI
    JMP COPY_LOOP

COPY_DONE:

```

```

    POP AX
    POP DI
    POP SI
    RET
STRCOPY ENDP

; =====
; 3.2 复制指定长度的字符串
; =====
; 输入: DS:SI=源, ES:DI=目标, CX=长度
STRNCOPY PROC NEAR
    PUSH SI
    PUSH DI
    PUSH CX
    PUSH AX

    JCXZ COPY_N_DONE
COPY_N_LOOP:
    MOV AL, [SI]
    MOV ES:[DI], AL
    INC SI
    INC DI
    LOOP COPY_N_LOOP

COPY_N_DONE:
    POP AX
    POP CX
    POP DI
    POP SI
    RET
STRNCOPY ENDP

```

4. 字符串比较

```

; =====
; 4.1 比较两个以'$'结尾的字符串
; =====
; 输入: DS:SI=字符串1, ES:DI=字符串2
; 输出: ZF=1相等, ZF=0不相等
STRCMP PROC NEAR
    PUSH SI
    PUSH DI
    PUSH AX

CMP_LOOP:
    MOV AL, [SI]
    CMP AL, ES:[DI]
    JNE CMP_NOT_EQUAL

    CMP AL, '$'
    JE  CMP_EQUAL

    INC SI
    INC DI
    JMP CMP_LOOP

```

```

CMP_EQUAL:
    CMP BYTE PTR ES:[DI], '$'
    JE  CMP_DONE_EQUAL
    JMP CMP_NOT_EQUAL

CMP_DONE_EQUAL:
    ; 设置ZF=1
    XOR AX, AX          ; AX=0, 设置ZF=1
    JMP CMP_EXIT

CMP_NOT_EQUAL:
    ; 设置ZF=0
    MOV AX, 1           ; AX≠0, 设置ZF=0

CMP_EXIT:
    POP AX
    POP DI
    POP SI
    RET
STRCMP ENDP

; =====
; 4.2 不区分大小写的字符串比较
; =====

STRCMP_NOCASE PROC NEAR
    PUSH SI
    PUSH DI
    PUSH BX

CMP_NC_LOOP:
    MOV AL, [SI]
    MOV BL, ES:[DI]

    ; 都转小写比较
    CALL TO_LOWER      ; AL转小写
    MOV AH, AL         ; 保存AL
    MOV AL, BL
    CALL TO_LOWER      ; BL转小写
    CMP AH, AL
    JNE CMP_NC_NOT_EQUAL

    CMP AH, '$'
    JE  CMP_NC_EQUAL

    INC SI
    INC DI
    JMP CMP_NC_LOOP

CMP_NC_EQUAL:
    CMP BYTE PTR ES:[DI], '$'
    JE  CMP_NC_DONE_EQUAL

CMP_NC_NOT_EQUAL:
    STC                ; CF=1表示不相等
    JMP CMP_NC_EXIT

```

```

CMP_NC_DONE_EQUAL:
    CLC                ; CF=0表示相等

CMP_NC_EXIT:
    POP BX
    POP DI
    POP SI
    RET
STRCMP_NOCASE ENDP

```

第三部分：字符串转换模板

5. 大小写转换函数

```

; =====
; 5.1 完整大小写转换子程序
; =====

TO_LOWER PROC NEAR
    ; 输入: AL=字符
    ; 输出: AL=小写字符
    PUSH BX
    MOV BL, AL

    CMP AL, 'A'
    JB TL_EXIT
    CMP AL, 'Z'
    JA TL_EXIT
    OR AL, 20H

TL_EXIT:
    POP BX
    RET
TO_LOWER ENDP

TO_UPPER PROC NEAR
    ; 输入: AL=字符
    ; 输出: AL=大写字符
    PUSH BX
    MOV BL, AL

    CMP AL, 'a'
    JB TU_EXIT
    CMP AL, 'z'
    JA TU_EXIT
    AND AL, 0DFH

TU_EXIT:
    POP BX
    RET
TO_UPPER ENDP

; =====

```

; 5.2 字符串批量大小写转换

; =====

; 字符串转小写

; 输入: DS:SI=字符串地址, CX=长度

STRING_TO_LOWER PROC NEAR

PUSH SI

PUSH CX

PUSH AX

JCXZ STL_DONE

STL_LOOP:

MOV AL, [SI]

CALL TO_LOWER

MOV [SI], AL

INC SI

LOOP STL_LOOP

STL_DONE:

POP AX

POP CX

POP SI

RET

STRING_TO_LOWER ENDP

; 字符串转大写

STRING_TO_UPPER PROC NEAR

PUSH SI

PUSH CX

PUSH AX

JCXZ STU_DONE

STU_LOOP:

MOV AL, [SI]

CMP AL, 'a'

JB STU_SKIP

CMP AL, 'z'

JA STU_SKIP

AND AL, 0DFH

MOV [SI], AL

STU_SKIP:

INC SI

LOOP STU_LOOP

STU_DONE:

POP AX

POP CX

POP SI

RET

STRING_TO_UPPER ENDP

6. 数字字符串转换

```
; =====
; 6.1 字符串转数字（十进制）
; =====
; 输入: DS:SI=数字字符串地址（以非数字结尾）
; 输出: AX=数值
STR_TO_NUM PROC NEAR
    PUSH SI
    PUSH BX
    PUSH DX

    XOR AX, AX        ; AX=0
    MOV BX, 10        ; 乘数

CONVERT_LOOP:
    MOV DL, [SI]
    CMP DL, '0'
    JB  CONVERT_DONE
    CMP DL, '9'
    JA  CONVERT_DONE

    SUB DL, '0'        ; 字符转数字
    IMUL BX             ; AX = AX * 10
    ADD AX, DX         ; 加新数字
    INC SI
    JMP CONVERT_LOOP

CONVERT_DONE:
    POP DX
    POP BX
    POP SI
    RET
STR_TO_NUM ENDP

; =====
; 6.2 数字转字符串（十进制）
; =====
; 输入: AX=数值, ES:DI=缓冲区地址
; 输出: 缓冲区填充字符串
NUM_TO_STR PROC NEAR
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH DI

    MOV BX, 10
    XOR CX, CX        ; CX=0（计数器）

    ; 处理0的特殊情况
    CMP AX, 0
    JNE DIVIDE_LOOP
    MOV BYTE PTR ES:[DI], '0'
    INC DI
```

```

MOV BYTE PTR ES:[DI], '$'
JMP NTS_DONE

DIVIDE_LOOP:
    XOR DX, DX          ; DX:AX / BX
    DIV BX              ; AX=商, DX=余数
    ADD DL, '0'         ; 余数转字符
    PUSH DX             ; 保存字符
    INC CX              ; 计数++
    CMP AX, 0
    JNE DIVIDE_LOOP

    ; 弹出并存储
POP_LOOP:
    POP DX
    MOV ES:[DI], DL
    INC DI
    LOOP POP_LOOP

    MOV BYTE PTR ES:[DI], '$'

NTS_DONE:
    POP DI
    POP DX
    POP CX
    POP BX
    POP AX
    RET
NUM_TO_STR ENDP

```

第四部分：字符串搜索与处理

7. 字符串搜索

```

; =====
; 7.1 在字符串中查找字符
; =====
; 输入: DS:SI=字符串, AL=要查找的字符
; 输出: SI=找到的位置(没找到为0), CF=1找到
STRCHR PROC NEAR
    PUSH SI
    PUSH BX

    MOV BL, AL          ; 保存要查找的字符

    CHR_LOOP:
        MOV AL, [SI]
        CMP AL, '$'
        JE  CHR_NOT_FOUND
        CMP AL, BL
        JE  CHR_FOUND
        INC SI
        JMP CHR_LOOP

```

```

CHR_FOUND:
    STC                ; CF=1表示找到
    JMP CHR_EXIT

CHR_NOT_FOUND:
    MOV SI, 0          ; 没找到, 返回0
    CLC                ; CF=0表示没找到

CHR_EXIT:
    POP BX
    POP SI
    RET
STRCHR ENDP

; =====
; 7.2 在字符串中查找子串
; =====
; 输入: DS:SI=主串, DS:BX=子串
; 输出: SI=找到的位置, CF=1找到
STRSTR PROC NEAR
    PUSH SI
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH DI

    MOV DI, BX        ; DI=子串

MAIN_LOOP:
    MOV AL, [SI]
    CMP AL, '$'
    JE  STR_NOT_FOUND

    ; 从当前位置开始比较
    PUSH SI
    PUSH DI
    MOV CX, 0

COMPARE_LOOP:
    MOV AL, [SI]
    MOV DL, [DI]
    CMP DL, '$'
    JE  STR_FOUND
    CMP AL, DL
    JNE COMPARE_FAIL
    INC SI
    INC DI
    JMP COMPARE_LOOP

COMPARE_FAIL:
    POP DI
    POP SI
    INC SI
    JMP MAIN_LOOP

STR_FOUND:

```

```

    POP DI
    POP SI          ; 恢复找到的位置
    STC
    JMP STR_EXIT

STR_NOT_FOUND:
    MOV SI, 0
    CLC

STR_EXIT:
    POP DI
    POP DX
    POP CX
    POP BX
    POP SI
    RET
STRSTR ENDP

```

8. 字符串连接

```

; =====
; 8.1 连接两个以'$'结尾的字符串
; =====
; 输入: DS:SI=字符串1, ES:DI=字符串2, ES:BX=结果缓冲区
STRCAT PROC NEAR
    PUSH SI
    PUSH DI
    PUSH BX
    PUSH AX

    ; 复制第一个字符串
    MOV SI, OFFSET STR1
    MOV DI, BX
    CALL COPY_STR1

    ; 复制第二个字符串（从缓冲区末尾继续）
    MOV SI, OFFSET STR2
    CALL COPY_STR2

    POP AX
    POP BX
    POP DI
    POP SI
    RET

COPY_STR1:
    MOV AL, [SI]
    MOV ES:[DI], AL
    CMP AL, '$'
    JE COPY_STR1_DONE
    INC SI
    INC DI
    JMP COPY_STR1
COPY_STR1_DONE:
    DEC DI          ; 回退到'$'位置

```

```

    RET

COPY_STR2:
    MOV AL, [SI]
    MOV ES:[DI], AL
    CMP AL, '$'
    JE  COPY_STR2_DONE
    INC SI
    INC DI
    JMP COPY_STR2
COPY_STR2_DONE:
    RET
STRCAT ENDP

```

第五部分：实用宏定义模板

9. 常用字符串宏

```

; =====
; 9.1 打印字符串宏
; =====
PRINT_STRING MACRO str_addr
    PUSH DX
    PUSH AX
    LEA DX, str_addr
    MOV AH, 09H
    INT 21H
    POP AX
    POP DX
ENDM

; =====
; 9.2 大小写转换宏
; =====
TO_LOWER_MACRO MACRO reg
    LOCAL not_upper
    CMP reg, 'A'
    JB not_upper
    CMP reg, 'Z'
    JA not_upper
    OR reg, 20H
not_upper:
ENDM

TO_UPPER_MACRO MACRO reg
    LOCAL not_lower
    CMP reg, 'a'
    JB not_lower
    CMP reg, 'z'
    JA not_lower
    AND reg, 0DFH
not_lower:
ENDM

```

```

; =====
; 9.3 字符串复制宏
; =====

STRCPY_MACRO MACRO src, dest
    LOCAL copy_loop, copy_done
    PUSH SI
    PUSH DI
    PUSH AX

    LEA SI, src
    LEA DI, dest

copy_loop:
    MOV AL, [SI]
    MOV [DI], AL
    CMP AL, '$'
    JE copy_done
    INC SI
    INC DI
    JMP copy_loop

copy_done:
    POP AX
    POP DI
    POP SI
ENDM

```

第六部分：DOS输入输出模板

10. DOS字符串I/O

```

; =====
; 10.1 DOS输入缓冲区结构
; =====

INPUT_BUFFER STRUC
    max_len    DB 50          ; 最大长度
    actual_len DB ?           ; 实际长度（DOS填充）
    data       DB 50 DUP('$') ; 数据区
INPUT_BUFFER ENDS

; =====
; 10.2 读取字符串
; =====

READ_STRING PROC NEAR
    ; 输入：DS:DX=缓冲区地址
    ; 输出：缓冲区填充

    PUSH AX
    MOV AH, 0AH
    INT 21H

    ; 添加'$'结束符
    MOV BL, [DX+1]          ; 实际长度
    MOV BH, 0
    MOV BYTE PTR [BX+DX+2], '$'

```

```

    ; 换行
    CALL NEWLINE

    POP AX
    RET
READ_STRING ENDP

; =====
; 10.3 显示字符串
; =====
DISPLAY_STRING PROC NEAR
    ; 输入: DS:DX=字符串地址 (以'$'结尾)
    PUSH AX
    MOV AH, 09H
    INT 21H
    POP AX
    RET
DISPLAY_STRING ENDP

; =====
; 10.4 换行
; =====
NEWLINE PROC NEAR
    PUSH AX
    PUSH DX
    MOV AH, 02H
    MOV DL, 0DH
    INT 21H
    MOV DL, 0AH
    INT 21H
    POP DX
    POP AX
    RET
NEWLINE ENDP

```

快速参考表

```

; =====
; ASCII参考表
; =====
; 大写 A-Z: 41H-5AH
; 小写 a-z: 61H-7AH
; 数字 0-9: 30H-39H
; 空格: 20H, '$': 24H, 回车: 0DH, 换行: 0AH

; =====
; 常用掩码
; =====
; 转小写: OR 20H (00100000)
; 转大写: AND DFH (11011111)
; 互换: XOR 20H (00100000)

; =====
; DOS功能调用

```

```
; =====  
; 显示字符串: AH=09H, DS:DX=字符串地址  
; 输入字符串: AH=0AH, DS:DX=缓冲区地址  
; 显示字符:   AH=02H, DL=字符  
; 读取字符:   AH=01H, AL=字符
```

使用示例

```
; 示例: 读取字符串, 转大写, 显示  
DATA SEGMENT  
    prompt DB 'Enter string: $'  
    buffer DB 50, ?, 50 DUP('$')  
    result DB 50 DUP('$')  
DATA ENDS  
  
CODE SEGMENT  
    ASSUME CS:CODE, DS:DATA  
  
START:  
    MOV AX, DATA  
    MOV DS, AX  
  
    ; 显示提示  
    PRINT_STRING prompt  
  
    ; 读取字符串  
    LEA DX, buffer  
    CALL READ_STRING  
  
    ; 转大写  
    LEA SI, buffer+2      ; 跳过长度字节  
    MOV CL, buffer+1      ; 实际长度  
    MOV CH, 0  
    CALL STRING_TO_UPPER  
  
    ; 显示结果  
    LEA DX, buffer+2  
    CALL DISPLAY_STRING  
  
    MOV AH, 4CH  
    INT 21H  
  
    ; 包含所有需要的函数...  
CODE ENDS  
END START
```

这个模板包含了汇编语言中最常用的字符串处理功能，可以直接复制使用或根据需求修改。

