HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY
---***---

# ASSEMBLY LANGUAGE AND COMPUTER ARCHITECTURE LAB
# FINAL PROJECT REPORT

**IT3280E**

**Instructor:** MSc.Le Ba Vui

**GROUP 11**

| Student's full name | Student ID | Problem No. |
|---|---|---|
| Vu Nguyen Hao | 20226037 | 6 |
| Hoang Trung Hieu | 20226039 | 7 |

Hanoi, 2024

# Table of contents:

# Problem 6: Memory allocation malloc():

## 1. Source code:

```
.data
CharPtr: .word  0 # Pointer variable, type of asciiz
CharPtr1: .word  0 # Pointer variable, type of asciiz
CharPtr2: .word 0 # Pointer variable, type of asciiz
BytePtr: .word  0 # Pointer variable, type of Byte
WordPtr: .word  0 # Pointer variable, type of Word
ArrayPtr: .word 0 # Pointer variable, point to Array A

#--------------Text display------------------
CharPtrDis: .asciiz "CharPtr\n"
BytePtrDis: .asciiz "BytePtr\n"
WordPtrDis: .asciiz "WordPtr\n"
Enterdata: .asciiz "Enter your data \n"
ValueChar: .asciiz "The value of Char pointer: "
ValueByte: .asciiz "The value of Byte pointer: "
ValueWord: .asciiz "The value of Word pointer: "
AddrChar: .asciiz "Address of Char pointer: "
AddrByte: .asciiz "Address of Byte pointer: "
AddrWord: .asciiz "Address of Word pointer: "
StrLen: .asciiz "Enter the length of the source string: "
InputStr: .asciiz "Enter the source string: "
CpyStr: .asciiz "The copied string: "
Allocated: .asciiz "Memory Allocated: "
InputRow: .asciiz "\nA[i][j]\nEnter the number of rows: "
InputCol: .asciiz "Enter the number of columns j: "
SetAij: .asciiz "\nSet A[i][j]\n"
GetAij: .asciiz "\nGet A[i][j]\n"
Inputi: .asciiz "Enter i: "
Inputj: .asciiz "Enter j: "
Continue: .asciiz "Continue?"
Result: .asciiz "A[i][j] = "
Mem_Allocated: .asciiz "The amount of allocated memory (byte): "
#--------------------------------------------

.kdata
Sys_TheTopOfFree: .word  1
Sys_MyFreeSpace:
.text
```

```
#Initiate memory to dynamic allocation

        jal     SysInitMem
        li      $s7,0
#-------Task 1: Fix the Word address error-------
#$a1: number of elements
#$a2: size of each elements

        la $a0,CharPtrDis
        li $v0,4
        syscall
        la      $a0, CharPtr    #Allocate CharPtr
        addi    $a1, $zero, 3
        addi    $a2, $zero, 4   #set to 4 instead of 1
        jal     malloc
        jal     InputData

        la $a0,BytePtrDis
        li $v0,4
        syscall
        la      $a0, BytePtr    #Allocate ByePtr
        addi    $a1, $zero, 2
        addi    $a2, $zero, 4   #set to 4 instead of 1
        jal     malloc
        jal     InputData

        la $a0,WordPtrDis
        li $v0,4
        syscall
        la      $a0, WordPtr    #Allocate WordPtr
        addi    $a1, $zero, 1
        addi    $a2, $zero, 4
        jal     malloc
        jal     InputData
        j       PtrValue
#-----------------------------------------------
SysInitMem:
        la      $t9, Sys_TheTopOfFree
        la      $t7, Sys_MyFreeSpace
        sw      $t7, 0($t9)
        jr      $ra

malloc:
        la      $t9, Sys_TheTopOfFree
        lw      $t8, 0($t9)     #Get the address of the free memory
        sw      $t8, 0($a0)     #Store it in the pointer
        addi    $v0, $t8, 0      #Which is also the return value
        mul     $t7, $a1,$a2     #Calculate the size of allocation
        add     $t6, $t8, $t7    #Update the address of free memory
        sw      $t6, 0($t9)      #Save to Sys_TheTopOfFree
        jr  $ra

InputData:
        li      $s3, 0  #counter
        la      $a0, Enterdata
        li      $v0, 4
        syscall
        Loopl:
        li      $v0, 5
        syscall
        sw      $v0, 0($t8)      #save input data into free memory
        addi    $t8, $t8, 4      #move to the next free memory
        addi    $s3, $s3, 1      #count++
        beq     $s3, $a1, Loop1_end  #count = number of elements then end
        j       Loopl
        Loop1_end:
        jr      $ra

#------Task 2: Get the value of the pointer------
PtrValue:
        la      $a0, CharPtr
        lw      $t8, 0($a0)
        lw      $t5, 0($t8)
        la      $a0, ValueChar
        li      $v0, 56
        move    $a1, $t5
        syscall

        la      $a0, BytePtr
        lw      $t8, 0($a0)
        lw      $t5, 0($t8)
        la      $a0, ValueByte
        li      $v0, 56
        move    $a1, $t5
        syscall

        la      $a0, WordPtr
        lw      $t8, 0($a0)
        lw      $t5, 0($t8)
        la      $a0, ValueWord
        li      $v0, 56
        move    $a1, $t5
        syscall
#---------------------------------------------

#------Task 3: Get the address of the pointer------
        la      $a0, CharPtr
        lw      $t8, 0($a0)
        la      $a0, AddrChar
        li      $v0, 56
        move    $a1, $t8
        syscall

        la      $a0, BytePtr
        lw      $t8, 0($a0)
        la      $a0, AddrByte
        li      $v0, 56
        move    $a1, $t8
        syscall

        la      $a0, WordPtr
        lw      $t8, 0($a0)
        la      $a0, AddrWord
        li      $v0, 56
        move    $a1, $t8
        syscall
#----------------------------------------------

#-----------Task 4: Copy 2 CharPtr-----------
        li $v0,4
        la $a0, StrLen
        syscall
        li $v0,5
        syscall         #Enter string length

        move $a1,$v0
        addi $a2,$0,1
        la $a0, CharPtr1        #source
        jal malloc
        move $s0,$v0  #Address of the source pointer

        la $a1, CharPtr2        #target
        jal malloc
        move $s1,$v0  #Address of the target pointer

        la $a0,InputStr
        li $v0,4
        syscall
        move $a0,$s0  #$a0 =  address source string
        li $v0,8
        syscall #Input string into the source pointer
        move $a1,$s1  #$a1 = address target string

strcpy:
        add $t0,$a0,$0#Initiate $t0 first character of source string
        add $t1,$a1,$0#Initiate $t1 first character of target string
        addi $t2,$0,1   #Initiate $t2 as a character != '\0' to start the loop

cpyLoop:
        beq $t2,0,cpyLoop_end         #End the loop if the latest copied char is '\0'
        lb $t2, 0($t0)
        sb $t2, 0($t1)
        addi $t0,$t0,1
        addi $t1,$t1,1
        j cpyLoop
cpyLoop_end:
        la $a0,CpyStr
        li $v0,4
        syscall
        move $a0,$s1
        syscall
#-------------------------------------------

#------Task 6: Calculate allocated memory-----
        la $t0, Sys_MyFreeSpace       # Lay dia chi dau tien duoc cap phat
        la $t1, Sys_TheTopOfFree      # Lay dia chi luu dia chi dau tien con trong
        lw $t2, 0($t1)             # Lay dia chi dau tien con trong
        sub $t3, $t2, $t0             # Tru hai dia chi cho nhau

        la $a0,Allocated
        li $v0,4
        syscall
        move $a0,$t3
        li $v0,1
        syscall
#----------------------------------------

#----------Task 5: Free the allocated-----
        jal SysInitMem
        nop
#----------------------------------------
#------------Task 7: Malloc2-------------
        jal   SysInitMem
        la   $a0, ArrayPtr
        #ham nhap so dong i
        la       $a0, InputRow
        li       $v0, 4
        syscall
        li       $v0, 5
        syscall
        move     $a1, $v0         #so dong

        #Ham nhap so cot j
        la       $a0, InputCol
        li       $v0, 4
        syscall
        li       $v0, 5
        syscall
        move     $a2, $v0         #so cot
        addi     $a3, $zero, 4
        jal malloc2
        j        Task8

        malloc2:
        la       $t9, Sys_TheTopOfFree
        lw       $t8, 0($t9)      #Lay dia chi dau tien con trong
        sw       $t8, 0($a0)      #Cat dia chi do vao bien con tro
        addi     $v0, $t8, 0      #Dong thoi laket qua tra ve cua ham
        mul      $t7, $a1,$a2     #Tinh kich thuoc cua mang can cap phat
        mul      $t5, $t7, $a3
        add      $t6, $t8, $t5    #Tinh dia chi dau tien controng
        sw       $t6, 0($t9)      #Luu tro lai dia chi dau tien do vao bien Sys_TheTopOfFree
        jr   $ra

#--------Task 8: Get/Set A[i][j]---------
Task8:
        #Ham set
        NhapDulieu1:
        la       $t8, Sys_MyFreeSpace
        la       $a0, SetAij
        li       $v0, 4
        syscall
        #Nhap hang i
        la       $a0, Inputi
        li       $v0, 4
        syscall
        laptimi:
        li       $v0, 5
        syscall
```

```
slt     $s4, $v0, $a1
bne     $s4, $0, hetlaptimi
j       laptimi
hetlaptimi:
move    $s1, $v0        #hang i luu vao s1

#Nhap cot j
la      $a0, Inputj
li      $v0, 4
syscall
laptimj:
li      $v0, 5
syscall
slt     $s4, $v0, $a2
bne     $s4, $0, hetlaptimj
j       laptimj
hetlaptimj:
move    $s2, $v0        #cot j luu vao s2

la      $a0, Result
li      $v0, 4
syscall
li      $v0, 5
syscall
mul     $s3, $s1, $a2
add     $s3, $s3, $s2
mul     $s3, $s3, $a3
add     $t8, $t8, $s3
sw      $v0, 0($t8)

la      $a0, Continue
li      $v0, 50
syscall
beq     $a0, $0, NhapDulieu1
j       XuatDulieu

XuatDulieu:   #ham get

la      $a0, GetAij
li      $v0, 4
syscall
la      $t8, Sys_MyFreeSpace
#Nhap hang i
la      $a0, Inputi
li      $v0, 4
syscall
```

```
        laptimi1:
        li      $v0, 5
        syscall
        slt     $s4, $v0, $a1
        bne     $s4, $0, hetlaptimi1
        j       laptimi1
        hetlaptimi1:
        move    $s1, $v0        #hang i luu vao s1

        #Nhap cot j
        la      $a0, Inputj
        li      $v0, 4
        syscall
        laptimj1:
        li      $v0, 5
        syscall
        slt     $s4, $v0, $a2
        bne     $s4, $0, hetlaptimj1
        j       laptimj1
        hetlaptimj1:
        move    $s2, $v0        #cot j luu vao s2

        la      $a0, Result
        li      $v0, 4
        syscall
        mul     $s3, $s1, $a2
        add     $s3, $s3, $s2
        mul     $s3, $s3, $a3
        add     $t8, $t8, $s3
        lw      $a0, 0($t8)
        li      $v0, 1
        syscall
        la      $a0, Continue
        li      $v0, 50
        syscall
        beq     $a0, $0, XuatDulieu
        j       exit
exit:
```

## 2. Ideas and algorithms:

Here are the basic ideas of the solutions for each tasks:
1)The word memory allocation has an error since the rule that the word address must be divisible by 4 is not guaranteed.

-> Change the size of each pointer elements ($a2) to 4 bytes

4)Write a function to copy 2 character pointers:
Copy the value of the index pointer of the source string to where the corresponding index pointer of the target string is pointing to. Repeat the process until the latest copied value is '\0'

5)Write a function to free the memory allocated to the pointers.
Set the first allocated address to the pointer of the first empty space's address.

6)Write a function to calculate the amount of allocated memory.
Take the first empty space's address and subtract the first allocated address.

7) Write a function malloc2 to allocate a 2-dimensional array of type .word with parameters including:

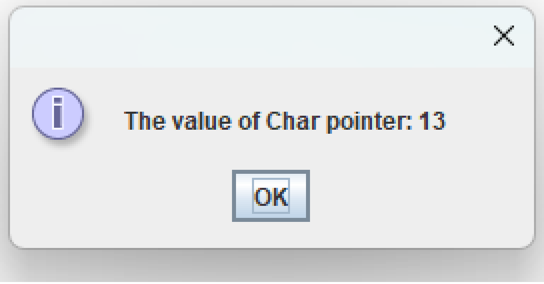
a.The starting address of the array
b.Number of rows
c.Number of columns

-> (Number of rows) x (Number of columns) x 4; then add it to the first empty space pointer. This will fully allocate memory for the array

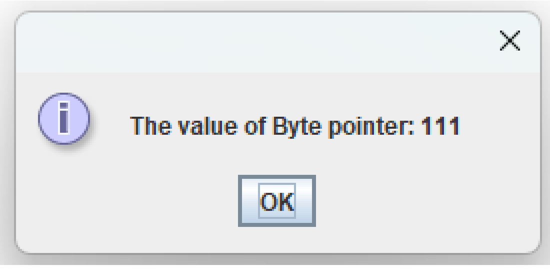
8)Based on question 7, write two functions getArray[i][j] and setArray[i][j] to get/set the value of the array in row i column j of the array.
-> Input i,j
-> Add (i x #number of columns + j) to the value of Sys_MyFreeSpace to set the value of the array in row i, column j

## 3. Simulation results:

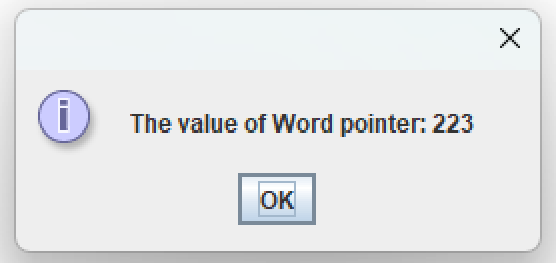
2)Get value of the pointer:



The value of Char pointer: 13

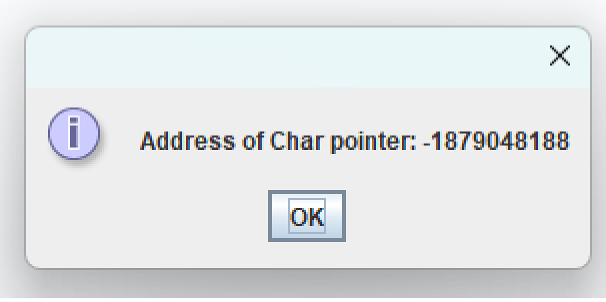


The value of Byte pointer: 111

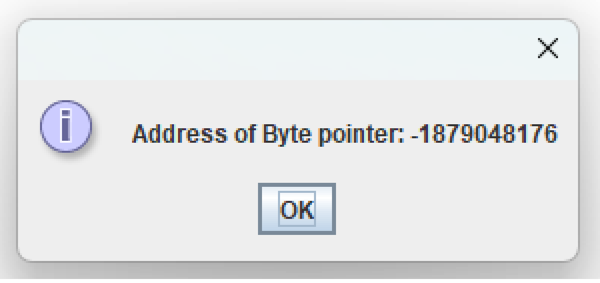


The value of Word pointer: 223

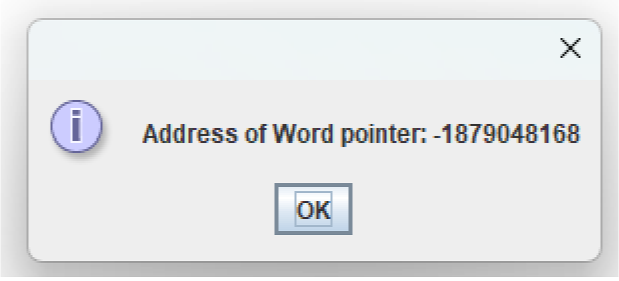
3)Get address of the pointer:



Address of Char pointer: -1879048188

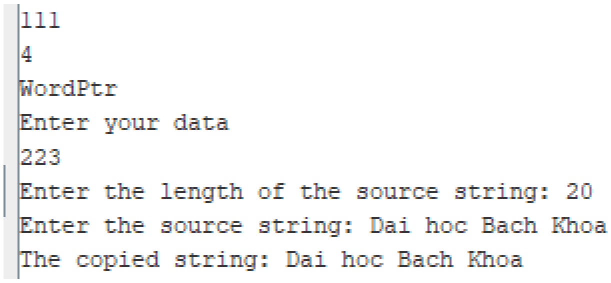


Address of Byte pointer: -1879048176



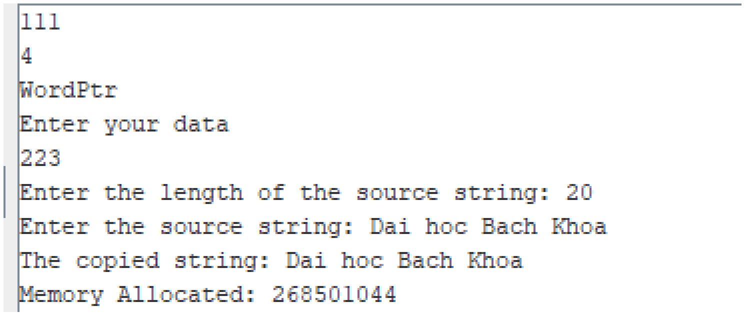
Address of Word pointer: -1879048168

4)Copy char Pointer:



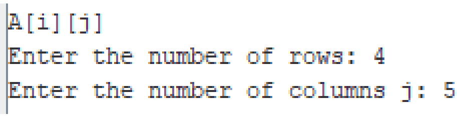
```
111
4
WordPtr
Enter your data
223
Enter the length of the source string: 20
Enter the source string: Dai hoc Bach Khoa
The copied string: Dai hoc Bach Khoa
```

6)Allocated memory:

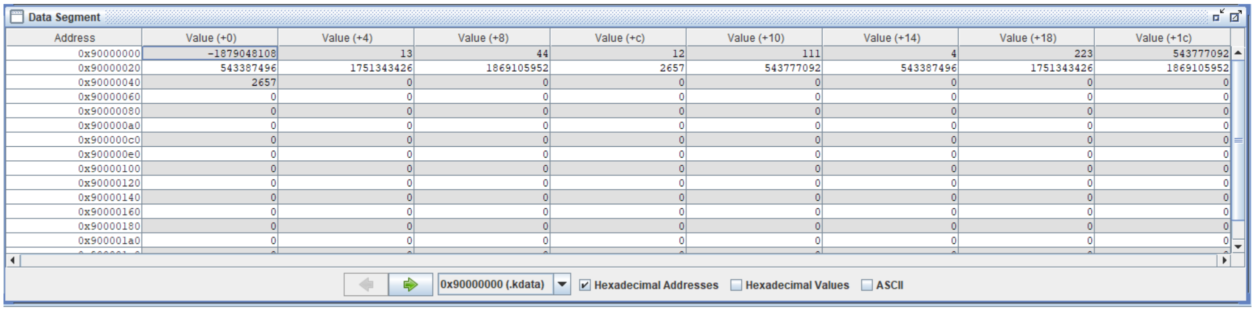


```
111
4
WordPtr
Enter your data
223
Enter the length of the source string: 20
Enter the source string: Dai hoc Bach Khoa
The copied string: Dai hoc Bach Khoa
Memory Allocated: 268501044
```

7)Malloc2 - 2-dimensional array memory allocation:



```
A[i][j]
Enter the number of rows: 4
Enter the number of columns j: 5
```



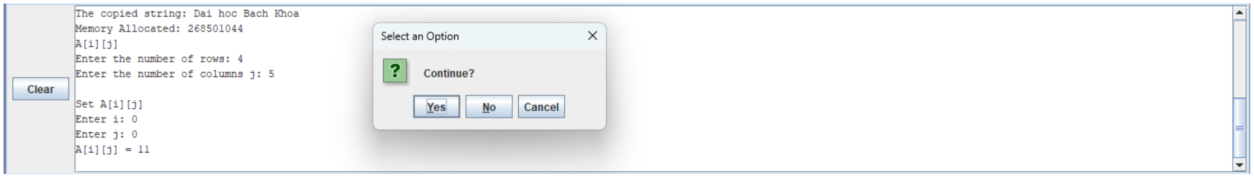
Data Segment

| Address | Value (+0) | Value (+4) | Value (+8) | Value (+c) | Value (+10) | Value (+14) | Value (+18) | Value (+1c) |
|---|---|---|---|---|---|---|---|---|
| 0x90000000 | -1879048108 | 13 | 44 | 12 | 111 | 4 | 223 | 543777092 |
| 0x90000020 | 543387496 | 1751343426 | 1869105952 | 2657 | 543777092 | 543387496 | 1751343426 | 1869105952 |
| 0x90000040 | 2657 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x90000060 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x90000080 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x900000a0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x900000c0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x900000e0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x90000100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x90000120 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x90000140 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x90000160 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x90000180 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0x900001a0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

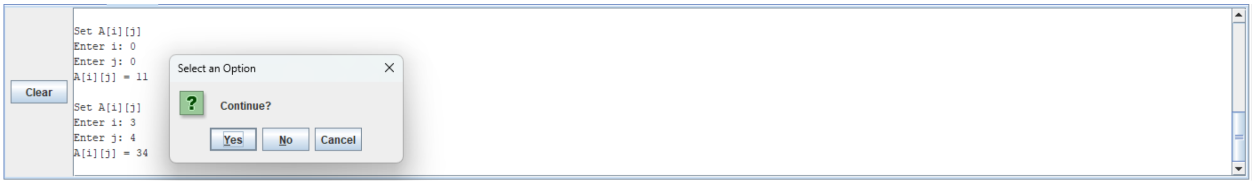
0x90000000 (.kdata) ☑ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

8)Get/Set A[i][j]

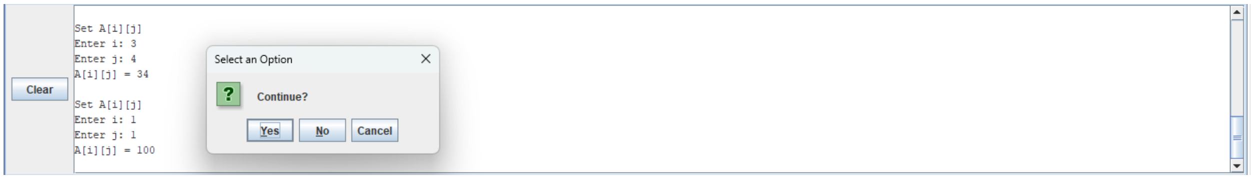
-Set A[i][j]



```
The copied string: Dai hoc Bach Khoa
Memory Allocated: 268501044
A[i][j]
Enter the number of rows: 4
Enter the number of columns j: 5

Set A[i][j]
Enter i: 0
Enter j: 0
A[i][j] = 11
```

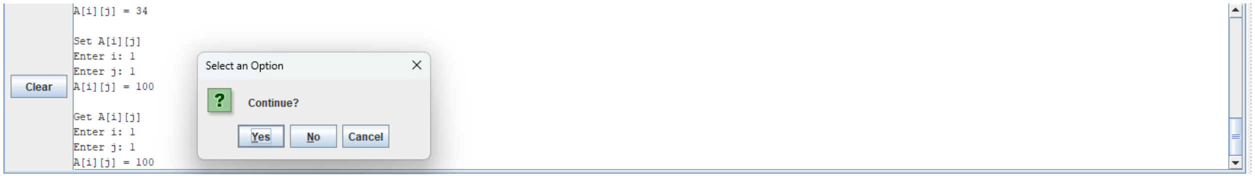


```
Set A[i][j]
Enter i: 0
Enter j: 0
A[i][j] = 11

Set A[i][j]
Enter i: 3
Enter j: 4
A[i][j] = 34
```

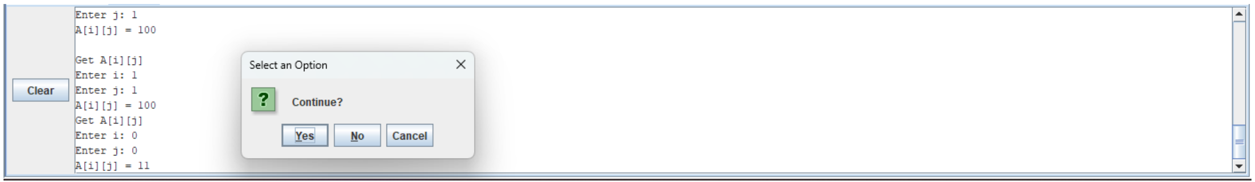


```
Set A[i][j]
Enter i: 3
Enter j: 4
A[i][j] = 34

Set A[i][j]
Enter i: 1
Enter j: 1
A[i][j] = 100
```

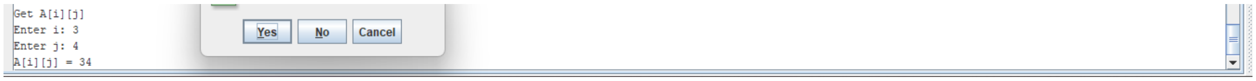
-Get A[i][j]:



```
A[i][j] = 34

Set A[i][j]
Enter i: 1
Enter j: 1
A[i][j] = 100

Get A[i][j]
Enter i: 1
Enter j: 1
A[i][j] = 100
```



```
Enter j: 1
A[i][j] = 100

Get A[i][j]
Enter i: 1
Enter j: 1
A[i][j] = 100
Get A[i][j]
Enter i: 0
Enter j: 0
A[i][j] = 11
```



```
Get A[i][j]
Enter i: 3
Enter j: 4
A[i][j] = 34
```

# Problem 7: Checking the syntax of MIPS instruction:

## 1. Source code

```
# Hoang Trung Hieu 20226039
# Subject 7: check MIPS instruction syntax

.data
# ---------------------------------------------------------------------------- #
# Opcode library                                                               #
# Rule: each opcode has length of 8 byte, separated by type and syntax         #
# ---------------------------------------------------------------------------- #

# Opcode Library:
opcodeLibrary:   .asciiz "add,1   sub,1   addu,1  subu,1  mul,1   and,1   or,1    nor,1   xor,1   slt,1   addi,2  addiu,2 andi,2  ori,2   sll,2   srl,2   slti,2  sltiu,2 mult,3  div,3   move,3  lw,4    sw,4    lb,4    sb,4    lbu,4   lhu,4   ll,4    sh,4    lui,5   li,5    la,6    mfhi,7  mflo,7  jr,7    beq,8   bne,8   j,9     jal,9   "


buffer:        .space        100
opcode:        .space        10
```

```
# Mess
Message:                    .asciiz      "Enter string: "
correct_opcode_prompt:      .asciiz      "\nCorrect opcode: "
end_prompt:                 .asciiz      "\nCorrect syntax."
not_valid_register_prompt:  .asciiz      "\nInvalid register syntax."
not_valid_number_prompt:    .asciiz      "\nNot valid number."
not_valid_address_prompt:   .asciiz      "\nNot valid address"
valid_syntax_prompt:        .asciiz      "\nCorrect MIPS syntax."
continue_prompt:            .asciiz      "\nContinue? (1. Yes 0. No): "
missing_prompt:             .asciiz      "\nMissing operand"

# Syntax error mess:
missing_comma_prompt: .asciiz "\nSyntax error: missing colon."
invalid_opcode_prompt: .asciiz "\nOpcode is invalid or doesn't exist."
too_many_variable_prompt: .asciiz "\nSyntax has too many variables."


# Registers library #
# each register has 8 bytes in registLibrary
registerLibrary: .asciiz "$zero   $at     $v0     $v1     $a0     $a1     $a2     $a3     $t0     $t1     $t2
$t3     $t4     $t5     $t6     $t7     $s0     $s1     $s2     $s3     $s4     $s5     $s6     $s7     $t8     $t9
$k0     $k1     $gp     $sp     $fp     $ra     $0      $1      $2      $3      $4      $5      $6      $7      $8
$9      $10     $11     $12     $13     $14     $15     $16     $17     $18     $19     $20     $21     $22
$21     $22     $23     $24     $25     $26     $27     $28     $29     $30     $31     $0      "

# $s0 is the address of input string
# $a0 is used for traversing input string
# $s1 is the address of opcode
# $a1 is used for traversing opcode
# $s2 is the address of opcodeLibrary
# $a2 is used for traversing opcodeLibrary
# $s3 is the address of registerLibrary
# $a3 is used for traversing registerLibrary


.text
main:
        la $a0, Message                         # print Message
        li $v0, 4
        syscall

read_data:
        li $v0, 8
        la $a0, buffer
        li $a1, 100
        syscall
        move $s0, $a0                              # store address of input string into $s0

        jal clear_whitespace                                   # jump to clear_whitespace function

read_opcode:
        la $a1, opcode                             # $a1 is used for incrementing opcode
character position
        la $s1, opcode                        # $s1 = address of opcode
loop_read_opcode:
        lb $t1, 0($a0)                        # $t1 = current character in opcode
        beq $t1, ' ', check_opcode                 # if a whitespace is found then check
        beq $t1, '\n', missing_                    # if a newline character is found then the
string is missing operands
        sb $t1, 0($a1)                        # store current character into opcode
        addi $a0, $a0, 1                           # continue checking next character
        addi $a1, $a1, 1                           # increment current address of opcode
        j loop_read_opcode


# ------------------------------------------------------------------------ #
check_opcode:
        move $a1, $s1                              # $a1 = $s1 = address of opcode
        move $s0, $a0                              # $s0 points to the character after opcode

        la $s2, opcodeLibrary                      # $s2 = address of opcodeLibrary
        jal check

        j invalid_opcode                           # jump to target
# ------------------------------------------------------------------------ #




# ------------------------------------------------------------------------ #
check:
        move $a2, $s2                              # a2 pointer to beginning of library
loop_check:
        lb $t2, 0($a2)                             # load each character from library
        beq $t2, ',', evaluation1                  # if encountered colon, evaluate whether it is
correct
        lb $t1, 0($a1)                             # load each character from  input opcode
        beq $t2, 0, jump_                          # if current character in the opcodeLibrary is
\0 then we have checked all possible opcodes in the Library -> no valid input opcode
        bne $t1, $t2, next_opcode                  # character mismatch
        addi $a1, $a1, 1                           # next character
        addi $a2, $a2, 1
        j loop_check


evaluation1:
        lb $t1, 0($a1)                                  # load current character of opcode
        beq $t1, 0, opcode_done                         # if current character of opcode is
null then it has matched an opcode in opcodeLibrary
        j next_opcode                                   # else continue checking opcode in
opcodeLibrary

next_opcode:
        addi $s2, $s2, 8                           # increment $s2 by 8 because each opcode
has 8 bytes in opcodeLibrary
        move $a2, $s2                                   # update $a2
        move $a1, $s1                                   # reset running index of opcode
        j loop_check                               # continue looping to check for next opcode
# ------------------------------------------------------------------------------ #



# ------------------------------------------------------------------------------ #
opcode_done:
        jal         correct_opcode                      # print correct opcode

        addi          $a2, $a2, 1
        lb          $t2, 0($a2)                 # Load syntax type in $t2

        jal         clear_whitespace               # point $s0 to next valid character after
opcode


        addi          $t2, $t2, -48             # Minus value of $t2 by 48 to get the interger value

        beq           $t2, 1, Type_1
        beq           $t2, 2, Type_2
        beq           $t2, 3, Type_3
        beq           $t2, 4, Type_4
        beq           $t2, 5, Type_5
        beq           $t2, 6, Type_6
        beq           $t2, 7, Type_7
        beq           $t2, 8, Type_8
        beq           $t2, 9, Type_9

end:
        j           ending                         # jump to ending
```

```
# ------------------------------------------------------------------------- #
```

```
# clear whitespace until the first valid character
# ------------------------------------------------------------------------- #
clear_whitespace:
	move		$a0, $s0			# load $a0 as the address of input string
	lb		$t1, 0($a0)			# read first character
	beq		$t1, ' ', loop_whitespace		# if first character is a whitespace then delete
	beq		$t1, 9, loop_whitespace			# if first character is a tab then delete
	jr		$ra				# in this case first character is neither a whitespace or a tab so we jump back
loop_whitespace:
	lb		$t1, 0($a0)			# read current character
	beq		$t1, ' ', whitespace_found		# if this character is a whitespace then increment address of input string by 1
	beq		$t1, 9, whitespace_found		# if this character is a tab then increment address of input string by 1
	move		$s0, $a0			# there is no more invalid character here so update the address
	jr		$ra				# then jump back
whitespace_found:
	addi		$a0, $a0, 1			# increment address of input string by 1 to delete invalid character
	j		loop_whitespace				# continue looping
# ------------------------------------------------------------------------- #
```

```
# check if current character is a comma
# ------------------------------------------------------------------------- #
check_comma:

	move		$a0, $s0			# update $a0 = $s0
	lb		$t1, 0($a0)			# get the current character
	bne		$t1, ',', missing_comma			# if current character is != comma then invalid syntax
	jr		$ra
# ------------------------------------------------------------------------- #
# clear gap in instruction and check for comma
# ------------------------------------------------------------------------------ #
check_gap:

        addi $sp, $sp, 4
        sw $ra, 0($sp)                                          # store $ra

        jal clear_whitespace
        jal check_comma
        addi $a0, $a0, 1                        # Point to character/whitespace after colon
        move $s0, $a0
        jal clear_whitespace

        lw $ra, 0($sp)
        addi $sp, $sp, -4                               # restore $ra

        jr $ra
# ------------------------------------------------------------------------------ #



jump_:
        jr      $ra


# All types of instructions
# ------------------------------------------------------------------------------ #
OPCODE_TYPES:
Type_1:
# ------------------------------------------------------------------------------ #
#       Format: xyz $1, $2, $3                                          #
# ------------------------------------------------------------------------------ #
        jal reg_check

        jal check_gap

        jal reg_check

        jal check_gap

        jal reg_check
        jal check_end

Type_2:
# ---------------------------------------------------------------------------- #
#       Format: xyz $1, $2, 10000                                    #
# ---------------------------------------------------------------------------- #
        jal reg_check

        jal check_gap

        jal reg_check

        jal check_gap

        jal num_check

        jal check_end

Type_3:
# ---------------------------------------------------------------------------- #
#       Format: mult $2,$3                                       #
# ---------------------------------------------------------------------------- #
        jal reg_check

        jal check_gap

        jal reg_check

        jal check_end

Type_4:
# ---------------------------------------------------------------------------- #
#       Format: lw $1, 100($2)                                     #
# ---------------------------------------------------------------------------- #
        jal reg_check

        jal check_gap

        jal address_check

        jal check_end

Type_5:
# ---------------------------------------------------------------------------- #
#       Format: lui $1, 100                                      #
# ---------------------------------------------------------------------------- #
```

```
        jal reg_check

        jal check_gap

        jal num_check

        jal check_end

Type_6:
# ----------------------------------------------------------------------------- #
#       Format: la $1, label                                         #
# ----------------------------------------------------------------------------- #
        jal reg_check

        jal check_gap

        jal label_check

        beq $s7, 1, check_end                                   # case label is character and syntax is
correct

        jal num_check                                           # case label is numerical value

        jal check_end

Type_7:
# ----------------------------------------------------------------------------- #
#       Format mfhi $2                                               #
# ----------------------------------------------------------------------------- #
        jal reg_check

        jal check_end

Type_8:
# ----------------------------------------------------------------------------- #
#       Format: beq $1, $2, label or beq $1,$2,100                    #
# ----------------------------------------------------------------------------- #
        jal reg_check

        jal check_gap

        jal reg_check

        jal check_gap

        jal label_check
        beq $s7, 1, check_end                                   # case label is character and syntax is
correct

        jal num_check                                           # case label is numerical value

        jal     check_end

Type_9:
# --------------------------------------------------------------------------- #
#       Format j 1000 ; j label                                 #
# --------------------------------------------------------------------------- #
        jal     label_check

        beq     $s7, 1, check_end

        jal     num_check


        jal     check_end

# End of instruction types
# --------------------------------------------------------------------------- #



# All syntax checking functions:
# --------------------------------------------------------------------------- #


# check whether input string has ended or not
# --------------------------------------------------------------------------- #
check_end:
            jal clear_whitespace
        lb $t5, 0($s0)
        beq $t5, '\n', valid_syntax
        beq $t5, '\0', valid_syntax
        beq $t5,  '#', valid_syntax
        j too_many_variable                                     # not valid
# --------------------------------------------------------------------------- #




# Check whether string is register or not
# --------------------------------------------------------------------------- #
```

```
reg_check:
        la $s3, registerLibrary
        move $a3, $s3                           # a3 points to beginning of register library
        move $a0, $s0

loop_reg_check:
        lb $t3, 0($a3)                          # load each character from library
        lb $t0, 0($a0)                          # load each character from input string
        beq $t3, ' ', evaluation2               # if encountered space, evaluate whether it is
correct
        beq $t3, 0, not_valid_register                  # if encountered \0 then we have
checked every registers inside registerLibrary
        bne $t0, $t3, next_reg                  # character mismatch
        addi $a0, $a0, 1                        # next character
        addi $a3, $a3, 1
        j loop_reg_check
evaluation2:
        lb $t0, 0($a0)
        beq $t0, ',', found_reg             # Correct register
        beq $t0, ' ', found_reg             # Correct register
        beq $t0, 0, found_reg                   # Correct register
        beq $t0, '\n', found_reg                # Correct register
        j next_reg                                      # jump to next_register
next_reg:
        addi $s3, $s3, 8                        # Move to next register
        move $a3, $s3
        move $a0, $s0
        j loop_reg_check                        # check again
found_reg:
        move $s0, $a0                           # move pointer forward
        j jump_                                         # jump to jump_
# ------------------------------------------------------------------------------ #




# check whether current parameter is a valid number
# ------------------------------------------------------------------------------ #
num_check:
            move $a0, $s0

num_check_loop:
        lb $t0, 0($a0)
        beq $t0, ',', is_num                    # end of parameter
        beq $t0, ' ', is_num                    # end of parameter
        beq $t0, 0, is_num                      # end of parameter
        beq $t0, '\n', is_num                   # end of parameter
        bgt $t0, '9', not_num                   # if $t0 > '9' then not a number
        blt $t0, '0', not_num                   # if $t0 < '0' then not a number
        addi $a0, $a0, 1
        j num_check_loop                                # continue checking

is_num:
        move $s0, $a0
        j jump_                                         # jump back

not_num:
        j not_num_error                                 # jump to not_num_error

# ------------------------------------------------------------------------ #



# check whether address syntax is correct
# ------------------------------------------------------------------------ #
address_check:
adnum_check:

num_check_loop2:
          lb $t0, 0($a0)
        beq $t0, '(', is_num2                   # correct syntax for shift amount
        bgt $t0, '9', not_num2                  # if $t0 > 9 then not a valid number
        blt $t0, '0', not_num2                  # if $t0 < 0 then not a valid number
        addi $a0, $a0, 1
        j num_check_loop2                               # continue checking next character

is_num2:
        move $s0, $a0
        j adreg_check                                   # continue checking for second
register

not_num2:
        j not_valid_address
# ------------------------------------------------------------------------ #



# check whether register in address is correct
# ------------------------------------------------------------------------ #
adreg_check:
reg_check2:
           addi $a0, $a0, 1
        move $s0, $a0
        la $s3, registerLibrary
        move $a3, $s3                                   # a3 points to beginning of register library
        move $a0, $s0
loop_reg_check2:
        lb $t3, 0($a3)                                  # load each character from registerLibrary
        lb $t0, 0($a0)                                  # load each character from input string
        beq $t3, ' ', evaluation3                       # if encountered space, evaluate whether it is
correct
        beq $t3, 0, not_valid_address2                          # if encountered \0 then we have
checked all available registers in registerLibrary
        bne $t0, $t3, next_reg2                         # if current characters are different
        addi $a0, $a0, 1                                # continue checking next character
        addi $a3, $a3, 1
        j loop_reg_check2
evaluation3:
        lb $t0, 0($a0)
        beq $t0, ')', found_reg2                        # correct syntax
        j next_reg2                                             # else continue checking for next
register
next_reg2:
        addi $s3, $s3, 8                                # Move to next register in registerLibrary
        move $a3, $s3
        move $a0, $s0
        j loop_reg_check2                                       # continue checking
not_valid_address2:
        j not_valid_address
found_reg2:
        addi $a0, $a0, 1
        move $s0, $a0                                   # move pointer forward
        jr $ra                                                  # jump back
# ----------------------------------------------------------------------------- #



# check whether label syntax is correct (for characters)
# ----------------------------------------------------------------------------- #
# output: $s7 = 1 if it is character and syntax is correct
#         $s7 = 0 if it not character and to signal that input label could be in numerical values
# ----------------------------------------------------------------------------- #
label_check:
           move $a0, $s0
```

```
first_char_check: # Can't be number and can't be underscore:
        lb $t0, ($a0)                                   # get current character of input string
        blt $t0, 'a', not_lower                         # if less than 'a' then it is not lower case
character
        bgt $t0, 'z', not_lower                         # if greater than 'z' then it is not lower case
chracter

        j loop_label_check                                      # it's lower so we jump to 2nd
character

not_lower:
        blt $t0, 'A', fail_case                         # if less than 'A' then not alphabet
            bgt $t0, 'Z', fail_case                             # if greater than 'Z' then not alphabet

loop_label_check: # Can be alphabet, number and underscore

        addi $a0, $a0, 1                                # increment $a0 by 1 to get next character
        lb $t0, ($a0)                             # load current character of input string

        beq $t0, ' ', valid_label                       # if we are here then all preceeding
charactes are valid
        beq $t0, '\n', valid_label                      # if we are here then all preceeding
charactes are valid
        beq $t0, 0, valid_label                                 # if we are here then all preceeding
charactes are valid


        blt $t0, 'a', not_lower2                        # if less than a then it is not lower case
character
        bgt $t0, 'z', not_lower2                        # if greater than z then it is not lower case
character
        j loop_label_check                                      # else valid, continue to check for
next character

not_lower2:
        bne $t0, '_', not_underscore                    # if it is not underscore then continue
checking
        j loop_label_check                                      # else valid, continue to check for
next character

not_underscore:
        blt $t0, 'A', not_upper2                        # If less than 'A' then it is not alphabet
            bgt $t0, 'Z', not_upper2                            # If greater than 'Z' then it is not
alphabet
            j loop_label_check                                  # else valid, continue to check for
next character
```

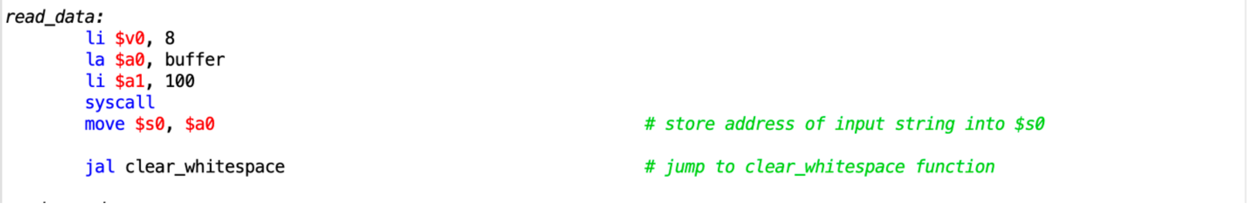
```
not_upper2:
      blt $t0, '0', fail_case                         # if less than 0 then it is not number either
         bgt $t0, '9', fail_case                              # if greater than 9 then it is not
number either, failcase
         j loop_label_check                                   # else valid, continue to check for
next character

fail_case:
      move $a0, $s0                                   # reset to before so we check other case (not
using label as address but numerical value instead)
      li $s7, 0                                       # set $s7 = 0 to signal to check for
numerical value
      jr $ra                                                  # jump back

valid_label:
      move $s0, $a0                                   # Move pointer forward
      li $s7, 1                                       # if label is all characters and correct then
set $s7 = 1
      jr $ra
# --------------------------------------------------------------------------- #


# End of syntax checking functions
# --------------------------------------------------------------------------- #




# print correct_opcode_prompt and input opcode
# --------------------------------------------------------------------------- #
correct_opcode:
         la $a0, correct_opcode_prompt
      li $v0, 4
      syscall
      la $a0, opcode
      li $v0, 4
      syscall
      move $a0, $s0                                   #  Return $a0
      jr $ra
# --------------------------------------------------------------------------- #
# All types of error messages when checking syntax:
# ---------------------------------------------------------------------------- #
missing_comma:
      la $a0, missing_comma_prompt
      li $v0, 4
      syscall
      j ending

invalid_opcode:
      la $a0, invalid_opcode_prompt
      li $v0, 4
      syscall
      j ending

too_many_variable:
      la $a0, too_many_variable_prompt
      li $v0, 4
      syscall
      j ending

not_valid_register:
      la $a0, not_valid_register_prompt
      li $v0, 4
      syscall
      j ending

not_num_error:
      la $a0, not_valid_number_prompt
      li $v0, 4
      syscall
      j ending

not_valid_address:
      la $a0, not_valid_address_prompt
      li $v0, 4
      syscall
      j ending

missing_:
      la $a0, missing_prompt
      li $v0, 4
      syscall
      j ending

# End of error types
# --------------------------------------------------------------------------- #



valid_syntax:
      la $a0, valid_syntax_prompt
      li $v0, 4
      syscall
      j ending

ending:
      la $a0, continue_prompt
      li $v0, 4
      syscall

      li $v0, 5
      syscall

      beq $v0, 1, resetAll_andContinue                 # if user choose to continue
      # else end program

      li $v0, 10
      syscall


resetAll_andContinue:

            li $v0, 0
            li $v1, 0
            jal clean_block                                     # jump to clean_block
      jal clean_opcode                          # jump to clean_block
      li $a0, 0
      li $a1, 0
            li $a2, 0
            li $a3, 0
            li $t0, 0
            li $t1, 0
            li $t2, 0
            li $t3, 0
            li $t4, 0
            li $t5, 0
            li $t6, 0
            li $t7, 0
            li $t8, 0
            li $t9, 0
            li $s0, 0
            li $s1, 0
            li $s2, 0
            li $s3, 0
            li $s4, 0
            li $s5, 0
            li $s6, 0
        li $s7, 0
            li $k0, 0
            li $k1, 0

            j main


# reset all values stored in previous input string to 0
# ---------------------------------------------------------------------------- #
clean_block:
        li $a0, 0
        li $a1, 0
        la $s0, buffer                                      # point $s0 to the address of buffer
loop_block:
        beq $a1, 100, jump_
        sb $a0, 0($s0)
        addi $s0, $s0, 1
        addi $a1, $a1, 1
        j loop_block
# ---------------------------------------------------------------------------- #



# reset all values stored in previous opcode to 0
# ---------------------------------------------------------------------------- #
clean_opcode:
        li $a0, 0
        li $a1, 0
        la $s1, opcode                                      # point $s1 to the address of opcode
loop_opcode:
        beq $a1, 10, jump_
        sb  $a0, 0($s1)
        addi $s1, $s1, 1
        addi $a1, $a1, 1
        j loop_opcode
# ---------------------------------------------------------------------------- #
```

# 2.Idea

-We create libraries that store possible opcodes and registers
-For opcodes library we need to store its syntax and also its type (depends on number of parameters, registers or label)

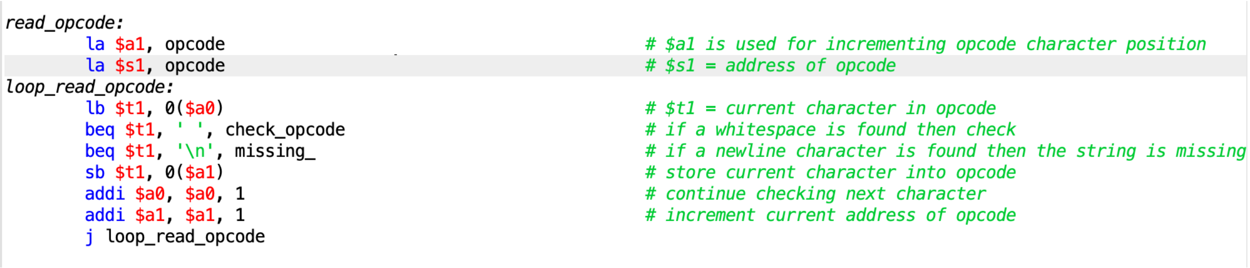
# 3. Flow of whole program:

-First read in input string, then we clear preceding whitespace or tab characters with clear_whitespace function.



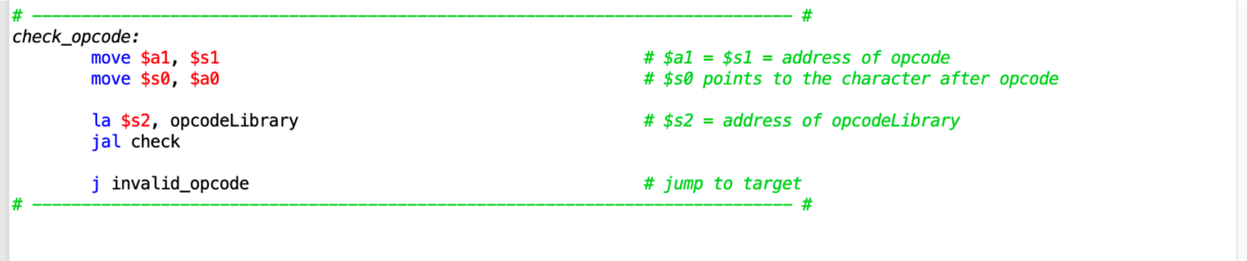
```
read_data:
        li $v0, 8
        la $a0, buffer
        li $a1, 100
        syscall
        move $s0, $a0                                           # store address of input string into $s0

        jal clear_whitespace                                    # jump to clear_whitespace function
```

-After that traverse the input string to find first newline charater or whitespace character and simultaneously store each character into ‘opcode’

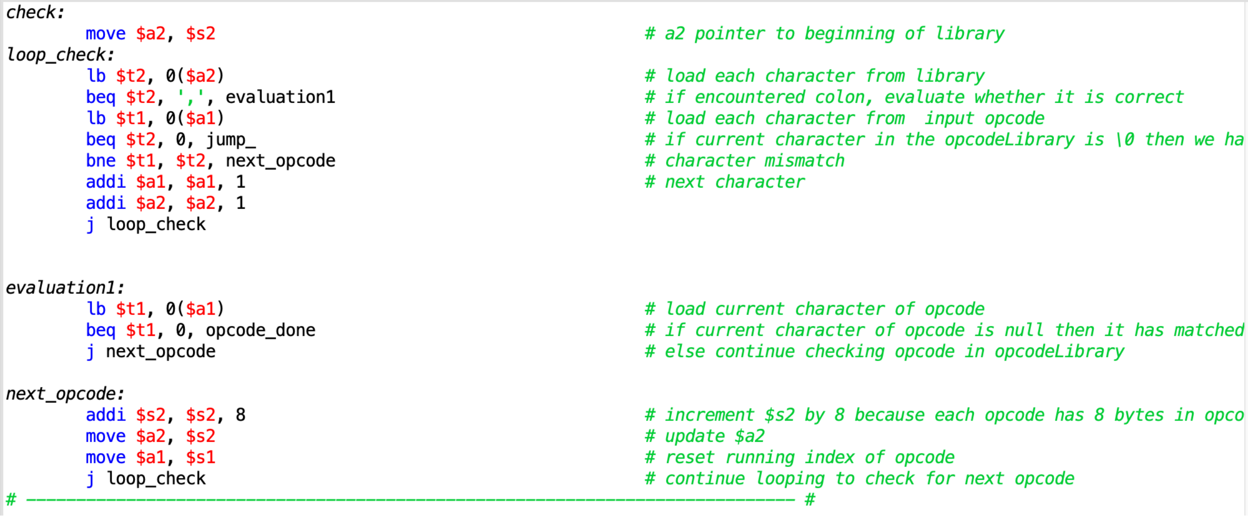


```
read_opcode:
        la $a1, opcode                                          # $a1 is used for incrementing opcode character position
        la $s1, opcode                                          # $s1 = address of opcode
loop_read_opcode:
        lb $t1, 0($a0)                                          # $t1 = current character in opcode
        beq $t1, ' ', check_opcode                              # if a whitespace is found then check
        beq $t1, '\n', missing_                                 # if a newline character is found then the string is missing
        sb $t1, 0($a1)                                          # store current character into opcode
        addi $a0, $a0, 1                                        # continue checking next character
        addi $a1, $a1, 1                                        # increment current address of opcode
        j loop_read_opcode
```

+ if it is ‘\n’ then user did not provide operands -> jump to missing operand
+ if it is whitespace then we check if the provided opcode existed in the opcodeLibrary with the check_opcode function

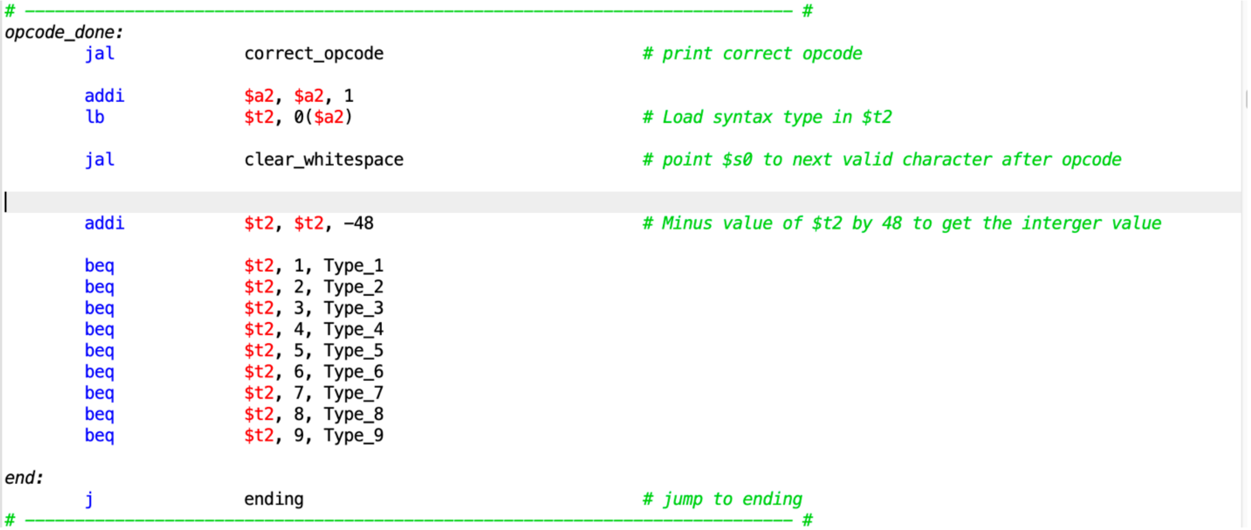


```
# ---------------------------------------------------------------------------- #
check_opcode:
        move $a1, $s1                                           # $a1 = $s1 = address of opcode
        move $s0, $a0                                           # $s0 points to the character after opcode

        la $s2, opcodeLibrary                                   # $s2 = address of opcodeLibrary
        jal check

        j invalid_opcode                                        # jump to target
# ---------------------------------------------------------------------------- #
```



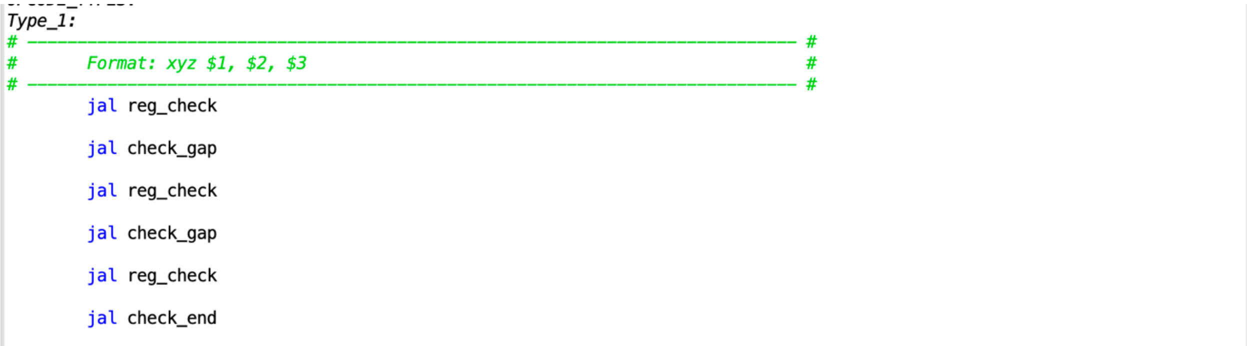
```
check:
        move $a2, $s2                                       # a2 pointer to beginning of library
loop_check:
        lb $t2, 0($a2)                                      # load each character from library
        beq $t2, ',', evaluation1                           # if encountered colon, evaluate whether it is correct
        lb $t1, 0($a1)                                      # load each character from  input opcode
        beq $t2, 0, jump_                                   # if current character in the opcodeLibrary is \0 then we ha
        bne $t1, $t2, next_opcode                           # character mismatch
        addi $a1, $a1, 1                                    # next character
        addi $a2, $a2, 1
        j loop_check


evaluation1:
        lb $t1, 0($a1)                                      # load current character of opcode
        beq $t1, 0, opcode_done                             # if current character of opcode is null then it has matched
        j next_opcode                                       # else continue checking opcode in opcodeLibrary

next_opcode:
        addi $s2, $s2, 8                                    # increment $s2 by 8 because each opcode has 8 bytes in opco
        move $a2, $s2                                       # update $a2
        move $a1, $s1                                       # reset running index of opcode
        j loop_check                                        # continue looping to check for next opcode
# ---------------------------------------------------------------------------- #
```

-If valid opcode then go to opcode_done, after clearing the whitespaces in the input string then we get the type of instruction, compare it with the 9 types and check for syntax of each type



```
# ---------------------------------------------------------------------------- #
opcode_done:
        jal             correct_opcode                      # print correct opcode

        addi            $a2, $a2, 1
        lb              $t2, 0($a2)                         # Load syntax type in $t2

        jal             clear_whitespace                    # point $s0 to next valid character after opcode


        addi            $t2, $t2, -48                       # Minus value of $t2 by 48 to get the interger value

        beq             $t2, 1, Type_1
        beq             $t2, 2, Type_2
        beq             $t2, 3, Type_3
        beq             $t2, 4, Type_4
        beq             $t2, 5, Type_5
        beq             $t2, 6, Type_6
        beq             $t2, 7, Type_7
        beq             $t2, 8, Type_8
        beq             $t2, 9, Type_9

end:
        j               ending                              # jump to ending
# ---------------------------------------------------------------------------- #
```

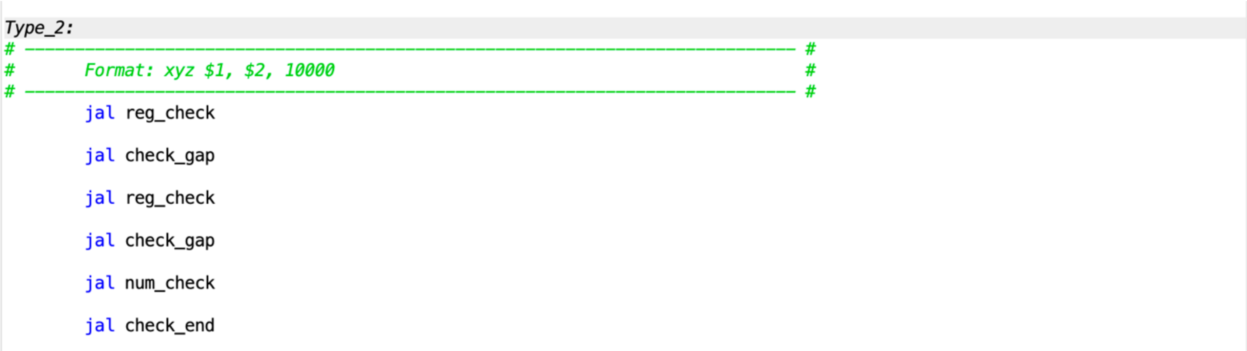
*) Type_1: consists of opcode and 3 registers



```
Type_1:
# ------------------------------------------------------------------------- #
#       Format: xyz $1, $2, $3                                              #
# ------------------------------------------------------------------------- #
        jal reg_check

        jal check_gap

        jal reg_check

        jal check_gap

        jal reg_check

        jal check_end
```

-call reg_check to check if first parameter is register
-call check_gap to get the second parameter's first character, then check if it is register with reg_check
-call check_gap to get the third parameter's first character, then reg_check
-then call check_end to check if string ends here or not

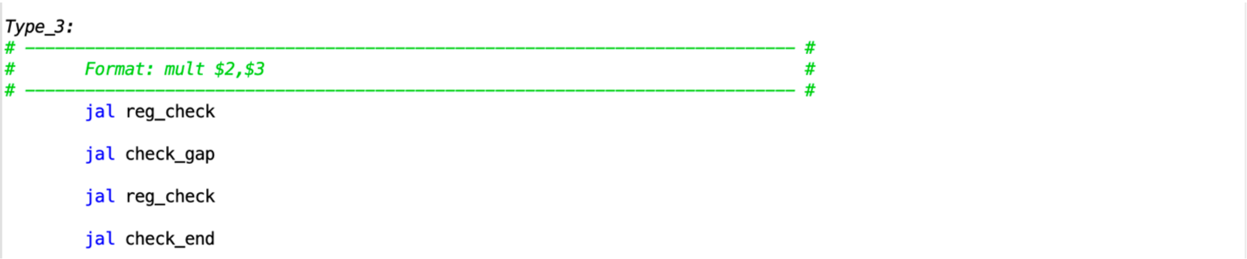
*) Type_2: consists of opcode, 2 registers and a number (immediate)



```
Type_2:
# ------------------------------------------------------------------------- #
#       Format: xyz $1, $2, 10000                                           #
# ------------------------------------------------------------------------- #
        jal reg_check

        jal check_gap

        jal reg_check

        jal check_gap

        jal num_check

        jal check_end
```

-call reg_check to check if first parameter is register
-call check_gap to get the second parameter's first character, then check if it is register with reg_check
-call check_gap to get the third parameter's first character, then num_check to check if it is a valid number
-then call check_end to check if string ends here or not

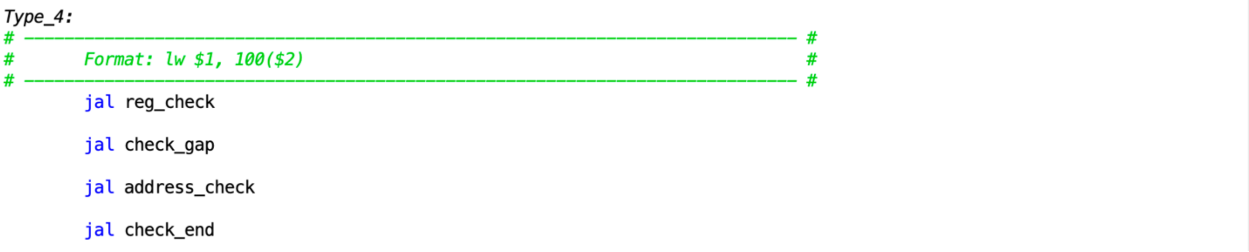
*) Type 3: consists of opcode, 2 registers



```
Type_3:
# ------------------------------------------------------------------------------ #
#       Format: mult $2,$3                                                       #
# ------------------------------------------------------------------------------ #
        jal reg_check

        jal check_gap

        jal reg_check

        jal check_end
```

-call reg_check to check if first parameter is register
-call check_gap to get the second parameter's first character, then check if it is register with reg_check
-then call check_end to check if string ends here or not

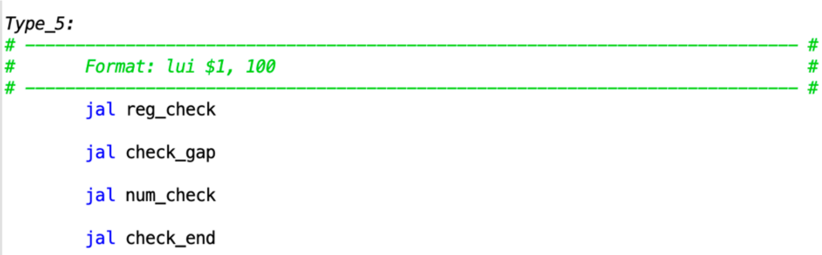
*) Type_4: consists of opcode, register, shift amount and address in the second register



```
Type_4:
# ------------------------------------------------------------------------------ #
#       Format: lw $1, 100($2)                                                   #
# ------------------------------------------------------------------------------ #
        jal reg_check

        jal check_gap

        jal address_check

        jal check_end
```

-call reg_check to check if first parameter is register
-call check_gap to get the second parameter's first character
-then call address_check to check if the address syntax is correct
-then call check_end to check if string ends here or not

*) Type_5: consists of opcode, 1 register and a number (immediate)

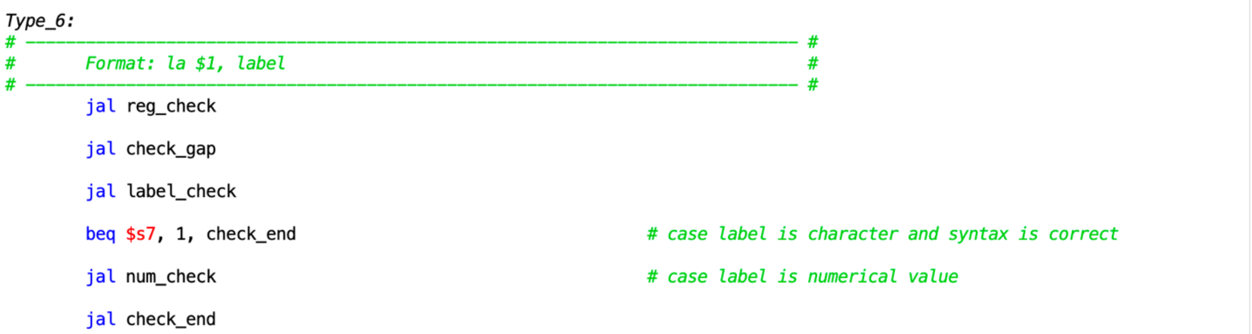


```
Type_5:
# ------------------------------------------------------------------------------ #
#       Format: lui $1, 100                                                      #
# ------------------------------------------------------------------------------ #
        jal reg_check

        jal check_gap

        jal num_check

        jal check_end
```

-call reg_check to check if first parameter is register

-call check_gap to get the second parameter's first character, then call num_check to check if it is a valid number
-then call check_end to check if string ends here or not

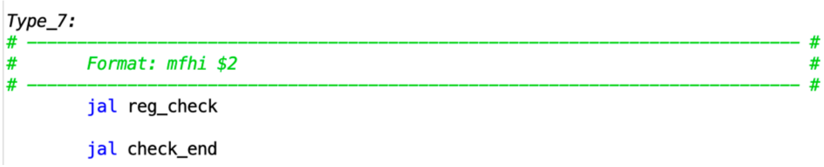
*) Type_6: consists of opcode, 1 register and a label



```
Type_6:
# ----------------------------------------------------------------------------- #
#       Format: la $1, label                                                    #
# ----------------------------------------------------------------------------- #
        jal reg_check

        jal check_gap

        jal label_check

        beq $s7, 1, check_end                    # case label is character and syntax is correct

        jal num_check                            # case label is numerical value

        jal check_end
```

-call reg_check to check if first parameter is register
-call check_gap to get the second parameter's first character, then call label_check to check if the label syntax is correct, if it is not then continue to check label if is is in numerical value
-then call check_end to check if string ends here or not

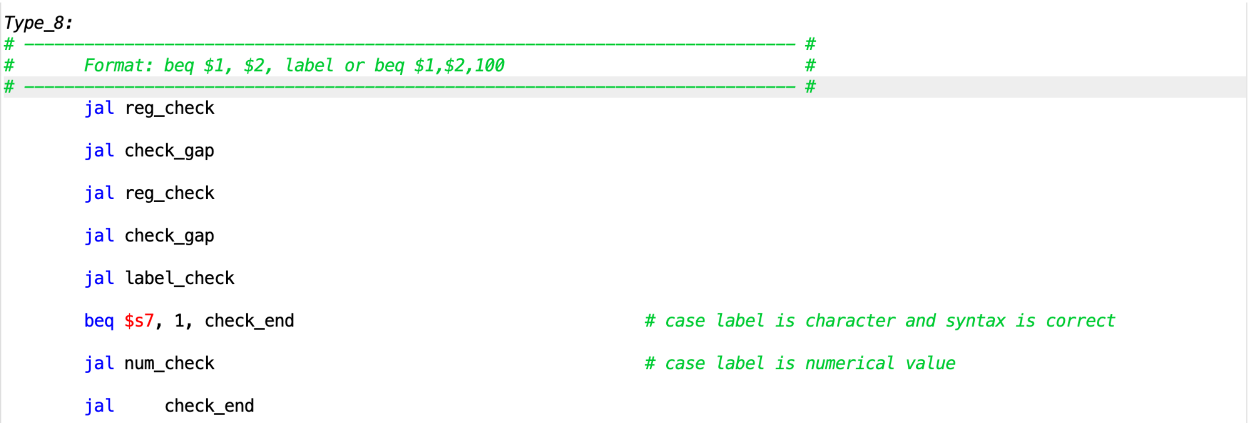
*) Type_7: consists of opcode and 1 register



```
Type_7:
# ----------------------------------------------------------------------------- #
#       Format: mfhi $2                                                         #
# ----------------------------------------------------------------------------- #
        jal reg_check

        jal check_end
```

-call reg_check to check if first parameter is register
-then call check_end to check if string ends here or not

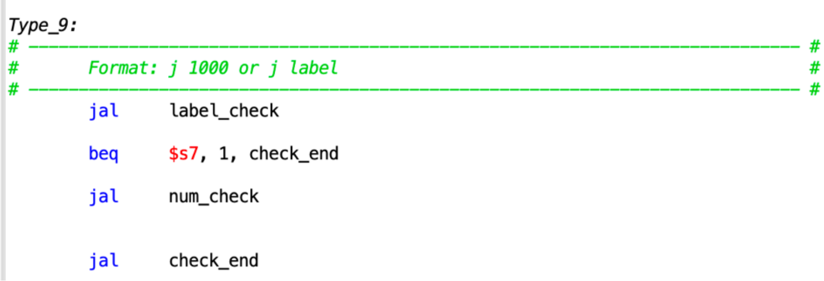
*) Type_8: consists of opcode, 2 registers and a label



```
Type_8:
# ------------------------------------------------------------------------------ #
#       Format: beq $1, $2, label or beq $1,$2,100                              #
# ------------------------------------------------------------------------------ #
        jal reg_check

        jal check_gap

        jal reg_check

        jal check_gap

        jal label_check

        beq $s7, 1, check_end                           # case label is character and syntax is correct

        jal num_check                                   # case label is numerical value

        jal     check_end
```

-call reg_check to check if first parameter is register
-call check_gap to get the second parameter’s first character, then check if it is register with reg_check
-then call check_gap to get the third parameter’s first character, then check if the label is correct in syntax with label_check, if it is not then continue to check if it is in numerical value
-then call check_end to check if string ends here or not

*) Type_9: consists of opcode and a label



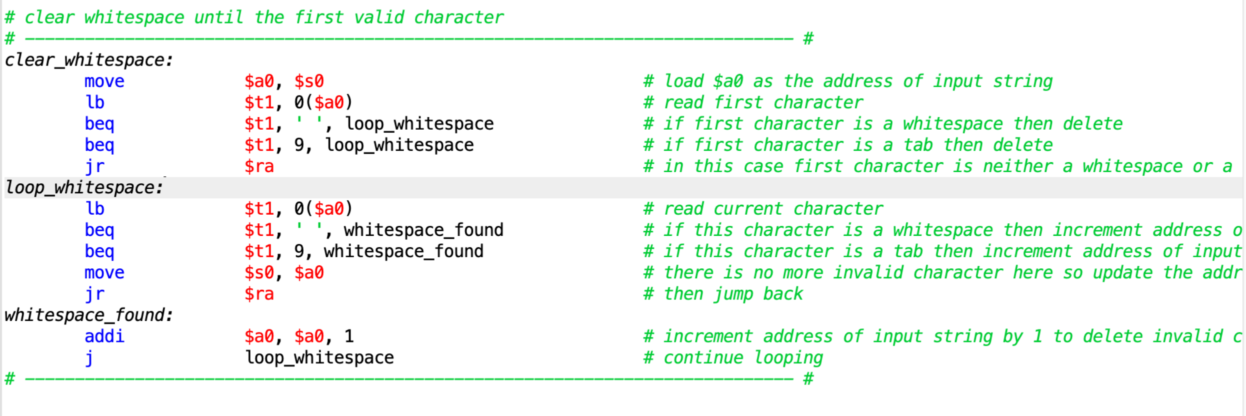
```
Type_9:
# ------------------------------------------------------------------------------ #
#       Format: j 1000 or j label                                               #
# ------------------------------------------------------------------------------ #
        jal     label_check

        beq     $s7, 1, check_end

        jal     num_check


        jal     check_end
```

-call label_check to check if the label is correct in syntax, if it is not then continue to check if it is in numerical value
-then call check_end to check if string ends here or not

After type check

## 4. Functions and subprocesses

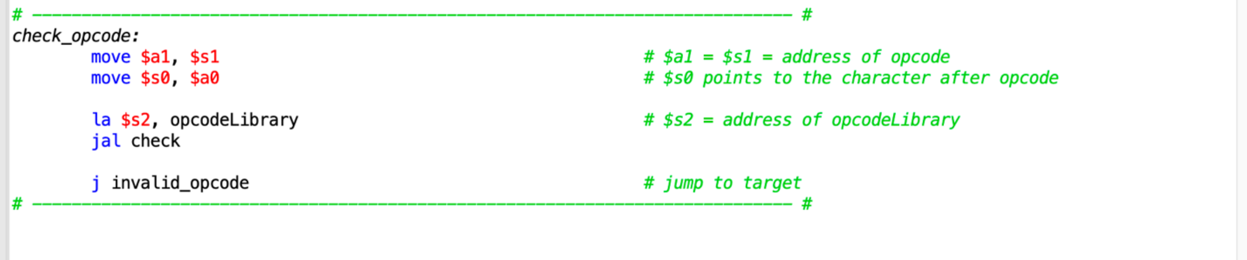
### 4.1. clear_whitespace:



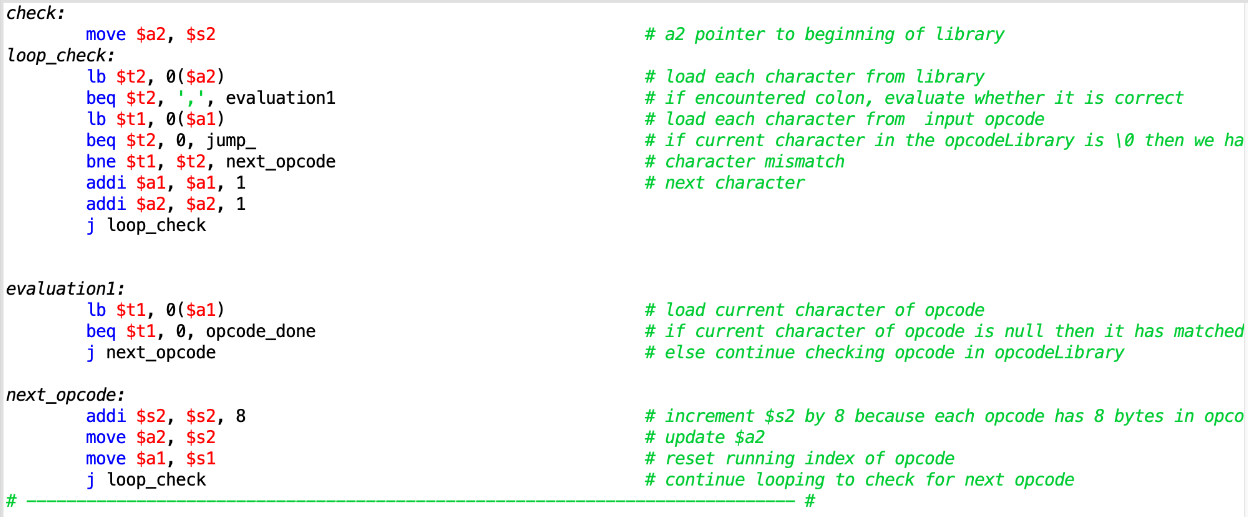
```
# clear whitespace until the first valid character
# ---------------------------------------------------------------------------- #
clear_whitespace:
        move            $a0, $s0                                # load $a0 as the address of input string
        lb              $t1, 0($a0)                             # read first character
        beq             $t1, ' ', loop_whitespace               # if first character is a whitespace then delete
        beq             $t1, 9, loop_whitespace                 # if first character is a tab then delete
        jr              $ra                                     # in this case first character is neither a whitespace or a
loop_whitespace:
        lb              $t1, 0($a0)                             # read current character
        beq             $t1, ' ', whitespace_found              # if this character is a whitespace then increment address o
        beq             $t1, 9, whitespace_found                # if this character is a tab then increment address of input
        move            $s0, $a0                                # there is no more invalid character here so update the addr
        jr              $ra                                     # then jump back
whitespace_found:
        addi            $a0, $a0, 1                             # increment address of input string by 1 to delete invalid c
        j               loop_whitespace                         # continue looping
# ---------------------------------------------------------------------------- #
```

-check if first character in input string is whitespace or tab character
-if it is then loop to clear all invalid characters
-if not then done function
-in the loop if current character is different from whitespace or tab then done
-after clear_whitespace $s0 will point to the next character that is different from whitespace
After opcode

4.2. check_opcode:



```
# ---------------------------------------------------------------------------- #
check_opcode:
        move $a1, $s1                                           # $a1 = $s1 = address of opcode
        move $s0, $a0                                           # $s0 points to the character after opcode

        la $s2, opcodeLibrary                                   # $s2 = address of opcodeLibrary
        jal check

        j invalid_opcode                                        # jump to target
# ---------------------------------------------------------------------------- #
```



```
check:
        move $a2, $s2                                           # a2 pointer to beginning of library
loop_check:
        lb $t2, 0($a2)                                          # load each character from library
        beq $t2, ',', evaluation1                               # if encountered colon, evaluate whether it is correct
        lb $t1, 0($a1)                                          # load each character from  input opcode
        beq $t2, 0, jump_                                       # if current character in the opcodeLibrary is \0 then we ha
        bne $t1, $t2, next_opcode                               # character mismatch
        addi $a1, $a1, 1                                        # next character
        addi $a2, $a2, 1
        j loop_check


evaluation1:
        lb $t1, 0($a1)                                          # load current character of opcode
        beq $t1, 0, opcode_done                                 # if current character of opcode is null then it has matched
        j next_opcode                                           # else continue checking opcode in opcodeLibrary

next_opcode:
        addi $s2, $s2, 8                                        # increment $s2 by 8 because each opcode has 8 bytes in opco
        move $a2, $s2                                           # update $a2
        move $a1, $s1                                           # reset running index of opcode
        j loop_check                                            # continue looping to check for next opcode
# ---------------------------------------------------------------------------- #
```

-check input opcode string with existing available opcodes in opcodeLibrary

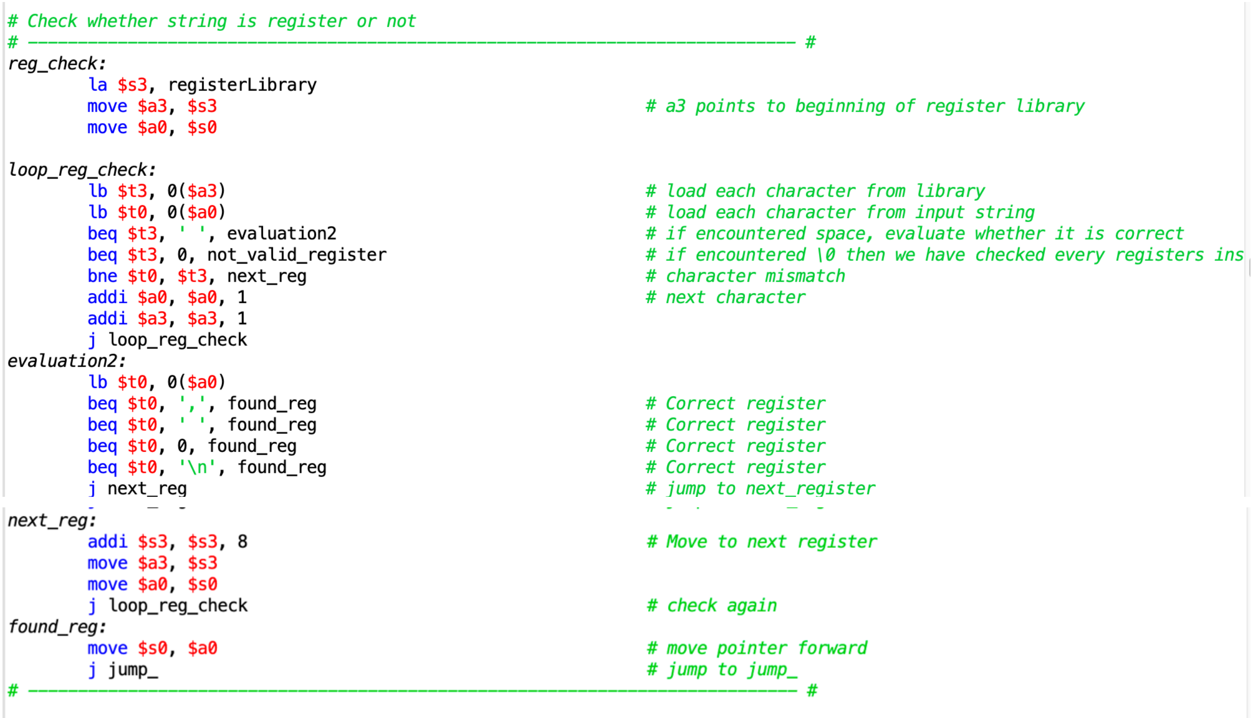
-we do this by loading running address $a2 in opcodeLibrary and $a1 is running address of input opcode

-with each iteration we check if value of $a2 is equal to $a1, if not then check for next opcode in opcodeLibrary by incrementing $s2 with 8 and update $a2 = $s2

-if current value of $a2 is `,` then check if current value of $a1 is 0 (end of opcode input) or not, value of $a1 = 0 then opcode found -> jump to opcode_done, else not found, continue checking next opcode

-if current value of $a2 is equal to 0 then we have checked all characters in opcodeLibrary -> opcode not found, jump back to where we call check process -> jump to invalid_opcode

### 4.3. reg_check:



```
# Check whether string is register or not
# ---------------------------------------------------------------------------- #
reg_check:
        la $s3, registerLibrary
        move $a3, $s3                                # a3 points to beginning of register library
        move $a0, $s0

loop_reg_check:
        lb $t3, 0($a3)                               # load each character from library
        lb $t0, 0($a0)                               # load each character from input string
        beq $t3, ' ', evaluation2                    # if encountered space, evaluate whether it is correct
        beq $t3, 0, not_valid_register               # if encountered \0 then we have checked every registers ins
        bne $t0, $t3, next_reg                       # character mismatch
        addi $a0, $a0, 1                             # next character
        addi $a3, $a3, 1
        j loop_reg_check
evaluation2:
        lb $t0, 0($a0)
        beq $t0, ',', found_reg                      # Correct register
        beq $t0, ' ', found_reg                      # Correct register
        beq $t0, 0, found_reg                        # Correct register
        beq $t0, '\n', found_reg                     # Correct register
        j next_reg                                   # jump to next_register
next_reg:
        addi $s3, $s3, 8                             # Move to next register
        move $a3, $s3
        move $a0, $s0
        j loop_reg_check                             # check again
found_reg:
        move $s0, $a0                                # move pointer forward
        j jump_                                      # jump to jump_
# ---------------------------------------------------------------------------- #
```

-each time reg_check is called it's going to check if the next input parameter is register or not with a loop

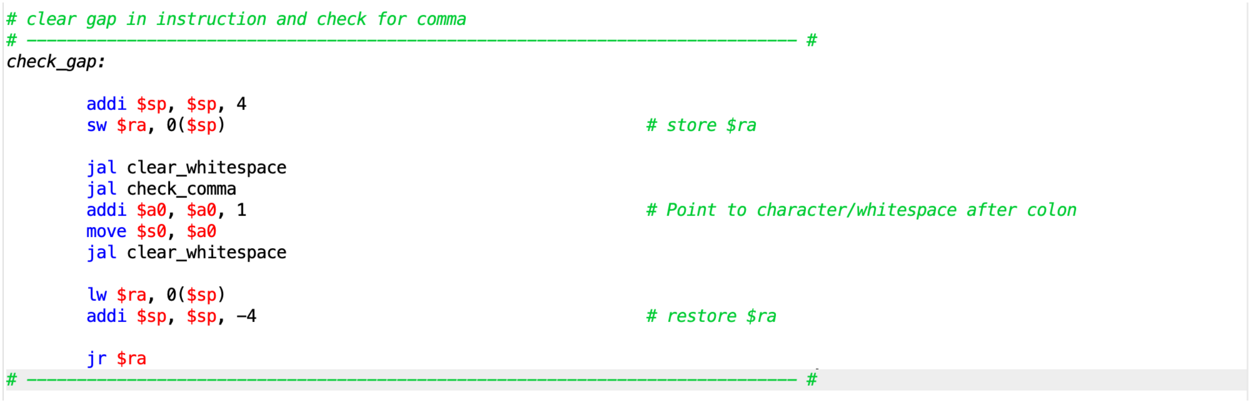
-inside the loop we load each character from registerLibrary and store in $t3, load each character from input string and store in $t0

-if $t3 is a whitespace then evaluate $t0, if $t0 is a comma or a whitespace then register found

-if $t3 != $t0 then check the next registers in registerLibrary

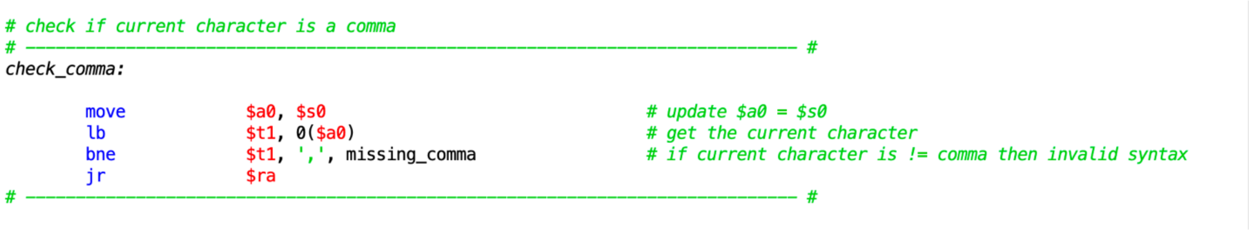
-after reg_check $s0 will be updated to to next character after the current parameter

### 4.4. check_gap:



```
# clear gap in instruction and check for comma
# ---------------------------------------------------------------------------- #
check_gap:

        addi $sp, $sp, 4
        sw $ra, 0($sp)                                  # store $ra

        jal clear_whitespace
        jal check_comma
        addi $a0, $a0, 1                                # Point to character/whitespace after colon
        move $s0, $a0
        jal clear_whitespace

        lw $ra, 0($sp)
        addi $sp, $sp, -4                               # restore $ra

        jr $ra
# ---------------------------------------------------------------------------- #
```

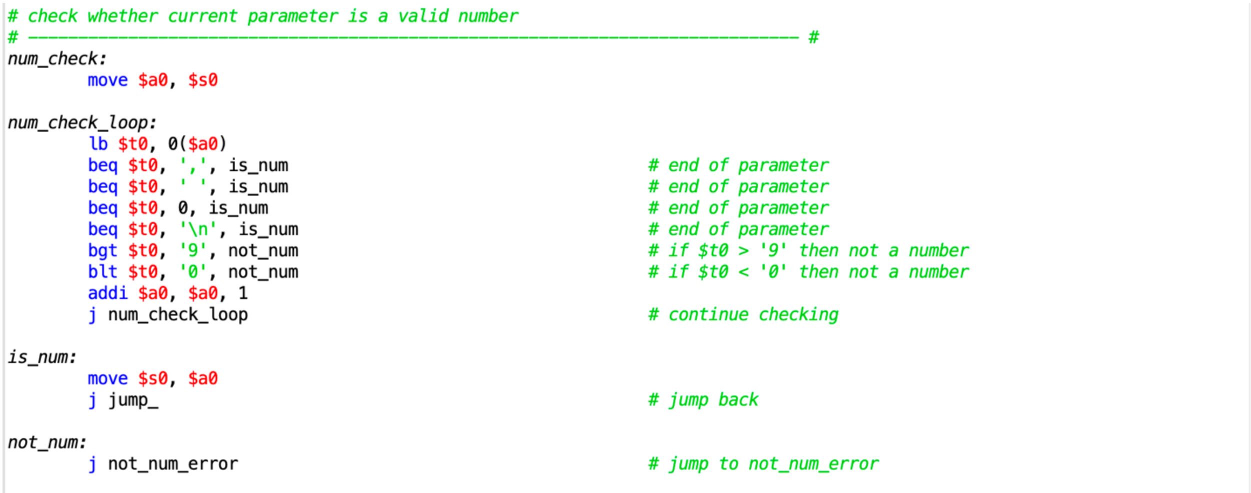
-first store $ra
-then call clear_whitespace to get the first valid character that is different from whitespace
-after this call check_comma, if the next character is not comma then the syntax is wrong



```
# check if current character is a comma
# ---------------------------------------------------------------------------- #
check_comma:

        move            $a0, $s0                        # update $a0 = $s0
        lb              $t1, 0($a0)                     # get the current character
        bne             $t1, ',', missing_comma         # if current character is != comma then invalid syntax
        jr              $ra
# ---------------------------------------------------------------------------- #
```

-then increment $s0 and $a0 by 1 and call clear_whitespace again
-after check_gap $s0 should be pointing to the first character of next parameter
-restore $ra

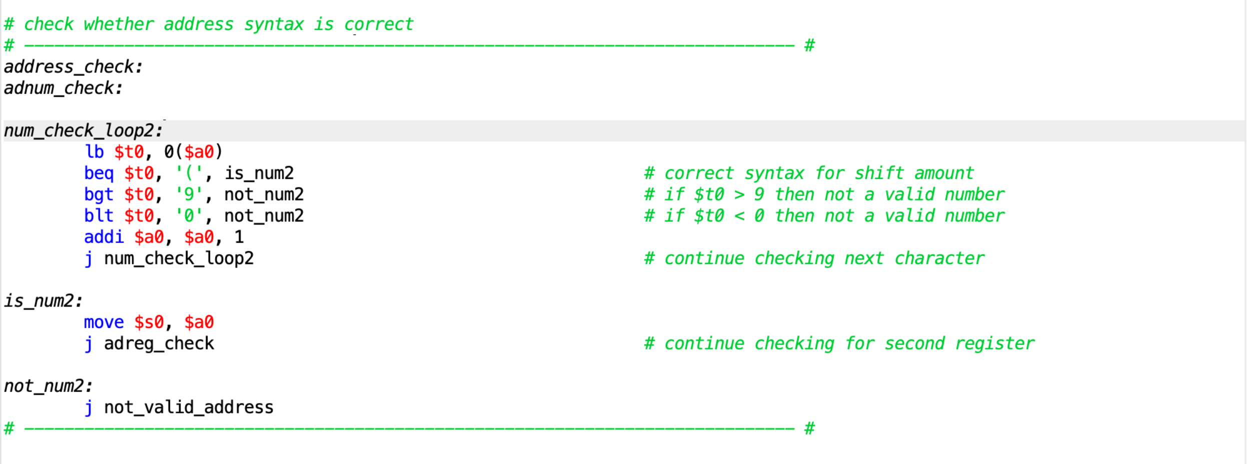
4.5. num_check:



```
# check whether current parameter is a valid number
# ---------------------------------------------------------------------------- #
num_check:
        move $a0, $s0

num_check_loop:
        lb $t0, 0($a0)
        beq $t0, ',', is_num                            # end of parameter
        beq $t0, ' ', is_num                            # end of parameter
        beq $t0, 0, is_num                              # end of parameter
        beq $t0, '\n', is_num                           # end of parameter
        bgt $t0, '9', not_num                           # if $t0 > '9' then not a number
        blt $t0, '0', not_num                           # if $t0 < '0' then not a number
        addi $a0, $a0, 1
        j num_check_loop                                # continue checking

is_num:
        move $s0, $a0
        j jump_                                         # jump back

not_num:
        j not_num_error                                 # jump to not_num_error
```

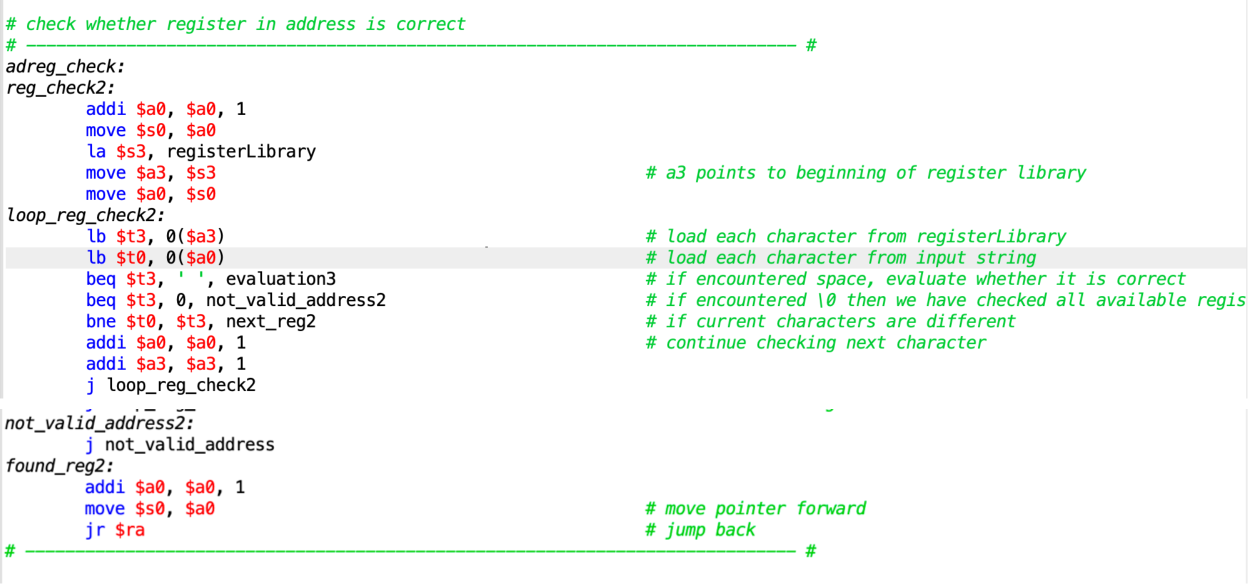
-check each character if it is not between 0 and 9 then not num
-if current character is comma, whitespace, NULL, \n then it has ended and we have checked all preceding characters and they are valid
-after num_check update $s0 to point to the next character after the number string

## 4.6. address_check:



```
# check whether address syntax is correct
# ---------------------------------------------------------------------------- #
address_check:
adnum_check:

num_check_loop2:
        lb $t0, 0($a0)
        beq $t0, '(', is_num2                                  # correct syntax for shift amount
        bgt $t0, '9', not_num2                                 # if $t0 > 9 then not a valid number
        blt $t0, '0', not_num2                                 # if $t0 < 0 then not a valid number
        addi $a0, $a0, 1
        j num_check_loop2                                      # continue checking next character

is_num2:
        move $s0, $a0
        j adreg_check                                          # continue checking for second register

not_num2:
        j not_valid_address
# ---------------------------------------------------------------------------- #
```

-check if each character is between 0 and 9 or not, it it is then continue checking, else not a valid number of shift amount
-if current character is `(` then we have checked or preceding numbers and they are valid -> valid syntax of address
-if shift amount’s syntax is correct we continue to check for address of second register with adreg_check:



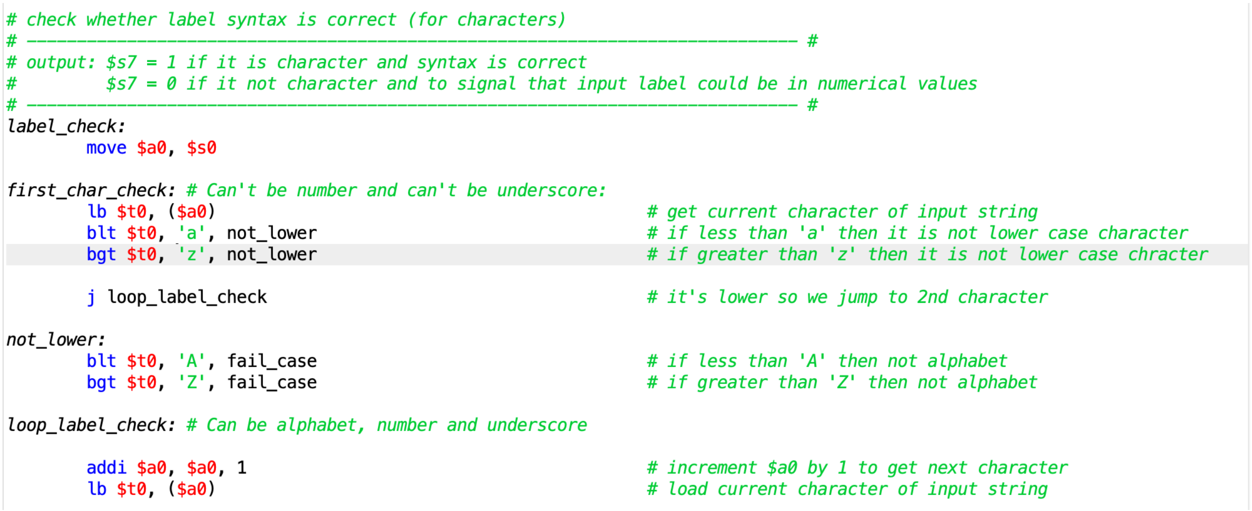
```
# check whether register in address is correct
# ---------------------------------------------------------------------------- #
adreg_check:
reg_check2:
        addi $a0, $a0, 1
        move $s0, $a0
        la $s3, registerLibrary
        move $a3, $s3                                          # a3 points to beginning of register library
        move $a0, $s0
loop_reg_check2:
        lb $t3, 0($a3)                                         # load each character from registerLibrary
        lb $t0, 0($a0)                                         # load each character from input string
        beq $t3, ' ', evaluation3                              # if encountered space, evaluate whether it is correct
        beq $t3, 0, not_valid_address2                         # if encountered \0 then we have checked all available regis
        bne $t0, $t3, next_reg2                                # if current characters are different
        addi $a0, $a0, 1                                       # continue checking next character
        addi $a3, $a3, 1
        j loop_reg_check2
not_valid_address2:
        j not_valid_address
found_reg2:
        addi $a0, $a0, 1
        move $s0, $a0                                          # move pointer forward
        jr $ra                                                 # jump back
# ---------------------------------------------------------------------------- #
```

+ similar to reg_check, this process is used for checking if the parameter is a valid register

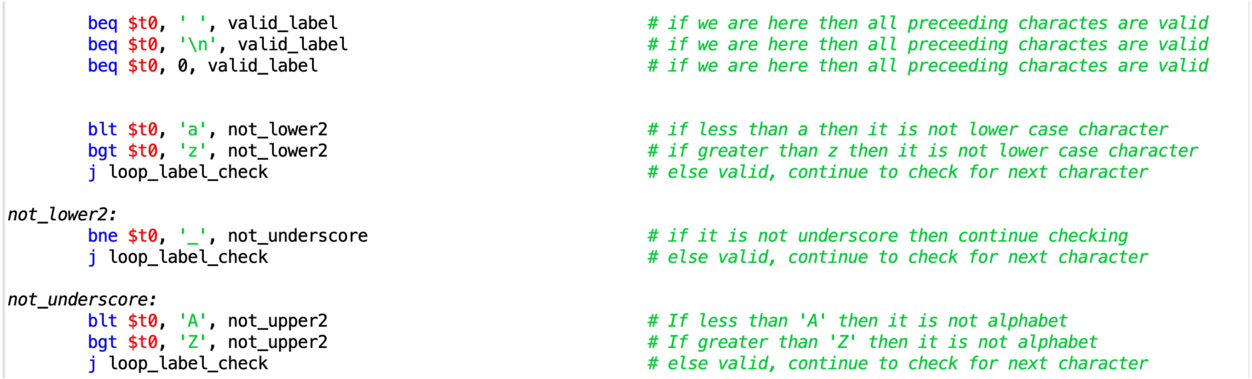
+ if $t3 is a whitespace then evaluate if $t0 is `)`, if it is then correct syntax, else not a valid address

+ once found $s0 will point to the next character after `)`

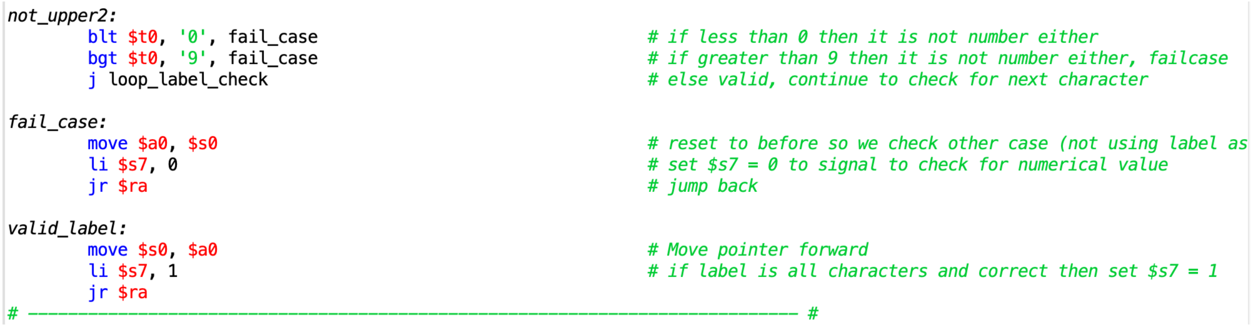
4.7. label_check: output $s7 = 1 if label is correct in syntax, $s7 = 0 to signal that we need to do a further check if label is in numerical value



```
# check whether label syntax is correct (for characters)
# -------------------------------------------------------------------------- #
# output: $s7 = 1 if it is character and syntax is correct
#         $s7 = 0 if it not character and to signal that input label could be in numerical values
# -------------------------------------------------------------------------- #
label_check:
        move $a0, $s0

first_char_check: # Can't be number and can't be underscore:
        lb $t0, ($a0)                                           # get current character of input string
        blt $t0, 'a', not_lower                                 # if less than 'a' then it is not lower case character
        bgt $t0, 'z', not_lower                                 # if greater than 'z' then it is not lower case chracter

        j loop_label_check                                      # it's lower so we jump to 2nd character

not_lower:
        blt $t0, 'A', fail_case                                 # if less than 'A' then not alphabet
        bgt $t0, 'Z', fail_case                                 # if greater than 'Z' then not alphabet

loop_label_check: # Can be alphabet, number and underscore

        addi $a0, $a0, 1                                        # increment $a0 by 1 to get next character
        lb $t0, ($a0)                                           # load current character of input string
```



```
        beq $t0, ' ', valid_label                               # if we are here then all preceeding charactes are valid
        beq $t0, '\n', valid_label                              # if we are here then all preceeding charactes are valid
        beq $t0, 0, valid_label                                 # if we are here then all preceeding charactes are valid


        blt $t0, 'a', not_lower2                                # if less than a then it is not lower case character
        bgt $t0, 'z', not_lower2                                # if greater than z then it is not lower case character
        j loop_label_check                                      # else valid, continue to check for next character

not_lower2:
        bne $t0, '_', not_underscore                            # if it is not underscore then continue checking
        j loop_label_check                                      # else valid, continue to check for next character

not_underscore:
        blt $t0, 'A', not_upper2                                # If less than 'A' then it is not alphabet
        bgt $t0, 'Z', not_upper2                                # If greater than 'Z' then it is not alphabet
        j loop_label_check                                      # else valid, continue to check for next character
```



```
not_upper2:
        blt $t0, '0', fail_case                                 # if less than 0 then it is not number either
        bgt $t0, '9', fail_case                                 # if greater than 9 then it is not number either, failcase
        j loop_label_check                                      # else valid, continue to check for next character

fail_case:
        move $a0, $s0                                           # reset to before so we check other case (not using label as
        li $s7, 0                                               # set $s7 = 0 to signal to check for numerical value
        jr $ra                                                  # jump back

valid_label:
        move $s0, $a0                                           # Move pointer forward
        li $s7, 1                                               # if label is all characters and correct then set $s7 = 1
        jr $ra
# -------------------------------------------------------------------------- #
```

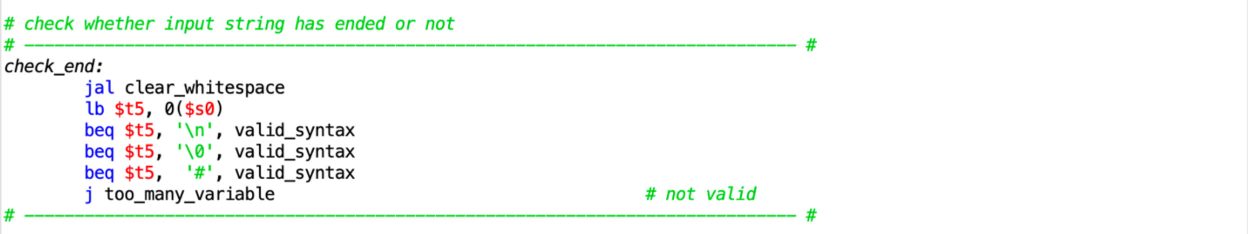
-check for first character (it cannot be a number and cannot be underscore), then do a loop to check remaining characters

-each iteration:

+ check if it is in alphabet, underscore or number, if it is then continue checking, if not then set $s7 = 0 and jump back

+ once we reach to a whitespace, \n or NULL then we have checked all preceding characters and they are valid, set $s7 = 1 and jump back

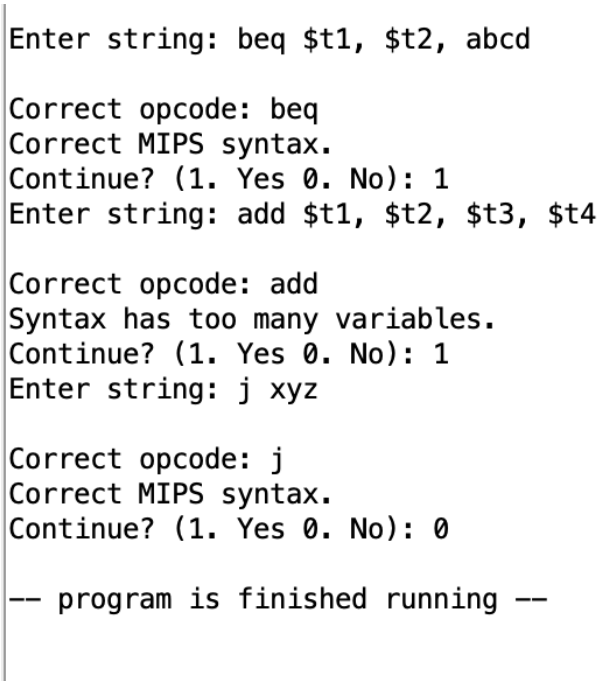
4.8. check_end:



```
# check whether input string has ended or not
# -------------------------------------------------------------------------------- #
check_end:
        jal clear_whitespace
        lb $t5, 0($s0)
        beq $t5, '\n', valid_syntax
        beq $t5, '\0', valid_syntax
        beq $t5,  '#', valid_syntax
        j too_many_variable                                     # not valid
# -------------------------------------------------------------------------------- #
```

-first call clear_whitespace delete all trailing whitespaces
-then check if current character is \n, \0 or #, if it is then valid, else not

## 5. Results demonstration



```
Enter string: beq $t1, $t2, abcd

Correct opcode: beq
Correct MIPS syntax.
Continue? (1. Yes 0. No): 1
Enter string: add $t1, $t2, $t3, $t4

Correct opcode: add
Syntax has too many variables.
Continue? (1. Yes 0. No): 1
Enter string: j xyz

Correct opcode: j
Correct MIPS syntax.
Continue? (1. Yes 0. No): 0

-- program is finished running --
```