Assembly Language & Computer Architecture Lab (143685)

Final Project Report

Nguyen Tieu Phuong (20210692)

Problem 8

Simulate the working of RAID 5, 3 disks, with parity. Assume each data block is 4 chars. Input size must be 8x (x is an integer).

Input: `DCE.****ABCD1234HUSTHUST`

Output:

```
   Disk 1              Disk 2              Disk 3
---------------     ---------------     ---------------
|   DCE.        |   |   ****        |   [[6e, 69, 6f, 04]]
|   ABCD        |   [[70, 70, 70, 70]]  |   1234        |
[[00, 00, 00, 00]]  |   HUST        |   |   HUST        |
---------------     ---------------     ---------------
```

Solution

Reading User Input

User input reading is fairly simple. Some exception handling when user input nothing or when user input a string of invalid length should be considered.

```
getInput:     li      $v0, 8
              la      $a0, input
              li      $a1, 200
              syscall
              lb      $t1, 0($a0)
              beq     $t1, 10, getInput   # handle exception when user input nothing
```

Reading finishes when we detect the end-of-string character - in this case is the null terminator '\n' with ASCII code of 10. To check for the length, we implement a counter that increases with each char, and check if this value is divisible by 8.

Calculating the Parity

MIPS does not provide support for printing out HEX characters in our format **`[[6e, 69, 6f, 04]]`** but only in the form **`0x000000`**. To get around this, we maintain a "dictionary" of the hex characters, and use some logic to get the character we needed. The masking technique is used.

```
hexChar:    .byte      '0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'

hex:         li      $t4, 7              # loop counter
hexLoop:     blt     $t4, $0, endHex
             sll     $s6, $t4, 2         # s6 = t4 * 4 = {28, 24, 20...}
             srlv    $a0, $t8, $s6       # a0 = t8 >> s6
             andi    $a0, $a0, 0x0f      # keep last byte of $a0
             la      $t7, hexChar
             add     $t7, $t7, $a0
             bgt     $t4, 1, continue
             lb      $a0, 0($t7)         # print hex[a0]
             li      $v0, 11
             syscall
continue:    addi    $t4, $t4, -1
             j       hexLoop
```

Printing the Output

To format a table properly, we need to calculate the exact number of spaces and character we need. The tab `'\t'` is not quite useful here, but rather a predefined space should help line things up.

```
diskHeader: .asciiz     "     Disk 1                Disk 2                Disk 3 \n"
border:     .asciiz     "-----------------       -----------------   -----------------\n"
comma:      .asciiz     ","
leftPipe:   .asciiz     "|     "
midPipe:    .asciiz     "     |       "
```

```
leftBracket:.asciiz     "[[ "
rightBracket:.asciiz    "]]       "
```

The most challenging part of this project is the loading and printing of the disks' data. We need to load data into each disk, while calculating the parity for 8-byte group-wise, with the disk that acts as the parity disk changed after each turn.

To achieve this, data loading and printing must run simultaneously while keeping track of the sequential turn of the data-or-parity disks.

```
split2:       la      $a2, parity
              la      $s1, Disk1
              la      $s3, Disk3
              addi    $s0, $s0, 4
              addi    $t0, $0, 0
print21:      li      $v0, 4
              la      $a0, leftPipe
              syscall
b12:          lb      $t1, ($s0)
              addi    $t3, $t3, -1
              sb      $t1, ($s1)
b32:          add     $s5, $s0, 4
              lb      $t2, ($s5)
              addi    $t3, $t3, -1
              sb      $t2, ($s3)
b22:          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       b12
```

In the first turn, disk 1 and 2 are for data, disk 3 is for parity. In the second turn, disk 1 and 3 are for data, disk 2 is for parity. Finally in the third turn, disk 2 and 3 are for data and disk 1 is for parity. In the attached source codes, the turns are represented by the subroutines **`split1, split2, split3`**; and byte-wise printing is represented as the subroutines **`b[i][j]`**,

with **i** is the disk index we are writing to, and **j** is the turn number (1, 2 or 3). **print[i][j]** means the printing of the turn **i** at the **j**-th time.

Michellaneous

There are additional subroutines that allow users to continuously provide input and see the output until they no longer want to.