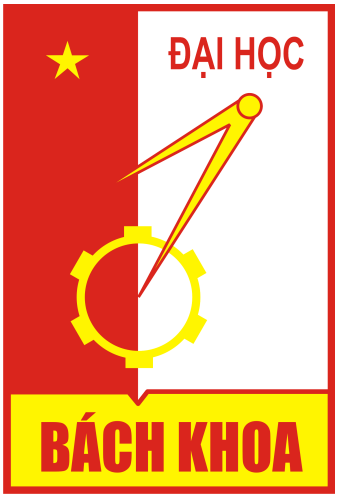
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



Group 10

Report

Computer Architecture Lab

Advisor: Master Le Ba Vui

HA NOI, JANUARY 2024

Member list

No.	Full name	Student ID
1	Dương Hoàng Hải	20215198
2	Nguyễn Hà Hiếu	20215202

Programming Task

- Dương Hoàng Hải : Project 7
- Nguyễn Hà Hiếu : Project 8

Contents

1 Project 7 - Dương Hoàng Hải

1.1 Problem

Checking the syntax of MIPS instruction

The MIPS processor's compiler checks the syntax of the instructions in the source code, whether they are correct or not, and then translates those instructions into machine code. Write a program that checks the syntax of any MIPS instruction (not include of pseudo instructions) as follows:

- **Enter a line of instructions from the keyboard**. For example, `beq $s1,31,$t4`
- **Check if the opcode is correct or not?** In this example, `beq` is correct so the program should display "opcode: beq, correct"
- **Check if the operands are correct or not?** In this example, `$s1` is correct, `31` is incorrect, `$t4` doesn't need to check anymore.

1.2 Method

Construct a string that stores the format of each instruction with the instruction name and type of each operand.

"add**111;addi*112;addu*111;and**111;andi*112;beq**113;bgez*130;bgtz*130;blez*130;
bltz*130;bne**113;div**110;eret*000;jr***100;jal**300;j****300;lui**120;lb***121;lh***121;
lw***121;mfhi*100;mflo*100;mul**111;mult*110;nop**000;nor**111;ori**112;or***111;srl**112;
srlv*111;sll**112;sllv*111;slt**111;slti*112;sra**112;sb***121;sh***121;sw***121;sub**111;
teq**110;teqi*120;xor**111;xori*112"

In which, each instruction name (opcode) consists of 5 characters including * (instructions with name of length no more than 4). The next 3 numeric characters are the types of operand 1, operand 2 and operand 3, respectively. Instructions are separated by ';'

The types of operands are conventionally defined as follows:

- Register: 1
- Integer: 2
- Label: 3
- None: 0 (means that the instruction does not have enough 3 operands)

1.3 Algorithm

1. Get opcode

   Iterate through the input string, store each of it characters to token until reaching space or newline.

2. Check opcode

   - Iterate through each instruction (substring) of the string 'list'
   - For each instrution in the list, compare each character of the token with each character of the opcode.
   - If they are the same, compare the next character. Otherwise, move to the next instruction in the list
   - During the process, if we encounters the character '*', if the corresponding character in the token is null, then a match is found. Otherwise, move on to checking the next instruction in the list.
   - If we reach the end of string list, then no match is found.

3. Get operand and check brackets

   - Starting traverse from the current input string character
   - If the current character is '(', mark that a '(' is found
   - Igonre the space and ','
   - Then store each character of input string to token until we reach a comma, a newline, a null, a space, a '(' or a ')' character.

- If the current character is ‘)’, mark that a ‘)’ is found.
- If the current character is ‘ ’, check if the remaining characters contain a ‘)’.This rule is applied for the third operand.

4. Check null operand

   Check if the token is empty or not.

5. Check register operand

   - Iterate through each accepted register (substring) of the string ‘register’
   - Compare each character of the token with each character of the register.
   - If they are the same, compare the next character. Otherwise, move to the next register in the list
   - During the process, if we encounters the character ‘*’, if the corresponding character in the token is null, then a match is found. Otherwise, move on to checking the next register in the list.
   - If we reach the end of string register, then no match is found.

6. Check integer operand

   - Check if the token is empty or not.
   - Check if first character is ’-’ or not.If yes, and there is not any other character, the operand is incorrect.
   - For other characters, check if they are numeric characters or not.

7. Check label operand

   - Check if the token is empty or not.
   - Check if the first character is number or not.
   - Every character must be in the accepted characters string.

1.4 Source code

```
.data
  command: .asciiz "Enter an instruction: "
  str1: .asciiz "opcode: "
str2: .asciiz "operand: "
str3: .asciiz ", correct.\n"
str4: .asciiz ", incorrect!\n"
msg_correct: .asciiz "The instruction you entered has the correct syntax.\n"
msg_incorrect: .asciiz "The instruction you entered has incorrect syntax !\n"
msg_continue: .asciiz "Do you want to continue the program ? Enter 1 for Yes
and 0 for No: "
input: .space 100
token: .space 20
list: .asciiz "add**111;addi*112;addu*111;and**111;andi*112;beq**113;bgez*130;
bgtz*130;blez*130;bltz*130;bne**113;div**110;eret*000;jr***100;jal**300;
j****300;lui**120;lb***121;lh***121;lw***121;mfhi*100;mflo*100;mul**111;
mult*110;nop**000;nor**111;ori**112;or***111;srl**112;srlv*111;sll**112;
sllv*111;slt**111;slti*112;sra**112;sb***121;sh***121;sw***121;sub**111;
teq**110;teqi*120;xor**111;xori*112"
char: .asciiz "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_"
register: .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***;$23***;$24***;$25***;$26***;$27***;$28***;$29***;$30***;
$31***;$32***"
.text
main:

getInput:
        li $v0, 4
```

```
la $a0, command
syscall

li $v0, 8 # read a string
la $a0, input   # a0 = address of input string
li $a1, 100
syscall

la $a1,token # a1 = address of token (a substring of the input string, to
store opcode or operands)
li $s7,0 # i= 0 = current input string index
li $t8,0 # Used to check whether the input string contains '('
li $t9,0 # Used to check whether the input string contains ')'
j getOpcode
exit:
# Exit the program
li $v0,10
        syscall
end_main:

getOpcode: # Extract opcode from input string, store to token

        add $t1,$a0,$s7 # t1 = address of input[i]
        add $t3,$a1,$s7 # t3 = address of token[i]
        lb $t2,0($t1)
        beq $t2, ' ', done
beq $t2, '\n', done
        sb $t2,0($t3)
addi $s7, $s7, 1
j getOpcode
done:
        li $t2,'\0'
        sb $t2,0($t3) # Append '\0' to token
```

```
la $a2, list # a2 = address of list
jal checkOpcode

beq $s2, 1, correctOpcode
j incorrectOpcode

correctOpcode:

li $v0, 4
la $a0, str1
syscall

li $v0, 4
la $a0, token
syscall

li $v0, 4
la $a0, str3
syscall

li $s5,5 # Used to get the first operand's type
li $s6,8 # Used for condition after finish checking 3 operand's types
j getOpr
incorrectOpcode:

li $v0, 4
la $a0, str1
syscall

li $v0, 4
la $a0, token
syscall
```

```
li $v0, 4
la $a0, str4
syscall

j incorrect
checkOpcode:
        li $s2,0 # Initial value for check = 0
        li $s0,0 # i=0
        loop1:
        add $t0,$a2,$s0 # t0 = address of list[i]
        lb $t0,0($t0) # t0 = list[i]
        beq $t0,0,finish # No matches were found
        move $s1,$s0 # j=i
        li $s3,0 # k = 0
        inner_loop:
        add $t0,$a2,$s1 # t0 = address of list[j]
        lb $t0,0($t0) # t0 = list[j]

        add $t1,$a1,$s3 # t1 = address of token[k]
        lb $t1,0($t1) # t1 = token[k]

        bne $t0,'*',skip
        bne $t1,'\0',end # If we get a '*' means that the token must terminate
(the correspoding char must be '\0')
        li $s2,1 # Find a match
        j finish
        skip:
        beq $t0,$t1,cont # If list[j] == token[k], continue checking
this instruction opcode
        j end
        cont:
        addi $s1,$s1,1 # j++
        addi $s3,$s3,1 # k++
```

```
        j inner_loop
        end:
        addi $s0,$s0,9 # Go to next instruction opcode
j loop1
        finish:
        jr $ra
getOpr:         # Extract operands from input string, store to token
        la $a0,input
        li $t0,0  # i = 0

        add $t1,$a0,$s7
        lb $t2,0($t1)
        beq $t2, '(',openBracket
        # Ignore space and comma
        truncate:
        addi $s7,$s7,1
        add $t1,$a0,$s7
        lb $t2,0($t1)
        beq $t2, ' ',truncate
        beq $t2, ',',truncate
        beq $t2, '(',openBracket
        # Get the operand

        loop2:
        add $t1,$a0,$s7 # t1 = address of next input string character
        add $t3,$a1,$t0 # t3 = address of token[i]
        lb $t2,0($t1)
        beq $t2, ',', doneGetOpr
beq $t2, '\n', doneGetOpr
beq $t2,'\0',doneGetOpr
beq $t2,' ',closeBracket
beq $t2,'(',doneGetOpr
beq $t2,')',closeBracket
```

```
        sb $t2,0($t3)
        cont2:
addi $t0, $t0, 1
addi $s7,$s7,1
j loop2

doneGetOpr:
        li $t2,'\0'
        sb $t2,0($t3) # Append '\0' to token

        add $t0,$s0,$s5 # t0 = index of first operand's type in this
instruction format
        add $t0,$a2,$t0
        lb $t0,0($t0)
        addi $t0,$t0,-48 # t0 = first operand's type
        li $s4,0 # Initialize checkOpr = 0
        case0:
bne $t0, 0, case1
jal checkNullOpr
j checked
case1:
bne $t0, 1, case2
jal checkRegisterOpr
j checked
case2:
bne $t0, 2, case3
jal checkIntegerOpr
j checked
case3:
jal checkLabelOpr
j checked
checked:
        addi $t2,$s5,1
```

```
        bne $t2,$s6,skipBracket
        xor $t1,$t8,$t9
        bne $t1,$zero,incorrect
        skipBracket:
beq $s4, 1, correctOpr # If checkOpr = 1, print a correct message
j incorrectOpr

checkNullOpr: # Check the null operand (does not exist)
        lb $t2,0($a1)
        beq $t2,'\0',isEmpty
        j endCheckNullOpr
        isEmpty:
        li $s4,1
endCheckNullOpr:
        jr $ra

checkRegisterOpr: # Check the register operand
       # Compare the token with each register in the string 'register'
       li $s4,1
       la $a3,register # a3 = address of string of registers
       li $t1,0 # i=0
       loop5:
       move $t2,$t1 # j=i
       li $t5,0 # k=0
       inner_loop5:
       add $t3,$a3,$t2
       lb $t3,0($t3) # t3 = register[j]
       add $t4,$a1,$t5
       lb $t4,0($t4) # t4 = token[k]
       beq $t3,'\0', isNotRegister
       beq $t3,'*',skip5      # If we get a '*' means that the token must
       terminate (the correspoding char must be '\0')
       bne $t3,$t4,outer5
```

```
        j cont5
        skip5:
        beq $t4,'\0',endCheckRegisterOpr
        j outer5
        cont5:
        addi $t2,$t2,1 # j++
        addi $t5,$t5,1 # k++
        j inner_loop5
        outer5:
        addi $t1,$t1,7 # Check the next register in the list
        j loop5
        isNotRegister:
        li $s4,0
endCheckRegisterOpr:
         jr $ra

checkIntegerOpr: # Check the integer operand
        li $s4,1
        li $t1,0 # i=0
        add $t2,$a1,$t1
        lb $t2,0($t2) # t2 = token[0]
        beq $t2,'\0',isNotInteger  # To avoid the case that token is empty
        bne $t2,'-',loop3
        addi $t1,$t1,1 # i++
        add $t2,$a1,$t1
        lb $t2,0($t2)
        beq $t2,'\0',isNotInteger # To avoid the case that token contains
        only a '-'
        loop3:
        add $t2,$a1,$t1
        lb $t2,0($t2) # t2 = token[i]
        beq $t2,'\0',endCheckIntegerOpr
        blt $t2,48,isNotInteger
```

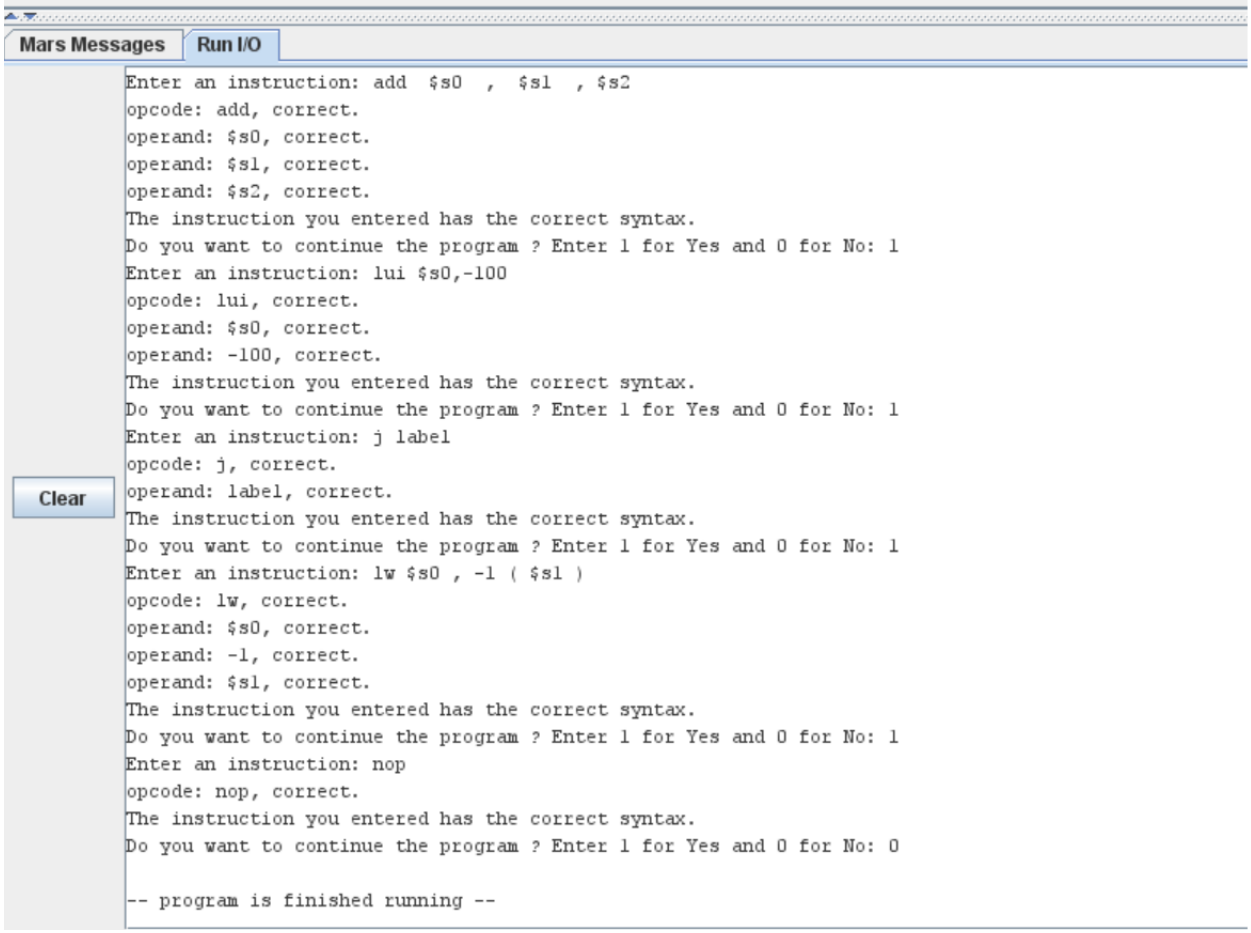
```
        bgt $t2,57,isNotInteger
        addi $t1,$t1,1 # i++
        j loop3
        isNotInteger:
        li $s4,0
endCheckIntegerOpr:
         jr $ra

checkLabelOpr: # Check the label
        li $s4,1
        li $t1,0 # i=0
        add $t2,$a1,$t1
        lb $t2,0($t2) # t2 = token[0]
        beq $t2,'\0',isNotLabel  # To avoid the case that token is empty
        la $a3,char # a3 = address of string of accepted characters
        add $t2,$a1,$t1
        lb $t2,0($t2) # t2 = token[0]
        slti $t3,$t2,58
        li $at,47
        slt $t4,$at,$t2
        and $t5,$t3,$t4
        beq $t5,1,isNotLabel
        loop4:
        add $t2,$a1,$t1
        lb $t2,0($t2) # t2 = token[i]
        beq $t2,'\0',endCheckLabelOpr
        li $t6,0 # j=0
        inner_loop4:
        add $t7,$a3,$t6
        lb $t7,0($t7) # t7 = char[j]
        beq $t7,'\0',isNotLabel
        beq $t7,$t2,outer3
        addi $t6,$t6,1 # j++
```

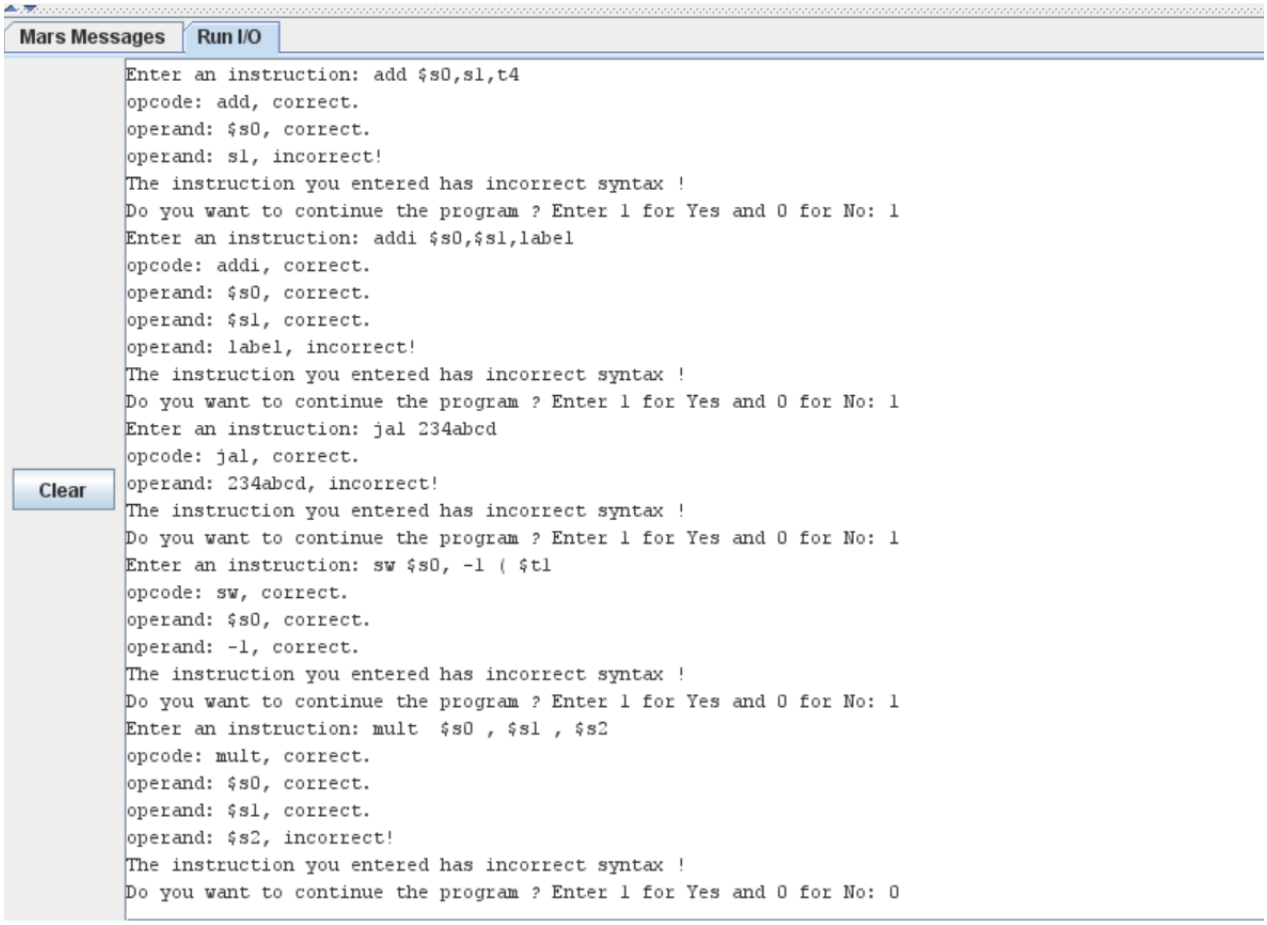
```
        j inner_loop4
        outer3:
        addi $t1,$t1,1 # i++
        j loop4
        isNotLabel:
        li $s4,0
endCheckLabelOpr:
         jr $ra
correctOpr:

         beq $t0,0,correct
li $v0, 4
la $a0, str2
syscall

li $v0, 4
la $a0, token
syscall

li $v0, 4
la $a0, str3
syscall

         addi $s5,$s5,1
blt $s5,$s6,getOpr # Continue to get the next operand, if there are not
enough 3 operands

j correct
incorrectOpr:

li $v0, 4
la $a0, str2
syscall
```

```
li $v0, 4
la $a0, token
syscall

li $v0, 4
la $a0, str4
syscall
incorrect: # The instruction is incorrect
li $v0, 4
la $a0, msg_incorrect
syscall
j continue
correct: # The instruction is correct
li $v0, 4
la $a0, msg_correct
syscall
continue: # Continue the program or not
li $v0, 4
la $a0, msg_continue
syscall

li $v0, 5
syscall

bne $v0, $zero, getInput
j exit
openBracket:
# Find a '(', set $t8 to 1
        li $t8,1
        j truncate
closeBracket:
# Check if the remaining characters of the input string contain ')'
```

```
move $t4,$s7
loop6:
add $t2,$a0,$t4
lb $t2,0($t2)
beq $t2,'\0',skip6
beq $t2,')',close
j cont6
close:
li $t9,1
j skip6
cont6:
addi $t4,$t4,1
j loop6
skip6:
j doneGetOpr
```

1.5 Simulation results




Mars Messages
Run I/O
Clear
Enter an instruction: add $s0,s1,t4
opcode: add, correct.
operand: $s0, correct.
operand: s1, incorrect!
The instruction you entered has incorrect syntax !
Do you want to continue the program ? Enter 1 for Yes and 0 for No: 1
Enter an instruction: addi $s0,$s1,label
opcode: addi, correct.
operand: $s0, correct.
operand: $s1, correct.
operand: label, incorrect!
The instruction you entered has incorrect syntax !
Do you want to continue the program ? Enter 1 for Yes and 0 for No: 1
Enter an instruction: jal 234abcd
opcode: jal, correct.
operand: 234abcd, incorrect!
The instruction you entered has incorrect syntax !
Do you want to continue the program ? Enter 1 for Yes and 0 for No: 1
Enter an instruction: sw $s0, -1 ($t1
opcode: sw, correct.
operand: $s0, correct.
operand: -1, correct.
The instruction you entered has incorrect syntax !
Do you want to continue the program ? Enter 1 for Yes and 0 for No: 1
Enter an instruction: mult $s0 , $s1 , $s2
opcode: mult, correct.
operand: $s0, correct.
operand: $s1, correct.
operand: $s2, incorrect!
The instruction you entered has incorrect syntax !
Do you want to continue the program ? Enter 1 for Yes and 0 for No: 0

2 Project 8 - Nguyễn Hà Hiếu

2.1 Problem

The RAID5 drive system requires at least 3 hard disks, in which parity data will be stored on 3 drives as shown below. Write a program to simulate the operation of RAID 5 with 3 drives, assuming each data block has 4 characters. The interface is as shown in the example below. Limit the length of the input string to a multiple of 8.

2.2 Method

- Get input: input is a string whose length is a multiple of 8.
- Check for validity of input: check if the input string's length is valid or not.
- Print out result: print each block of result, convert parity string to ascii and print.
- After finishing printing result, ask if user wants to continue or not.

2.3 Algorithm

1. Input

   This function is used to get input from user and store it to $s0$

```
input:
        la $a0, prompt
        li $v0, 4
        syscall

        la $a0, string
        li $a1, 1000
        li $v0, 8
        syscall

        move $s0, $a0
        jr $ra
```

2. Check validation
   This function counts number of characters of the input string. It sets s1=-1 if the input has length of 0 or not a multiple of 8. Otherwise s1=0 means the input is valid.
   If s1=-1 then print error message.

3. Convert

```
45  #check if string has length is a multiple of 8
46  check_validation:
47          li $s1, 0 #Set s1 = 0 means valid
48          li $t3, 0 #i=0
49          loop_check:
50                  add $t1, $s0, $t3
51                  lb $t2, 0($t1) #string[i]
52                  beq $t2, 10, end_loop_check
53                  addi $t3, $t3, 1
54                  j loop_check
55          end_loop_check:
56                  beq $t3, 0, error_msg
57                  div $t4, $t3, 8
58                  mfhi $t4
59                  bne $t4, 0, error_msg
60                  jr $ra
61                  nop
62                  error_msg:
63                          li $s1, -1 #s1=-1 means invalid input
64                          la $a0, error
65                          li $v0, 4
66                          syscall
67                          jr $ra
```

+This function print out the last 2 characters of each xor result.
+The parity string is saved in t8.
+First set t4=1, then t6=4 is 1 word, then shift t8 right to get the second last word, get the character by add it to t7 which is address of hex, and then print out.
+Decrease t4–, t4=0, similar to the above to print out the last word.

4. RAID5 Divide into 3 blocks

- Block1: disk 1 and 2 will have 4 bytes of input string, parity string will be saved in disk 3.
  After printing all, check if there remains characters, if not exit the function, if yes continue to Block2.
- Block2: disk 1 and 3 will have 4 bytes of input string, parity string will be saved in disk 2. It is similar to Block1.

```
convert:
# Convert parity string to hexa
        li $t4, 1                           #t4 = 7

loopH:
        blt $t4, $0, endloopH               # t4 < 0  -> endloop
        sll $s6, $t4, 2                     # s6 = t4*4
        srlv $a0, $t8, $s6                  # a0 = t8>>s6
        andi $a0, $a0, 0x0000000f           # a0 = a0 & 0000 0000 0000 0000 0000 0000 0000 1111
        la $t7, hex                         # t7 = adrress of hex
        add $t7, $t7, $a0                   # t7 = t7 + a0
        lb $a0, 0($t7)                      # print hex[a0]
        li $v0, 11
        syscall


nextc:  addi $t4,$t4,-1                     # t4 --
        j loopH
        nop

endloopH:
        jr $ra
        nop
```

- Block3: disk 2 and 3 will have 4 bytes of input string, parity string will be saved in disk 1.
  If after 3 blocks there still remains characters left then return to block1 and start over.

5. Check continue input Print out continue message, if user press 'Y' then continue, if user press 'N' then terminate the program.

2.4 Source code

```
    .data
parity: .space 32 #Parity string
prompt: .asciiz "Enter string: "
hex: .byte  '0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'
d1: .space 4 #Disk 1
d2: .space 4 #Disk 2
d3: .space 4 #Disk 3
string: .space 1000 #input string
endline: .asciiz "\n"
error: .asciiz "Invalid input"
```

```
continue: .asciiz "Continue input? (Y/N)"
disk: .asciiz  "      Disk 1                Disk 2                Disk 3\n"
ms1: .asciiz   "-----------------      ----------------      ----------------\n"
ms2: .asciiz   "|     "
ms3: .asciiz   "      |      "
ms4: .asciiz  "[[ "
ms5: .asciiz "]]       "
comma: .asciiz ","

.text

main:

main_loop:
jal input
jal check_validation
bgezal $s1, RAID5

continue_main:
j check_continue_input

input:
la $a0, prompt
li $v0, 4
syscall

la $a0, string
li $a1, 1000
li $v0, 8
syscall

move $s0, $a0
jr $ra
```

```
#check if string has length is a multiple of 8
check_validation:
li $s1, 0 #Set s1 = 0 means valid
li $t3, 0 #i=0
loop_check:
        add $t1, $s0, $t3
lb $t2, 0($t1) #string[i]
beq $t2, 10, end_loop_check
addi $t3, $t3, 1
j loop_check
end_loop_check:
beq $t3, 0, error_msg
div $t4, $t3, 8
mfhi $t4
bne $t4, 0, error_msg
jr $ra
nop
error_msg:
li $s1, -1 #s1=-1 means invalid input
la $a0, error
li $v0, 4
syscall
jr $ra

convert:
# Convert parity string to hexa
li $t4, 1 #t4 = 7

loopH:
blt $t4, $0, endloopH # t4 < 0  -> endloop
sll $s6, $t4, 2 # s6 = t4*4
srlv $a0, $t8, $s6 # a0 = t8>>s6
```

```
andi $a0, $a0, 0x0000000f  # a0 = a0 & 0000 0000 0000 0000 0000 0000 0000 1111 => Get th
la $t7, hex  # t7 = adrress of hex
add $t7, $t7, $a0  # t7 = t7 + a0
lb $a0, 0($t7)  # print hex[a0]
li $v0, 11
syscall


nextc: addi $t4,$t4,-1 # t4 --
j loopH
nop

endloopH:
jr $ra
nop


RAID5:
#block 1 : save byte parity into disk 3
#block 2 : save byte parity into disk 2
#block 3 : save byte parity into disk 1
block1:


addi $t0, $zero, 0 # bytes printed (4 byte)
addi $t9, $zero, 0
addi $t8, $zero, 0
la $s1, d1 # s1 = adress of d1
la $s2, d2 # s2 = address of d2
la $a2, parity #

print11:
li $v0, 4 # print message2 : "|     "
```

```
la $a0, ms2
syscall


b11:

lb $t1, ($s0) # t1 = first value of input string
addi $t3, $t3, -1 # t3 = t3 -1, decrease the length of string
sb $t1, ($s1) # store t1 to disk 1
b12:

add $s5, $s0, 4 # s5 = s0 +4
lb $t2, ($s5) # t2 = inputstring[5]
addi $t3, $t3, -1 # t3 = t3  - 1
sb $t2, ($s2) # store t2 vao disk 2
b13:
# Save xor result into disk 3
xor $a3, $t1, $t2 # a3 = t1 xor t2
sw $a3, ($a2) # save a3 to ($a2)
addi $a2, $a2, 4 # Parity string
addi $t0, $t0, 1 # Get next character
addi $s0, $s0, 1
addi $s1, $s1, 1 # Increment disk 1 index
addi $s2, $s2, 1 # Increment disk 2 index
bgt $t0, 3, reset #  Reset disk
j b11
nop
reset:
la $s1, d1 # reset index to the first element in disk 1
la $s2, d2 # reset index to the first element in disk 2

print12:  #in Disk 1
lb $a0, ($s1) #print each char  in Disk 1
```

```
li $v0, 11
syscall
addi $t9, $t9, 1
addi $s1, $s1, 1
bgt $t9, 3, next11
j print12
nop

next11:   #Prepare to print Disk 2    "|         |"
li $v0, 4
la $a0, ms3
syscall
li $v0, 4
la $a0, ms2
syscall

print13: # Print disk 2
lb $a0, ($s2)
li $v0, 11
syscall
addi $t8, $t8, 1
addi $s2, $s2, 1
bgt $t8, 3, next12
j print13
nop

next12: # Prepare to print Disk 3
li $v0, 4
la $a0, ms3
syscall
li $v0, 4
la $a0, ms4
syscall
```

```
la $a2, parity # a2 = address of parity string[i]
addi $t9, $zero, 0 # t9 = i

print14: # Convert parity string to ASCII and print
lb $t8, ($a2) # t8 = adress of parity string[i]
jal convert
nop
li $v0, 4
la $a0, comma # print  " , "
syscall

addi $t9, $t9, 1 # parity string's index  + 1
addi $a2, $a2, 4
bgt $t9, 2, end1 # Print the first 3 pair of parity string with ","
j print14
end1: # Print the last pair of parity string and finish Disk 3
lb $t8, ($a2)
jal convert
nop
li $v0, 4
la $a0, ms5
syscall

li $v0, 4 # Endline to start new block
la $a0, endline
syscall
beq $t3, 0, exit1 # If string's length = 0 means no more string to print   , exit
j block2 # If there are still more left => block2
nop

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

block2:
```

```
#Similar to block1 but with parity string printed in Disk 2

la $a2, parity
la $s1, d1 # s1 = address of Disk 1
la $s3, d3 # s3 =            Disk 3
addi $s0, $s0, 4
addi $t0, $zero, 0

print21:
# print "|     "
li $v0, 4
la $a0, ms2
syscall

b21:

lb $t1, ($s0) # t1 = address of Disk 1
addi $t3, $t3, -1
sb $t1, ($s1)
b23:
# Next 4 bytes into Disk 3
add $s5, $s0, 4
lb $t2, ($s5)
addi $t3, $t3, -1
sb $t2, ($s3)

b22:
# 4 byte parity vao Disk 2
xor $a3, $t1, $t2
sw $a3, ($a2)
addi $a2, $a2, 4
addi $t0, $t0, 1
addi $s0, $s0, 1
```

```
addi $s1, $s1, 1
addi $s3, $s3, 1
bgt $t0, 3, reset2
j b21
nop
reset2:
la $s1, d1
la $s3, d3
addi $t9, $zero, 0

print22:
# print Disk 1
lb $a0, ($s1)
li $v0, 11
syscall
addi $t9, $t9, 1
addi $s1, $s1, 1
bgt $t9, 3, next21
j print22
nop

next21:
li $v0, 4
la $a0, ms3
syscall
la $a2, parity
addi $t9, $zero, 0
li $v0, 4
la $a0, ms4
syscall

print23:
lb $t8, ($a2)
```

```
jal convert
nop
li $v0, 4
la $a0, comma #print ","
syscall
addi $t9, $t9, 1
addi $a2, $a2, 4
bgt $t9, 2, next22
j print23
nop

next22:
#print Disk 2 in ACSII
lb $t8, ($a2)
jal convert
nop

li $v0, 4
la $a0, ms5
syscall

li $v0, 4
la $a0, ms2
syscall
addi $t8, $zero, 0

print24:
# print Disk 3
lb $a0, ($s3)
li $v0, 11
syscall
addi $t8, $t8, 1
addi $s3, $s3, 1
```

```
bgt $t8, 3, end2
j print24
nop

end2:
# If there are no more characters then exit
# If there still remains then => block3
li $v0, 4
la $a0, ms3
syscall
li $v0, 4
la $a0, endline
syscall
beq $t3, 0, exit1
#-------------------------------------------------------------
block3:
# Byte parity saved in Disk1
# 2 block 4 byte saved in Disk 2 , Disk 3
la $a2, parity
la $s2, d2
la $s3, d3
addi $s0, $s0, 4
addi $t0, $zero, 0
print31:

li $v0, 4
la $a0, ms4
syscall
b32:
#  byte stored in Disk 2

lb $t1, ($s0) # in first loop, t1 = first H
addi $t3, $t3, -1
```

```
sb $t1, ($s2)
b33:
# store in Disk 3 first
add $s5, $s0, 4 #
lb $t2, ($s5) # in first loop , t2 = the second "H"
addi $t3, $t3, -1 # stored in disk 3
sb $t2, ($s3) # stored t2 in disk 3

b31:
# ham xor tinh parity
xor $a3, $t1, $t2 # a3 = parity number
sw $a3, ($a2) # stored in parity string
addi $a2, $a2, 4 # parity string's index + 4
addi $t0, $t0, 1
addi $s0, $s0, 1
addi $s2, $s2, 1 # disk2 +1
addi $s3, $s3, 1 #  disk 3 +1
bgt $t0, 3, reset3
j b32
nop
reset3:
# Reset to first of disk2 , disk 3
la $s2, d2
la $s3, d3
la $a2, parity
addi $t9, $zero, 0 #index

print32:
# Print parity string
lb $t8, ($a2)
jal convert
nop
li $v0, 4
```

```
la $a0, comma
syscall

addi $t9, $t9, 1
addi $a2, $a2, 4
bgt $t9, 2, next31
j print32
nop

next31:
# print the last parity byte
lb $t8, ($a2)
jal convert
nop

li $v0, 4
la $a0, ms5
syscall
li $v0, 4
la $a0, ms2
syscall
addi $t9, $zero, 0

print33:
#print disk 2, print 4 byte from Disk 2
lb $a0, ($s2)
li $v0, 11
syscall
addi $t9, $t9, 1
addi $s2, $s2, 1
bgt $t9, 3, next32
j print33
nop
```

```
next32:

addi $t9, $zero, 0
addi $t8, $zero, 0
li $v0, 4
la $a0, ms3
syscall
li $v0, 4
la $a0, ms2
syscall
print34:
#  print  4 byte from Disk 3
lb $a0, ($s3)
li $v0, 11
syscall
addi $t8, $t8, 1
addi $s3, $s3, 1
bgt $t8, 3, end3
j print34
nop

end3:
#Finish printing Disk 3
li $v0, 4
la $a0, ms3 # Print: "      |"
syscall

li $v0, 4
la $a0, endline
syscall
beq $t3, 0, exit1 # If there are no more characters -> exit
# If there still remains  -> return to block1
```

```
nextloop: addi $s0, $s0, 4 #Skip 4 characters already printed
j block1
nop

exit1: # Print ---------- and finish RAID 5 simulation
li $v0, 4
la $a0, ms1
syscall

j continue_main

check_continue_input:
la $a0, endline
li $v0, 4
syscall

la $a0, continue  #Print continue message to ask if the user wants to continue input
li $v0, 4
syscall

la $a0, endline
li $v0, 4
syscall

li $v0, 12 #Read command character "Y" to continue, "N" to terminate
syscall

move $s7, $v0
la $a0, endline
li $v0, 4
```

```
syscall

beq $s7, 'N', exit
j main_loop

exit:
li $v0, 10
syscall
```

2.5 Simulation results