TRƯỜNG ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH

KHOA KHOA HỌC & KĨ 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

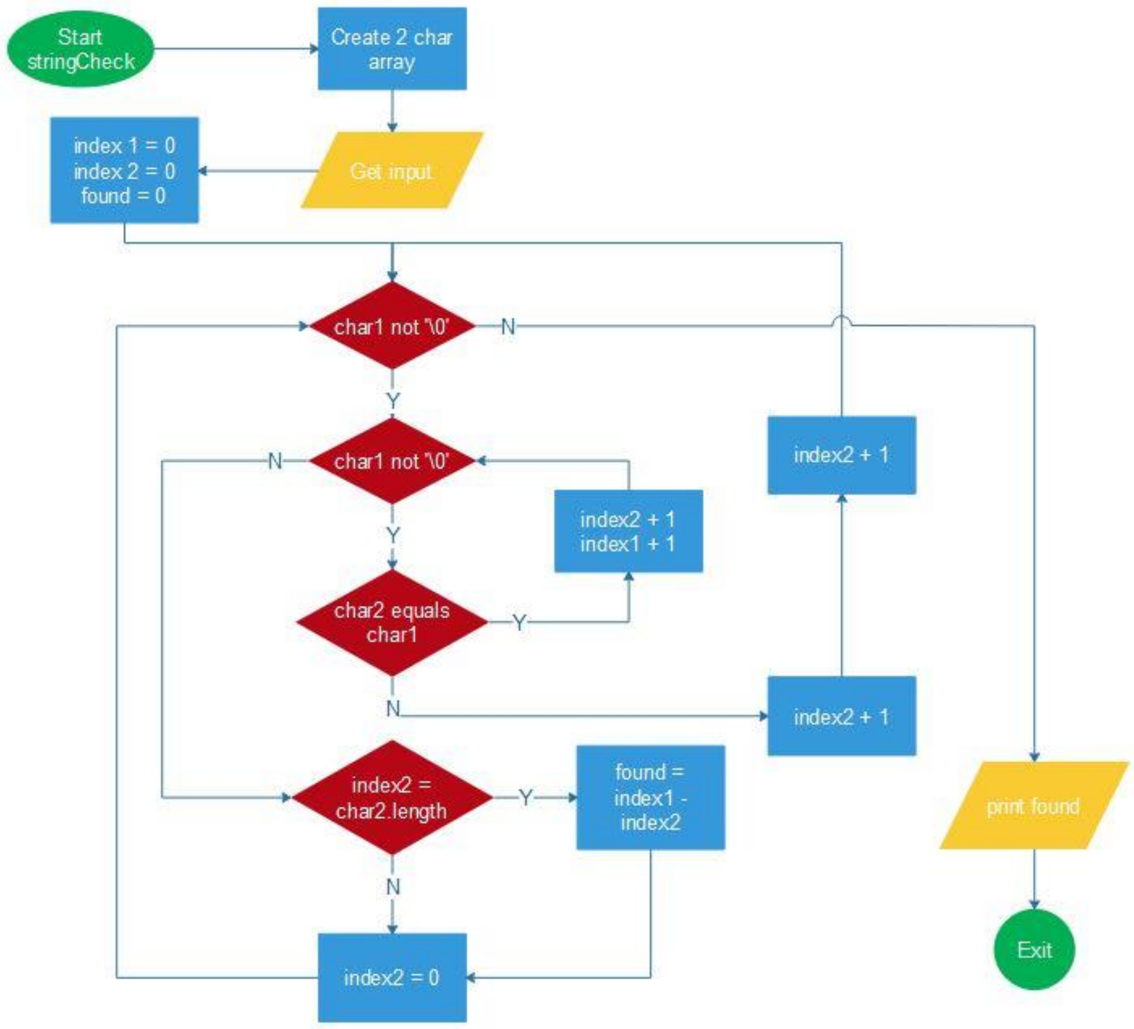
Thành phố Hồ Chí Minh, tháng 12/2017

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** đươc 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)

```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         nooddspreg
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
```

```
addiu   $sp,$sp,-56
sw      $31,52($sp)
sw      $fp,48($sp)
sw      $16,44($sp)
# now $fp get the address of $sp
# then store value of $a0, $a1
move    $fp,$sp
sw      $4,56($fp)
sw      $5,60($fp)
# create 1000-bit in $a0
li      $4,1000                     # 0x3e8 #create array of char in $v0
jal     malloc
nop

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

# do the same as above
sw      $2,32($fp)
lui     $2,%hi(.LC0)
addiu   $4,$2,%lo(.LC0)
lw      $5,28($fp) # pString
lw      $6,32($fp) # myString
jal     scanf
nop

sw      $0,16($fp) # pIndex = 0
sw      $0,20($fp) # mIndex = 0
sw      $0,24($fp) # found = 0
b       .L2
nop

.L8:
b       .L3
nop

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

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

        lw      $2,20($fp) # mIndex increase 1 unit # load it from $fp
        addiu   $2,$2,1 # increase
        sw      $2,20($fp) # store back to $fp
        lw      $2,16($fp) # the same with pIndex
        addiu   $2,$2,1
        sw      $2,16($fp)
        b       .L3
        nop

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

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

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

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

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

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

# .L7 = mIndex = 0
.L7:
sw      $0,20($fp)
# .L2 = while (pString[index] != '\0')
.L2:
lw      $2,16($fp) # pString
lw      $3,28($fp) # pIndex
addu    $2,$3,$2 # increase pString to pString + index
lbu     $2,0($2)
bne     $2,$0,.L8 # if our pString character not equals to zero
nop

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

# restore memmory
move    $2,$0
move    $sp,$fp
lw      $31,52($sp)
lw      $fp,48($sp)
lw      $16,44($sp)
addiu   $sp,$sp,56
jr      $31
nop

# dont care of these, it contrast which in the beginning
.set    macro
.set    reorder
.end    main
.size   main, .-main
.ident  "GCC: (Codescape GNU Tools 2016.05-03 for MIPS MTI Bare Metal)
4.9.2"
```

3. Phát hiện hazard

```
.data
```

```
        strIn: .space 1000
        newline: .ascii "\n"
.text

main:
        # move stack pointer to 56-bit lower
        # first store value of $ra to $sp + 52 ($ra = $fp + 56)
        # then store value of $fp
        # then store value of $s0
        addiu   $sp,$sp,-56
        sw      $v1,52($sp)
        sw      $fp,48($sp)
        sw      $s0,44($sp)
        # now $fp get the address of $sp
        # then store value of $a0, $a1
        move    $fp,$sp                 #forwarding
        sw      $a0,56($fp)
        sw      $a1,60($fp)
        la      $t1, newline

        # create 1000-bit in $a0
        la      $a0, strIn
        lbu     $t1, 0($t1)             #reorder
        li      $a1,1000                # 0x3e8
        li      $v0, 8
        syscall

        # after getting input from screen, store it to $fp + 28
        sw      $a0,28($fp)
        # get another string
        la      $a0, strIn
        li      $a2,1000        # 0x3e8
        li      $v0, 8          #reorder
        addi    $a0, $a0, 100

        syscall

        # do the same as above
        sw      $a0, 32($fp)
        lw      $a1,28($fp)     # pString
        lw      $a2,32($fp)     # myString

        sw      $0,16($fp)      # pIndex = 0
        sw      $0,20($fp)      # mIndex = 0
        sw      $0,24($fp)      # found = 0
        b       .L2
        nop                     #branch hazard

.L6:
        lw      $v0,20($fp)     # load myString and its mIndex
        lw      $v1,32($fp)
        lw      $a0,28($fp)     #reorder
        addu    $v0,$v1,$v0     #forwarding
        lbu     $v1,0($v0)      # assign it to $v1
        lw      $v0,16($fp)     # load pString and its pIndex
        nop                     #load hazard
        addu    $v0,$a0,$v0     #forwarding
        lbu     $v0,0($v0)      # assign it to $v0
        nop                     #load hazard
        bne     $v1,$v0,.L4     # if (myString[mIndex] != pString[pIndex])
        nop                     #Branch hazard

        lw      $v0,20($fp)     # mIndex increase 1 unit # load it from $fp
        nop                     #load hazard
        addiu   $v0,$v0,1       # increase, forwarding
        sw      $v0,20($fp)     # store back to $fp, forwarding
        lw      $v0,16($fp)     # the same with pIndex
        nop                     #load hazard
        addiu   $v0,$v0,1       #forwarding
        sw      $v0,16($fp)
        b       .L3
        nop                     #branch hazard

# .L4 = else
.L4:
        lw      $v0,16($fp)     # load from $fp pIndex
        nop                     #load hazard
        addiu   $v0,$v0,1       # then increase it to 1 unit, forwarding
        sw      $v0,16($fp)     # store it again back to $fp
        b       .L5
        nop                   #branch hazard

# .L3 = while (myString[mIndex] != '\0')
.L3:
        lw      $v0,20($fp)     # mIndex
        lw      $v1,32($fp)     # myString
        nop                     #load hazard
        addu    $v1,$v1,$v0     # increase myString to myString + mIndex
        lbu     $v1,0($v1)
        nop                     #load hazard
```

```
            bne     $v1,$t1,.L6       # if myString character not equals to zero
            nop                       #Branch hazard

    # .L5 = if (mIndex == strlen(myString)) found = pIndex - mIndex;
    .L5:
            lw      $s0,20($fp)       # mIndex
            lw      $a0,32($fp)       # myString

            bne     $s0,$v0,.L7       #compare mIndex ($s0) and strlen(myString)
($v0)
            nop                       #Branch hazard

            lw      $v1,16($fp)       # load pIndex
            lw      $v0,20($fp)       # load mIndex
            nop                       #load hazard
            subu    $v0,$v1,$v0       # pIndex - mIndex, forwarding
            sw      $v0,24($fp)       # store to found variable address

    # .L7 = mIndex = 0
    .L7:
            sw      $0,20($fp)

    # .L2 = while (pString[index] != '\0')
    .L2:
            lw      $v0,16($fp)               # pIndex
            lw      $v1,28($fp)               # pString
            nop                       #load hazard
            addu    $v0,$v1,$v0       # increase pString to pString + index, forwarding
            lbu     $v0,0($v0)
            nop                       #load hazard
            bne     $v0,$0,.L3        # if our pString character not equals to zero
            nop                       #Branch hazard

            # print found variable
            # lui   $v0,%hi(.LC1)
            # addiu         $a0,$v0,%lo(.LC1)
            lw      $a1,24($fp)
            nop                       #load hazard
            la      $a0, ($a1)
            li      $v0, 1
            syscall

            # restore memmoryx
            move    $v0,$0
            move    $sp,$fp
```

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

4. Xử lí hazard

```
.data
        strIn: .space 100
        newline: .ascii "\n"
.text

main:
        # move stack pointer to 56-bit lower
        # first store value of $ra to $sp + 52 ($ra = $fp + 56)
        # then store value of $fp
        # then store value of $s0
        addiu   $sp,$sp,-56
        sw      $v1,52($sp)
        sw      $fp,48($sp)
        sw      $s0,44($sp)
        # now $fp get the address of $sp
        # then store value of $a0, $a1
        move    $fp,$sp
        sw      $a0,56($fp)
        sw      $a1,60($fp)
        la      $t1, newline

        # create 1000-bit in $a0
        la      $a0, strIn
        lbu     $t1, 0($t1)                     #reorder
        li      $a1,1000                        # 0x3e8
        li      $v0, 8
        syscall

        # after getting input from screen, store it to $fp + 28
        sw      $a0,28($fp)
        # get another string
        la      $a0, strIn
        li      $a2,1000                        # 0x3e8
        li      $v0, 8                  #reorder
        addi    $a0, $a0, 100

        syscall

        # do the same as above
        sw      $a0, 32($fp)
        lw      $a1,28($fp) # pString
        lw      $a2,32($fp) # myString

        sw      $0,16($fp) # pIndex = 0
        sw      $0,20($fp) # mIndex = 0
        sw      $0,24($fp) # found = 0
        b       .L2

.L8:
        b       .L3

.L6:
        lw      $v0,20($fp) # load myString and its mIndex
        lw      $v1,32($fp)
        lw      $a0,28($fp) #reorder
        addu    $v0,$v1,$v0
        lbu     $v1,0($v0) # assign it to $v1
        lw      $v0,16($fp) # load pString and its pIndex

        addu    $v0,$a0,$v0
        lbu     $v0,0($v0) # assign it to $v0
        bne     $v1,$v0,.L4 # if (myString[mIndex] != pString[pIndex])


        lw      $v0,20($fp) # mIndex increase 1 unit # load it from $fp
        addiu   $v0,$v0,1 # increase
        sw      $v0,20($fp) # store back to $fp
        lw      $v0,16($fp) # the same with pIndex
        addiu   $v0,$v0,1
        sw      $v0,16($fp)
        b       .L3

# .L4 = else
.L4:
        lw      $v0,16($fp) # load from $fp pIndex
        addiu   $v0,$v0,1 # then increase it to 1 unit
        sw      $v0,16($fp) # store it again back to $fp
        b       .L5


# .L3 = while (myString[mIndex] != '\0')
```

```
.L3:
        lw      $v0,20($fp) # mIndex
        lw      $v1,32($fp) # myString
        addu    $v1,$v1,$v0 # increase myString to myString + mIndex
        lbu     $v1,0($v1)
        bne     $v1,$t1,.L6 # if myString character not equals to zero

# .L5 = if (mIndex == strlen(myString)) found = pIndex - mIndex;
.L5:
        lw      $s0,20($fp) # mIndex
        lw      $a0,32($fp) # myString

        bne     $s0,$v0,.L7 # compare mIndex ($s0) and strlen(myString) ($v0)


        lw      $v1,16($fp) # load pIndex
        lw      $v0,20($fp) # load mIndex
        subu    $v0,$v1,$v0 # pIndex - mIndex
        sw      $v0,24($fp) # store to found variable address

# .L7 = mIndex = 0
.L7:
        sw      $0,20($fp)

# .L2 = while (pString[index] != '\0')
.L2:
        lw      $v0,16($fp) # pIndex
        lw      $v1,28($fp) # pString
        addu    $v0,$v1,$v0 # increase pString to pString + index
        lbu     $v0,0($v0)
        bne     $v0,$0,.L8 # if our pString character not equals to zero

        # print found variable
        # lui   $v0,%hi(.LC1)
        # addiu         $a0,$v0,%lo(.LC1)
        lw      $a1,24($fp)
        la      $a0, ($a1)
        li      $v0, 1
        syscall

        # restore memmoryx
        move    $v0,$0
        move    $sp,$fp
        lw      $v1,52($sp)
        lw      $fp,48($sp)
        lw      $s0,44($sp)
        #addiu $sp,$sp,56
```

5. Điều chỉnh cấu hình file để sử dụng MIPS MARS4_5

```
.data
        strIn: .space 1000
        newline: .ascii "\n"
.text

main:
        # move stack pointer to 56-bit lower
        # first store value of $ra to $sp + 52 ($ra = $fp + 56)
        # then store value of $fp
        # then store value of $s0
        addiu   $sp,$sp,-56
        sw      $v1,52($sp)
        sw      $s0,44($sp)
        # now $fp get the address of $sp
        # then store value of $a0, $a1
        # move          $fp,$sp
        sw      $a0,56($sp)
        sw      $a1,60($sp)
        la      $t1, newline

        # create 1000-bit in $a0
        la      $a0, strIn
        lbu     $t1, 0($t1)
        li      $a1,1000                # 0x3e8
        li      $v0, 8
        syscall

        # after getting input from screen, store it to $fp + 28
        sw      $a0,28($sp)
        # get another string
        la      $a0, strIn
        li      $a2,1000
        li      $v0, 8
        addi    $a0, $a0, 1000                  # 0x3e8
        syscall

        # do the same as above
        sw      $a0, 32($sp)
        lw      $a1,28($sp)           # pString
        lw      $a2,32($sp)           # myString

        sw      $0,16($sp)            # pIndex = 0
        sw      $0,20($sp)            # mIndex = 0
        sw      $0,24($sp)            # found = 0
        b       .L2
        nop

.L6:
        lw      $v0,20($sp)            # load myString and its mIndex
        lw      $v1,32($sp)
        lw      $a0,28($sp)`
        addu    $v0,$v1,$v0

        lbu     $v1,0($v0)            # assign it to $v1

        lw      $v0,16($sp)            # load pString and its pIndex
        nop
        addu    $v0,$a0,$v0

        lbu     $v0,0($v0)            # assign it to $v0
        nop
        bne     $v1,$v0,.L4           # if (myString[mIndex] != pString[pIndex])
        nop

        lw      $v0,20($sp)           # mIndex increase 1 unit # load it from $fp
        nop
        addiu   $v0,$v0,1             # increase

        sw      $v0,20($sp)            # store back to $sp
        lw      $v0,16($sp)            # the same with pIndex
        nop
        addiu   $v0,$v0,1

        sw      $v0,16($sp)
        b       .L3
        nop

# .L4 = else
.L4:
        lw      $v0,16($sp)           # load from $sp pIndex
        nop
        addiu   $v0,$v0,1             # then increase it to 1 unit

        sw      $v0,16($sp)         # store it again back to $sp
        b       .L5
        nop

# .L3 = while (myString[mIndex] != '\0')
.L3:
        lw      $v0,20($sp)         # mIndex
        lw      $v1,32($sp)         # myString
        nop
        addu    $v1,$v1,$v0         # increase myString to myString + mIndex

        lbu     $v1,0($v1)
        nop
        bne     $v1,$t1,.L6         # if myString character not equals to zero
        nop

# .L5 = if (mIndex == strlen(myString)) found = pIndex - mIndex;
.L5:
        lw      $s0,20($sp)         # mIndex
        lw      $a0,32($sp)         # myString

        bne     $s0,$v0,.L7         # compare mIndex ($s0) and strlen(myString) ($v0)
        nop
        lw      $v0,20($sp)         # load mIndex
        lw      $v1,16($sp)         # load pIndex
        nop
        subu    $v0,$v1,$v0         # pIndex - mIndex

        sw      $v0,24($sp)         # store to found variable address

# .L7 = mIndex = 0
.L7:
        sw      $0,20($sp)

# .L2 = while (pString[index] != '\0')
.L2:
        lw      $v0,16($sp)         # pIndex
        lw      $v1,28($sp)         # pString
        nop
        addu    $v0,$v1,$v0         # increase pString to pString + index

        lbu     $v0,0($v0)
        nop
        bne     $v0,$0,.L3      # if our pString character not equals to zero
        nop

        # print found variable
        lw      $a1,24($sp)
        li      $v0, 1
        la      $a0, ($a1)

        syscall

        # restore memmory
        move    $v0,$0
        lw      $v1,52($sp)
        lw      $s0,44($sp)
```