

TRƯỜNG ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH

**KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH**



**KIẾN TRÚC MÁY TÍNH (C02008)**

---

**BÁO CÁO BÀI TẬP LỚN 02**

**NHÓM 02**

ĐỀ 7: Xác định vị trí cuối cùng của chuỗi "Ten\_nhom"  
trong chuỗi "pString", chuỗi pString có N phần tử,  $N = 1000$

---

Sinh viên thực hiện:

1. Nguyễn Minh Khôi – 1611657
2. Nguyễn Trọng Nghĩa – 1612212
3. Nguyễn Văn Tường - 1614028

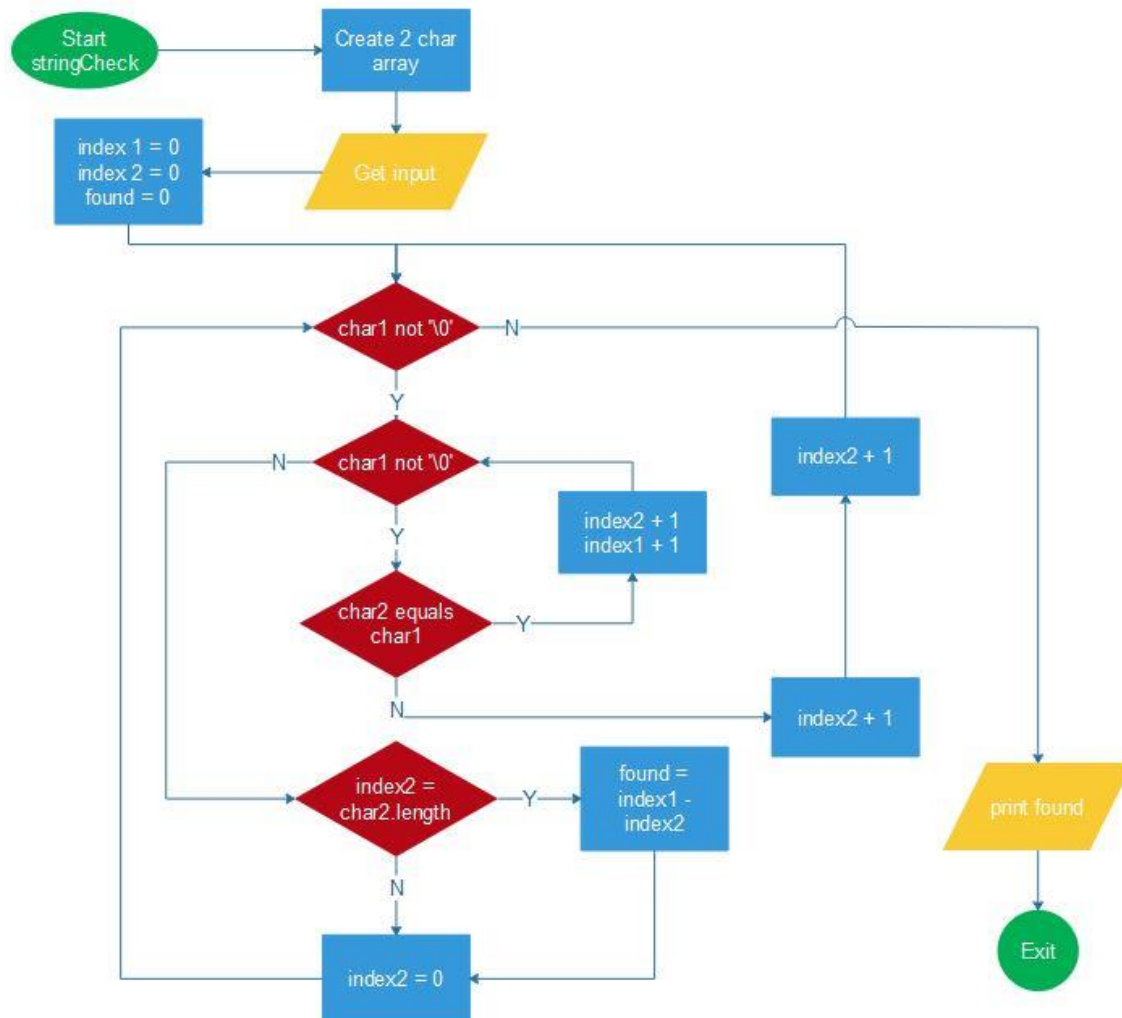
*Thành phố Hồ Chí Minh, tháng 12/2017*

## 1. Thiết kế giải thuật

### a. Ý tưởng

Gọi **mString** là chuỗi cần xác định nếu có tồn tại trong chuỗi **pString**. Trong khi phần tử đang xét của **pString** chưa phải là `\0`, ta duyệt tất cả các kí tự trong chuỗi **mString**, nếu phần tử của **pString** bằng với phần tử của **mString**, tăng chỉ số của **pString** và **mString** lên 1, nếu không, tăng chỉ số của **pString** lên 1 và thoát khỏi vòng lặp duyệt chuỗi **mString**. Nếu chỉ số của **mString** bằng với độ dài của chuỗi **mString**, lúc này chuỗi **mString** được tìm thấy, gán vị trí tìm thấy và gán chỉ số của chuỗi **mString** để chuẩn bị cho vòng lặp duyệt kí tự tiếp theo trong chuỗi **pString**.

### b. Flow-chart





### c. Giải thuật viết bằng ngôn ngữ C (main.c)

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define NUM 1000

int main(int argc, char** argv) {
    // create base string to compare
    char* pString = (char*)malloc(NUM * sizeof(char));
    // create string to compare
    char* myString = (char*)malloc(NUM * sizeof(char));

    // ok, let's read from screen our 2 screen
    scanf("%s%s", pString, myString);

    // pIndex is the index of character in pString
    int pIndex = 0;
    // myIndex is the index of character in myString
    int mIndex = 0;
    // found will record the position of myString when matched pString
    int found = 0;
    while (pString[pIndex] != '\0') {
        // traversal the pString character, and each time, compare with
myString
        while (myString[mIndex] != '\0') {
            // if myString character matched pString character
            // increase index of both
            // till the end of myString
            // if not reaching the end, which means unmatched, reset
myString index and increase pString index
            if (myString[mIndex] == pString[pIndex]) {
                mIndex++;
                pIndex++;
            }
            else {
                pIndex++;
                break;
            }
        }
        // after checking match, if mIndex equals to myString length
        // which means matched, record the position
        // 'cause we continuously searching, "found" at last will record the
final position those two matched
        if (mIndex == strlen(myString)) found = pIndex - mIndex;
        mIndex = 0;
    }

    // print the position of last found myString in pString
    printf("%d", found);
    return 0;
}
```

## 2. Biên dịch giải thuật viết bằng ngôn ngữ C với công cụ MIPS Cross Compiler (main.asm)

```
1      .file    1 "ktmt_11_code.c" # file name compiled
2      .section .mdebug.abi32 # for debugger 1-3
3      .previous
4      .nan      legacy # there are two type of NaN ("signalling" and "quiet", WIKI for
more details), "legacy" express using "signalling" NaN
5      .module    fp=xx # -mfpxx: floating-point number is executed exactly 32-
bit register or 64-bit register??
6      .module    nooddspre
7      .rdata
8      .align    2 # each instruction cost 4-byte
9      .LC0:
10     .ascii    "%s%s\000" # input instruction description
11     .align    2 # 4-byte instruction, pc + 4 each time
12     .LC1:
13     .ascii    "%d\000" # output instruction description
14     .text
15     .align    2
16     .globl    main # name of the scope can be called by another
17     .set      nomips16 # for certain that, this mips work on normal 32-bit mode,
not 16-bit mode
18     .set      nomicromips #micromips is a supersets of MIPS32 and MIPS64
which changes some 32-bit instruction to 16-bit version for using mix in MIPS16e
19     .ent      main # .ent makes the entry of main, tell the debugger
20     .type      main, @function
21     main:
22     # first we will create 56-bit storage in heap, storing our char array by calling
malloc
23     # and store the address of $fp + 56 to our $ra
24     .frame     $fp,56,$31          # vars= 24, regs= 3/0, args= 16, gp= 0
25     # frame will create a space of 56-bit in stack, pointer to $31, (heap data)
26     .mask      0xc0010000,-4      # for debugger, store variable at $16 and cost
4-bit lower
27     .fmask     0x00000000,0
28     .set      noreorder # tell the assembler not to move(rearrange) our instruction
29     .set      nomacro # no macro to translate-no statement is more than one
instruction
30     # move stack pointer to 56-bit lower
31     # first store value of $ra to $sp + 52 ($ra = $fp + 56)
32     # then store value of $fp
33     # then store value of $s0
```



```
34      addiu  $sp,$sp,-56
35      sw     $31,52($sp)
36      sw     $fp,48($sp)
37      sw     $16,44($sp)
38      # now $fp get the address of $sp
39      # then store value of $a0, $a1
40      move   $fp,$sp
41      sw     $4,56($fp)
42      sw     $5,60($fp)
43      # create 1000-bit in $a0
44      li     $4,1000                # 0x3e8 #create array of char in $v0
45      jal    malloc
46      nop

47      # after getting input from screen, store it to $fp + 28
48      sw     $2,28($fp)
49      # get another string
50      li     $4,1000                # 0x3e8
51      jal    malloc
52      nop

53      # do the same as above
54      sw     $2,32($fp)
55      lui    $2,%hi(.LC0)
56      addiu  $4,$2,%lo(.LC0)
57      lw     $5,28($fp) # pString
58      lw     $6,32($fp) # myString
59      jal    scanf
60      nop

61      sw     $0,16($fp) # pIndex = 0
62      sw     $0,20($fp) # mIndex = 0
63      sw     $0,24($fp) # found = 0
64      b      .L2
65      nop

66      .L8:
67      b      .L3
68      nop

69      .L6:
70      lw     $2,20($fp) # load myString and its mIndex
71      lw     $3,32($fp)
72      addu   $2,$3,$2
73      lbu    $3,0($2) # assign it to $v1
```



```
74      lw      $2,16($fp) # load pString and its pIndex
75      lw      $4,28($fp)
76      addu    $2,$4,$2
77      lbu     $2,0($2) # assign it to $v0
78      bne     $3,$2,.L4 # if (myString[mIndex] == pString[pIndex])
79      nop

80      lw      $2,20($fp) # mIndex increase 1 unit # load it from $fp
81      addiu    $2,$2,1 # increase
82      sw      $2,20($fp) # store back to $fp
83      lw      $2,16($fp) # the same with pIndex
84      addiu    $2,$2,1
85      sw      $2,16($fp)
86      b       .L3
87      nop

88      # .L4 = else
89      .L4:
90      lw      $2,16($fp) # load from $fp pIndex
91      addiu    $2,$2,1 # then increase it to 1 unit
92      sw      $2,16($fp) # store it again back to $fp
93      b       .L5
94      nop

95      # .L3 = while (myString[mIndex] != '\0')
96      .L3:
97      lw      $2,20($fp) # mIndex
98      lw      $3,32($fp) # myString
99      addu    $2,$3,$2 # increase myString to myString + mIndex
100     lbu     $2,0($2)
101     bne     $2,$0,.L6 # if myString character not equals to zero
102     nop

103     # .L5 = if (mIndex == strlen(myString)) found = pIndex - mIndex;
104     .L5:
105     lw      $16,20($fp) # mIndex
106     lw      $4,32($fp) # myString
107     jal     strlen
108     nop

109     bne     $16,$2,.L7 # compare mIndex ($16) and strlen(myString) ($2)
110     nop

111     lw      $3,16($fp) # load pIndex
112     lw      $2,20($fp) # load mIndex
```

```
113      subu    $2,$3,$2 # pIndex - mIndex
114      sw      $2,24($fp) # store to found variable address

115      # .L7 = mIndex = 0
116      .L7:
117      sw      $0,20($fp)
118      # .L2 = while (pString[index] != '\0')
119      .L2:
120      lw      $2,16($fp) # pString
121      lw      $3,28($fp) # pIndex
122      addu    $2,$3,$2 # increase pString to pString + index
123      lbu     $2,0($2)
124      bne     $2,$0,.L8 # if our pString character not equals to zero
125      nop

126      # print found variable
127      lui     $2,%hi(.LC1)
128      addiu   $4,$2,%lo(.LC1)
129      lw      $5,24($fp)
130      jal     printf
131      nop

132      # restore memmory
133      move    $2,$0
134      move    $sp,$fp
135      lw      $31,52($sp)
136      lw      $fp,48($sp)
137      lw      $16,44($sp)
138      addiu   $sp,$sp,56
139      jr      $31
140      nop

141      # dont care of these, it contrast which in the beginning
142      .set    macro
143      .set    reorder
144      .end    main
145      .size   main,.-main
146      .ident  "GCC: (Codescape GNU Tools 2016.05-03 for MIPS MTI Bare Metal)

4.9.2"
```

### 3. Phát hiện hazard

1 .data



```
2      strln: .space 1000
3      newline: .ascii "\n"
4      .text
5
6      main:
7          # move stack pointer to 56-bit lower
8          # first store value of $ra to $sp + 52 ($ra = $fp + 56)
9          # then store value of $fp
10         # then store value of $s0
11         addiu $sp,$sp,-56
12         sw    $v1,52($sp)
13         sw    $fp,48($sp)
14         sw    $s0,44($sp)
15         # now $fp get the address of $sp
16         # then store value of $a0, $a1
17         move  $fp,$sp          #forwarding
18         sw    $a0,56($fp)
19         sw    $a1,60($fp)
20         la    $t1,newline
21
22         # create 1000-bit in $a0
23         la    $a0, strln
24         lbu   $t1, 0($t1)      #reorder
25         li    $a1,1000        # 0x3e8
26         li    $v0, 8
27         syscall
28
29         # after getting input from screen, store it to $fp + 28
30         sw    $a0,28($fp)
31         # get another string
32         la    $a0, strln
33         li    $a2,1000        # 0x3e8
34         li    $v0, 8          #reorder
35         addi  $a0, $a0, 100
36
37         syscall
38
39         # do the same as above
40         sw    $a0, 32($fp)
41         lw    $a1,28($fp)     # pString
42         lw    $a2,32($fp)     # myString
43
44         sw    $0,16($fp)      # pIndex = 0
45         sw    $0,20($fp)      # mIndex = 0
46         sw    $0,24($fp)      # found = 0
```



```

47      b      .L2
48      nop                                #branch hazard
49
50      .L6:
51      lw      $v0,20($fp)  # load myString and its mIndex
52      lw      $v1,32($fp)
53      lw      $a0,28($fp)  #reorder
54      addu    $v0,$v1,$v0  #forwarding
55      lbu     $v1,0($v0)   # assign it to $v1
56      lw      $v0,16($fp)  # load pString and its pIndex
57      nop                                #load hazard
58      addu    $v0,$a0,$v0  #forwarding
59      lbu     $v0,0($v0)   # assign it to $v0
60      nop                                #load hazard
61      bne     $v1,$v0,.L4  # if (myString[mIndex] != pString[pIndex])
62      nop                                #Branch hazard
63
64      lw      $v0,20($fp)  # mIndex increase 1 unit # load it from $fp
65      nop                                #load hazard
66      addiu   $v0,$v0,1    # increase, forwarding
67      sw      $v0,20($fp)  # store back to $fp, forwarding
68      lw      $v0,16($fp)  # the same with pIndex
69      nop                                #load hazard
70      addiu   $v0,$v0,1    #forwarding
71      sw      $v0,16($fp)
72      b      .L3
73      nop                                #branch hazard
74
75      # .L4 = else
76      .L4:
77      lw      $v0,16($fp)  # load from $fp pIndex
78      nop                                #load hazard
79      addiu   $v0,$v0,1    # then increase it to 1 unit, forwarding
80      sw      $v0,16($fp)  # store it again back to $fp
81      b      .L5
82      nop                                #branch hazard
83
84      # .L3 = while (myString[mIndex] != '\0')
85      .L3:
86      lw      $v0,20($fp)  # mIndex
87      lw      $v1,32($fp)  # myString
88      nop                                #load hazard
89      addu    $v1,$v1,$v0  # increase myString to myString + mIndex
90      lbu     $v1,0($v1)
91      nop                                #load hazard

```



```
92         bne    $v1,$t1,.L6    # if myString character not equals to zero
93         nop                      #Branch hazard
94
95     # .L5 = if (mIndex == strlen(myString)) found = pIndex - mIndex;
96     .L5:
97         lw      $s0,20($fp)    # mIndex
98         lw      $a0,32($fp)    # myString
99
100        bne     $s0,$v0,.L7    #compare mIndex ($s0) and strlen(myString)
    ($v0)
101        nop                      #Branch hazard
102
103        lw      $v1,16($fp)    # load pIndex
104        lw      $v0,20($fp)    # load mIndex
105        nop                      #load hazard
106        subu    $v0,$v1,$v0    # pIndex - mIndex, forwarding
107        sw      $v0,24($fp)    # store to found variable address
108
109        # .L7 = mIndex = 0
110        .L7:
111        sw      $0,20($fp)
112
113        # .L2 = while (pString[index] != '\0')
114        .L2:
115        lw      $v0,16($fp)    # pIndex
116        lw      $v1,28($fp)    # pString
117        nop                      #load hazard
118        addu    $v0,$v1,$v0    # increase pString to pString + index, forwarding
119        lbu     $v0,0($v0)
120        nop                      #load hazard
121        bne     $v0,$0,.L3    # if our pString character not equals to zero
122        nop                      #Branch hazard
123
124        # print found variable
125        # lui    $v0,%hi(.LC1)
126        # addiu    $a0,$v0,%lo(.LC1)
127        lw      $a1,24($fp)
128        nop                      #load hazard
129        la      $a0,($a1)
130        li      $v0,1
131        syscall
132
133        # restore memmoryx
134        move    $v0,$0
135        move    $sp,$fp
```

```
136      lw      $v1,52($sp)
137      lw      $fp,48($sp)
138      lw      $s0,44($sp)
139      #addiu $sp,$sp,56
```

## 4. Xử lý hazard

```
1      .data
2      strln: .space 100
3      newline: .ascii "\n"
4      .text
5
6      main:
7          # move stack pointer to 56-bit lower
8          # first store value of $ra to $sp + 52 ($ra = $fp + 56)
9          # then store value of $fp
10         # then store value of $s0
11         addiu $sp,$sp,-56
12         sw     $v1,52($sp)
13         sw     $fp,48($sp)
14         sw     $s0,44($sp)
15         # now $fp get the address of $sp
16         # then store value of $a0, $a1
17         move   $fp,$sp
18         sw     $a0,56($fp)
19         sw     $a1,60($fp)
20         la     $t1, newline
21
22         # create 1000-bit in $a0
23         la     $a0, strln
24         lbu    $t1, 0($t1)          #reorder
25         li     $a1,1000             # 0x3e8
26         li     $v0, 8
27         syscall
28
29         # after getting input from screen, store it to $fp + 28
30         sw     $a0,28($fp)
31         # get another string
32         la     $a0, strln
33         li     $a2,1000             # 0x3e8
34         li     $v0, 8               #reorder
35         addi   $a0, $a0, 100
36
```



```
37      syscall
38
39      # do the same as above
40      sw      $a0, 32($fp)
41      lw      $a1, 28($fp) # pString
42      lw      $a2, 32($fp) # myString
43
44      sw      $0, 16($fp) # pIndex = 0
45      sw      $0, 20($fp) # mIndex = 0
46      sw      $0, 24($fp) # found = 0
47      b       .L2
48
49      .L8:
50      b       .L3
51
52      .L6:
53      lw      $v0, 20($fp) # load myString and its mIndex
54      lw      $v1, 32($fp)
55      lw      $a0, 28($fp) #reorder
56      addu    $v0, $v1, $v0
57      lbu     $v1, 0($v0) # assign it to $v1
58      lw      $v0, 16($fp) # load pString and its pIndex
59
60      addu    $v0, $a0, $v0
61      lbu     $v0, 0($v0) # assign it to $v0
62      bne     $v1, $v0, .L4 # if (myString[mIndex] != pString[pIndex])
63
64
65      lw      $v0, 20($fp) # mIndex increase 1 unit # load it from $fp
66      addiu   $v0, $v0, 1 # increase
67      sw      $v0, 20($fp) # store back to $fp
68      lw      $v0, 16($fp) # the same with pIndex
69      addiu   $v0, $v0, 1
70      sw      $v0, 16($fp)
71      b       .L3
72
73      # .L4 = else
74      .L4:
75      lw      $v0, 16($fp) # load from $fp pIndex
76      addiu   $v0, $v0, 1 # then increase it to 1 unit
77      sw      $v0, 16($fp) # store it again back to $fp
78      b       .L5
79
80
81      # .L3 = while (myString[mIndex] != '\0')
```



```
82     .L3:
83         lw     $v0,20($fp) # mIndex
84         lw     $v1,32($fp) # myString
85         addu   $v1,$v1,$v0 # increase myString to myString + mIndex
86         lbu    $v1,0($v1)
87         bne    $v1,$t1,.L6 # if myString character not equals to zero
88
89     # .L5 = if (mIndex == strlen(myString)) found = pIndex - mIndex;
90     .L5:
91         lw     $s0,20($fp) # mIndex
92         lw     $a0,32($fp) # myString
93
94         bne    $s0,$v0,.L7 # compare mIndex ($s0) and strlen(myString) ($v0)
95
96
97         lw     $v1,16($fp) # load pIndex
98         lw     $v0,20($fp) # load mIndex
99         subu   $v0,$v1,$v0 # pIndex - mIndex
100        sw     $v0,24($fp) # store to found variable address
101
102    # .L7 = mIndex = 0
103    .L7:
104        sw     $0,20($fp)
105
106    # .L2 = while (pString[index] != '\0')
107    .L2:
108        lw     $v0,16($fp) # pIndex
109        lw     $v1,28($fp) # pString
110        addu   $v0,$v1,$v0 # increase pString to pString + index
111        lbu    $v0,0($v0)
112        bne    $v0,$0,.L8 # if our pString character not equals to zero
113
114        # print found variable
115        # lui    $v0,%hi(.LC1)
116        # addiu   $a0,$v0,%lo(.LC1)
117        lw     $a1,24($fp)
118        la     $a0,($a1)
119        li     $v0,1
120        syscall
121
122        # restore memmoryx
123        move   $v0,$0
124        move   $sp,$fp
125        lw     $v1,52($sp)
126        lw     $fp,48($sp)
```

```
127      lw      $s0,44($sp)
128      #addiu $sp,$sp,56
```

## 5. Điều chỉnh cấu hình file để sử dụng MIPS MARS4\_5

```
1      .data
2          strln: .space 1000
3          newline: .ascii "\n"
4      .text
5
6      main:
7          # move stack pointer to 56-bit lower
8          # first store value of $ra to $sp + 52 ($ra = $fp + 56)
9          # then store value of $fp
10         # then store value of $s0
11         addiu $sp,$sp,-56
12         sw    $v1,52($sp)
13         sw    $s0,44($sp)
14         # now $fp get the address of $sp
15         # then store value of $a0, $a1
16         # move      $fp,$sp
17         sw    $a0,56($sp)
18         sw    $a1,60($sp)
19         la    $t1, newline
20
21         # create 1000-bit in $a0
22         la    $a0, strln
23         lbu   $t1, 0($t1)
24         li    $a1,1000          # 0x3e8
25         li    $v0, 8
26         syscall
27
28         # after getting input from screen, store it to $fp + 28
29         sw    $a0,28($sp)
30         # get another string
31         la    $a0, strln
32         li    $a2,1000
33         li    $v0, 8
34         addi  $a0, $a0, 1000     # 0x3e8
35         syscall
36
37         # do the same as above
38         sw    $a0, 32($sp)
```



```
39      lw      $a1,28($sp)      # pString
40      lw      $a2,32($sp)      # myString
41
42      sw      $0,16($sp)        # pIndex = 0
43      sw      $0,20($sp)        # mIndex = 0
44      sw      $0,24($sp)        # found = 0
45      b       .L2
46      nop
47
48      .L6:
49          lw      $v0,20($sp)    # load myString and its mIndex
50          lw      $v1,32($sp)
51          lw      $a0,28($sp)
52          addu    $v0,$v1,$v0
53
54          lbu     $v1,0($v0)     # assign it to $v1
55
56          lw      $v0,16($sp)    # load pString and its pIndex
57          nop
58          addu    $v0,$a0,$v0
59
60          lbu     $v0,0($v0)     # assign it to $v0
61          nop
62          bne     $v1,$v0,.L4    # if (myString[mIndex] != pString[pIndex])
63          nop
64
65          lw      $v0,20($sp)    # mIndex increase 1 unit # load it from
$fp
66          nop
67          addiu   $v0,$v0,1      # increase
68
69          sw      $v0,20($sp)    # store back to $sp
70          lw      $v0,16($sp)    # the same with pIndex
71          nop
72          addiu   $v0,$v0,1
73
74          sw      $v0,16($sp)
75          b       .L3
76          nop
77
78      # .L4 = else
79      .L4:
80          lw      $v0,16($sp)    # load from $sp pIndex
81          nop
82          addiu   $v0,$v0,1      # then increase it to 1 unit
```



```
83
84     sw    $v0,16($sp)      # store it again back to $sp
85     b     .L5
86     nop
87
88     # .L3 = while (myString[mIndex] != '\0')
89     .L3:
90         lw    $v0,20($sp)    # mIndex
91         lw    $v1,32($sp)    # myString
92         nop
93         addu   $v1,$v1,$v0    # increase myString to myString + mIndex
94
95         lbu    $v1,0($v1)
96         nop
97         bne    $v1,$t1,.L6    # if myString character not equals to zero
98         nop
99
100    # .L5 = if (mIndex == strlen(myString)) found = pIndex - mIndex;
101    .L5:
102        lw    $s0,20($sp)    # mIndex
103        lw    $a0,32($sp)    # myString
104
105        bne    $s0,$v0,.L7    # compare mIndex ($s0) and
        strlen(myString) ($v0)
106        nop
107        lw    $v0,20($sp)    # load mIndex
108        lw    $v1,16($sp)    # load pIndex
109        nop
110        subu   $v0,$v1,$v0    # pIndex - mIndex
111
112        sw    $v0,24($sp)    # store to found variable address
113
114    # .L7 = mIndex = 0
115    .L7:
116        sw    $0,20($sp)
117
118    # .L2 = while (pString[index] != '\0')
119    .L2:
120        lw    $v0,16($sp)    # pIndex
121        lw    $v1,28($sp)    # pString
122        nop
123        addu   $v0,$v1,$v0    # increase pString to pString + index
124
125        lbu    $v0,0($v0)
126        nop
```





```
127      bne    $v0,$0,.L3      # if our pString character not equals to zero
128      nop
129
130      # print found variable
131      lw      $a1,24($sp)
132      li      $v0,1
133      la      $a0,($a1)
134
135      syscall
136
137      # restore memmory
138      move    $v0,$0
139      lw      $v1,52($sp)
140      lw      $s0,44($sp)
```