数字逻辑与处理器作业
——汇编程序设计

2022 年 4 月 20 日

作业概述

本次作业要求同学们运用在大作业第一部分练习的 MIPS 汇编操作，根据我们提供的串匹配的 C 语言代码，在 MARS 模拟器上将其翻译成 MIPS 汇编指令，然后编译，运行，调试，并通过测试。目的在于:理解如何汇编语言如何完成实现具体的算法功能，同时学会如何编写调试大型汇编程序。

串匹配问题

给定一个字符串 S = "$a0a1 \cdots an$"和一个模式串 P = "p0p1 …pm"，判断模式串 P 在字符串 S 中出现的次数。

例：

字符串 S = “linux is not unix is not unix is not unix”

模式串 P = “unix”

模式串 P 在字符串 S 中出现了 3 次“linux is not unix is not unix is not unix”，因此程序输出结果 3。

输入/输出格式

输入文件为文本文件，文件中第一行是字符串 S，第二行是模式串 P。具体可参考提供的测试文件 test.dat

实验内容

本次作业提供了实现串匹配问题的三种不同算法的 C 语言代码：暴力算法（Brute-Force）、Horspool 算法和 KMP 算法。同时提供了包括文件读取和结果输出在内的三个汇编语言文件，需要各位同学补全三个汇编语言文件中相应的算法部分，以实现串匹配问题的三种算法。

算法原理可以参考 2021-2022 学年秋季学期《数据与算法》课件

作业要求

作业用一个压缩包提交，压缩包名称：”学号_姓名.7z”。推荐用 7z 格式，其他常见压缩格式也可以。

压缩包打开后需要包含:

一个“实验报告.pdf”文件，报告中介绍寄存器和变量之间的分配关系，过程调用时进行的操作以及最终程序输出等内容。

一个“string_matching_bf.asm”,一个“string_matching_horspool.asm”,一个“string_matching_kmp.asm”

评分标准

本次作业评分主要参考程序的正确性。最终将会根据提交的汇编程序代码，测试多个不同样例。当样例结果出现错误时，将根据实验报告中叙述的汇编程序编写思路酌情给分。

附录 A：C 语言代码

string_matching_bf.cpp

```
1.  #include "stdio.h"
2.  void main()
3.  {
4.      FILE * infile ,*outfile;
```

```
    int i,max_num=0,id;
    int* buffer;
    buffer = new int[2];
    infile = fopen("a.in","rb");
    fread(buffer, 4, 2, infile);
    fclose(infile);
    outfile = fopen("a.out","wb");
    fwrite(buffer, 4, 2, outfile);
    fclose(outfile);
    scanf("%d",&i);
    i = i + 1
    printf("%d",i);
}
```

string_matching_horspool.cpp

```
#include <stdio.h>

int horspool(int len_str, char* str, int len_pattern, char* pattern);

int main() {
    FILE* f;
    int len_str, len_pattern, cnt;
    char str[512], pattern[512];

    //read two lines from file
    f = fopen("test.dat", "rb");
    for(len_str = 0; len_str < 512; len_str += 1) {
        str[len_str] = fgetc(f);
        if(str[len_str] == '\n') break;
    }
    for(len_pattern = 0; len_pattern < 512; len_pattern += 1) {
        pattern[len_pattern] = fgetc(f);
        if(pattern[len_pattern] == '\n') break;
    }
    fclose(f);

    //string matching
    cnt = horspool(len_str, str, len_pattern, pattern);
    //prinf
    printf("%d\n", cnt);
    return 0;
}

```

```
29. int horspool(int len_str, char* str, int len_pattern, char* pattern) {
30.     int cnt = 0, i, j;
31.     int table[256];
32. 
33.     //generate table
34.     for(i = 0; i < 256; ++i) table[i] = -1;
35.     for(i = 0; i < len_pattern; ++i) table[pattern[i]] = i;
36. 
37.     //matching
38.     i = len_pattern - 1;
39.     while(i < len_str) {
40.         j = 0;
41.         while(j < len_pattern && pattern[len_pattern - 1 - j] == str[i - j])
    j += 1;
42.         if(j == len_pattern){
43.             cnt += 1;
44.         }
45.         if(table[str[i]] + 1 <= len_pattern - 1 - j) i += len_pattern - 1 -
    table[str[i]];
46.         else i += 1;
47.     }
48.     return cnt;
49. }
```

string_matching_kmp.cpp

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  
4.  int kmp(int len_str, char* str, int len_pattern, char* pattern);
5.  int generateNext(int* next, int len_pattern, char* pattern);
6.  
7.  int main() {
8.      FILE* f;
9.      int len_str, len_pattern, cnt;
10.     char str[512], pattern[512];
11. 
12.     //read two lines from file
13.     f = fopen("test.dat", "rb");
14.     for(len_str = 0; len_str < 512; len_str += 1) {
15.         str[len_str] = fgetc(f);
16.         if(str[len_str] == '\n') break;
17.     }
18.     for(len_pattern = 0; len_pattern < 512; len_pattern += 1) {
```

```
        pattern[len_pattern] = fgetc(f);
        if(pattern[len_pattern] == '\n') break;
    }
    fclose(f);

    //string matching
    cnt = kmp(len_str, str, len_pattern, pattern);
    //prinf
    printf("%d\n", cnt);
    return 0;
}

int kmp(int len_str, char* str, int len_pattern, char* pattern) {
    int cnt = 0, i = 0, j = 0;
    int *next = (int*)malloc(len_pattern * 4);

    generateNext(next, len_pattern, pattern);

    //matching
    while(i < len_str) {
        if(pattern[j] == str[i]) {
            if(j == len_pattern - 1) {
                cnt += 1;
                j = next[len_pattern - 1];
                i += 1;
            }
            else {
                i += 1;
                j += 1;
            }
        }
        else {
            if(j > 0) j = next[j - 1];
            else i += 1;
        }
    }

    free(next);
    return cnt;
}

int generateNext(int *next, int len_pattern, char* pattern) {
    int i = 1, j = 0;
    if(len_pattern == 0) return 1;
```

```
    next[0] = 0;
    while(i < len_pattern) {
        if(pattern[i] == pattern[j]) {
            next[i] = j + 1;
            i += 1;
            j += 1;
        }
        else if(j > 0) j = next[j - 1];
        else next[i++] = 0;
    }
    return 0;
}
```

附录 B：读取文件汇编语言代码

```
.data
str: .space 512
pattern: .space 512
filename: .asciiz "test.dat"

.text
main:
#fopen
la $a0, filename #load filename
li $a1, 0 #flag
li $a2, 0 #mode
li $v0, 13 #open file syscall index
syscall

#read str
move $a0, $v0 #load file description to $a0
la $a1, str
li $a2, 1
li $s0, 0 #len_pattern = 0
read_str_entry:
slti $t0, $s0, 512
beqz $t0, read_str_exit
li $v0, 14 #read file syscall index
syscall
lb $t0, 0($a1)
addi $t1, $zero, '\n'
beq $t0, $t1, read_str_exit
addi $a1, $a1, 1
addi $s0, $s0, 1
```

```
30. j read_str_entry
31. read_str_exit:
32.
33. #read pattern
34. la $a1, pattern
35. li $a2, 1
36. li $s1, 0 #len_pattern = 0
37. read_pattern_entry:
38. slti $t0, $s1, 512
39. beqz $t0, read_pattern_exit
40. li $v0, 14 #read file syscall index
41. syscall
42. lb $t0, 0($a1)
43. addi $t1, $zero, '\n'
44. beq $t0, $t1, read_pattern_exit
45. addi $a1, $a1, 1
46. addi $s1, $s1, 1
47. j read_pattern_entry
48. read_pattern_exit:
49.
50. #close file
51. li $v0, 16 #close file syscall index
52. syscall
53.
54. #call brute_force
55. move $a0, $s0
56. la $a1, str
57. move $a2, $s1
58. la $a3, pattern
59. jal brute_force
60.
61. #printf
62. move $a0, $v0
63. li $v0, 1
64. syscall
65. #return 0
66. li $a0, 0
67. li $v0, 17
68. syscall
69.
70.
71. brute_force:
72. ##### your code here #####
```