HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and Communications Technology

FINAL PROJECT REPORT

Assembly Language and Computer Architecture Lab

--IT3280E--

Supervisor: Lê Bá Vui

Students: Trần Huy Hoàng - 20225973

Đinh Xuân Toàn - 20226067

CATALOG

SUBJECT 6: RAID5 SIMULATION

1. Requirement

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.

In this example, a string is entered from the keyboard (DCE.****ABCD1234HUSTHUST) will be divided in to blocks of 4 byte. First 4 bytes “DCE.” stored in Disk 1, next 4 bytes “****” stored Disk 2, data stored in Disk 3 is 4 parity bytes computed from 2 first blocks 6e=’D’ xor ‘*’ ; 69=’C’ xor ‘*’; 6f=’E’ xor ‘*’ ; 04=’.’ xor ‘*’

```
Nhap chuoi ki tu : DCE.****ABCD1234HUSTHUST
     Disk 1                Disk 2                Disk 3
 --------------        --------------        --------------
|     DCE.     |      |     ****     |      [[ 6e,69,6f,04]]
|     ABCD     |      [[ 70,70,70,70]]      |     1234     |
[[ 00,00,00,00]]      |     HUST     |      |     HUST     |
 --------------        --------------        --------------
```

2. Methods

- Input Handling
  - Purpose: Input a string of characters and validate its length.
  - Steps:
    - Prompt the user for input.
    - Validate that the input length is a multiple of 8 (required for dividing into 4-byte blocks).
    - If invalid, display an error message and re-prompt for input.
- Disk and Parity Management
  - Virtual Disks:
    - disk1, disk2, disk3: Memory blocks representing three disks.
  - Parity Calculation:
    - Perform bitwise XOR on two blocks of data to calculate the parity block, stored in the third disk.

3. Algorithms

- Input Validation

Objective: Ensure the string length is a multiple of 8.

1. Loop through the input string to calculate its length.
2. Check the remainder (length % 8):
    a. If remainder == 0 or 8, proceed.
    b. Else, display an error and re-input.

- RAID5 Parity Simulation

The algorithm divides data into 4-byte blocks and stores them across disks in the following pattern:

Block1:
    Disk1: Data 1
    Disk2: Data 2
    Disk3: Parity (Data1 XOR Data2)

Block2:
    Disk1: Data 3
    Disk3: Data 4
    Disk2: Parity (Data3 XOR Data4)

Block3:
    Disk2: Data 5
    Disk3: Data 6
    Disk1: Parity (Data5 XOR Data6)

Steps:

1. Load Data: Fetch 4-byte blocks from the input string.
2. XOR Operation: Calculate parity:
3. Store Data: Save Data and Parity into respective disks.
4. Print Results: Display data stored in disk1, disk2, and disk3.

- Hexa convertion

To print parity values as hex:

1. Extract 4-bit segments from the byte using bit shifts and masks.
2. Map each segment to its hexadecimal representation.

4. Source code

```
.data
prompt: .asciz "Nhap chuoi ky tu : "
# ASCII into hexa
hex: .byte '0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'
disk1: .space 4
disk2: .space 4
disk3: .space 4
array: .space 32                    # Store parities (results for data XOR)
string: .space 5000                 # Input string
enter: .asciz "\n"
error_length: .asciz "Do dai chuoi khong hop le! Nhap lai.\n"
disk: .asciz  "      Disk 1              Disk 2              Disk 3\n"
msg1: .asciz   "-----------------      ----------------      ----------------\n"
msg2: .asciz   "|     "
msg3: .asciz   "      |      "
msg4: .asciz  "[[ "
msg5: .asciz "]]       "
comma: .asciz ","
message: .asciz "Try another string???"

.text
main:
la s1, disk1                        # s1 = address of disk 1
la s2, disk2                        # s2 = address of disk 2
la s3, disk3                        # s3 = address of disk 3
la a2, array                        # Address of parities

j input
nop

input: li a7, 4                             # Print " Nhap chuoi ky tu"
la a0, prompt
ecall

li a7, 8                            # Get string
la a0, string
li a1, 1000
ecall

mv s0, a0                           # s0 = address of input string

li a7, 4                            # Print " Disk1 Disk2 Disk3"
la a0, disk
ecall
li a7, 4                            # Print " ------ "
la a0, msg1
ecall
```

```
#------------------ Check whether input string's length is multiple of 8 ------------------
---------------
length:
addi t3, zero, 0                    # t3 = length
addi t0, zero, 0                    # t0 = index

check_char:
# Check \n?
add t1, s0, t0                          # t1 = address of string[i]
lb t2, 0(t1)                        # t2 = string[i]
li s4, 10                       # '\n' = 10 ASCII
beq t2, s4, test_length             # string[i] = '\n'
nop

addi t3, t3, 1                          # length++
addi t0, t0, 1                      # index++
j check_char
nop

test_length:
mv t5, t3                           # t5 = string length
beq t0,zero,error                   # If only '\n' -> error

andi t1, t3, 0x0000000f                 # t1 = last byte
bne t1, zero, test1                 # last byte = 0 or =8 --> multiple of 8
j block1                            # last byte != 0 and != 0 --> error
nop
test1:  li s11, 8
        beq t1, s11, block1
j error
nop

error: li a7, 4                             # Print "Do dai chuoi khong hop le! Nhap
lai.\n"
la a0, error_length
ecall
j input                                     # Back to input
nop



HEX:
#------------------------  Get parities  ------------------------------------------------------
----------
# Co 1 dau vao la s8 chua parity string roi chuyen tu ascii sang hexa
li t4, 7                            # t4 = 7

loopH:
blt t4, zero, endloopH                  # t4 < 0  -> endloop
```

```
slli s6, t4, 2                  # s6 = t4*4
srl a0, s8, s6                 # a0 = s8 >> s6
andi a0, a0, 0x0000000f        # Get the last byte of a0
la s7, hex                     # s7 = adrress of hex
add s7, s7, a0                      # s7 = s7 + a0
li a4, 1
bgt t4, a4, nextc              # if t4 > 1 , jump to nextC
lb a0, 0(s7)                   # Print hex[a0]
li a7, 11
ecall


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

endloopH:
jr ra
nop


#------------------------------ RAID5 SIMULATION-----------------------------------
RAID5:
# Block 1 : byte parity is stored in disk 3
# Block 2 : byte parity is stored in disk 2
# Block 3 : byte parity is stored in disk 1
block1:
# Funtion block1: First 2 4-byte blocks are stored in disk1, disk2; parity is stored in
disk3
addi t0, zero, 0
addi s9, zero, 0
addi s8, zero, 0
la s1, disk1                   # s1 = adress of disk1
la s2, disk2                   # s2 = address of disk2
la a2, array                   #

print11:
li a7, 4                       # print msg2 : "|     "
la a0, msg2
ecall

# Example: DCE. and  ****
b11:
# Store DCE. into disk1
lb t1, (s0)                    # t1 = first value of input string
addi t3, t3, -1                     # t3 = length -1
sb t1, (s1)                    # store t1 into disk1
b12:
# Store **** ịnto disk2
addi s5, s0, 4                      # s5 = s0 + 4
```

```
lb t2, (s5)                         # t2 = string[5]
addi t3, t3, -1                             # t3 = t3  - 1
sb t2, (s2)                         # store t2 into disk2
b13:
# Store XOR result into disk3
xor a3, t1, t2                      # a3 = t1 xor t2
sw a3, (a2)                           # Store a3 into a2
addi a2, a2, 4                              # Parity string
addi t0, t0, 1                      # Next char
addi s0, s0, 1                              # Eliminate considered char, eg : "D"
addi s1, s1, 1                              # Address of disk 1 +1
addi s2, s2, 1                              # Address of disk 2 +1
li a6, 3                          # a6 = 3
bgt t0, a6, reset                           # 4 byte are considered --> reset disk
j b11
nop
reset:
# Reset disks
la s1, disk1
la s2, disk2

print12:
# Print disk1
lb a0, (s1)                 # Print each char in disk1
li a7, 11
ecall
addi s9, s9, 1
addi s1, s1, 1
bgt s9, a6, next11          # Print 4 times --> end priting disk1
j print12
nop

next11:
# Prepair for printing disk2    "|        |"
li a7, 4
la a0, msg3
ecall
li a7, 4
la a0, msg2
ecall

print13:
# Print disk2
lb a0, (s2)
li a7, 11
ecall
addi s8, s8, 1
addi s2, s2, 1
bgt s8, a6, next12          # Print 4 times --> end printing disk2
```

```
j print13
nop

next12:
# Prepair for printing disk3
li a7, 4
la a0, msg3
ecall
li a7, 4
la a0, msg4
ecall
la a2, array                # a2 = address of parity string[i]
addi s9, zero, 0            # s9 = i

print14:
# Convert parity string --> ASCII and print
lb s8, (a2)                 # s8 = adress of parity string[i]
jal HEX
nop
li a7, 4
la a0, comma                        # Print ','
ecall

addi s9, s9, 1              # Parity string's index +1
addi a2, a2, 4              # Skip considered parity string
li a5, 2
bgt s9, a5, endisk1         # Print first 3 parities with ','
j print14
endisk1:                            # Print last parity --> end printing disk3
lb s8, (a2)
jal HEX
nop
li a7, 4
la a0, msg5
ecall

li a7, 4                    # Next line, new block
la a0, enter
ecall
beq t3, zero, exit1         # If string length = 0 --> exit
j block2                    # else --> block2
nop

#---------------------------------------
block2:
# Funtion block2: Next 2 4-byte blocks are stored in disk1, disk3; parity is stored in
disk2
la a2, array
la s1, disk1
```

```
la s3, disk3
addi s0, s0, 4
addi t0, zero, 0

print21:
# print "|     "
li a7, 4
la a0, msg2
ecall

# Example: ABCD and 1234
b21:
# Store 4 bytes into disk1
lb t1, (s0)
addi t3, t3, -1
sb t1, (s1)
b23:
# Store next 4 bytes into disk3
addi s5, s0, 4
lb t2, (s5)
addi t3, t3, -1
sb t2, (s3)

b22:
# Store XOR result into disk2
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, a6, reset2
j b21
nop
reset2:
# Reset disks
la s1, disk1
la s3, disk3
addi s9, zero, 0                # Index

print22:
lb a0, (s1)
li a7, 11
ecall
addi s9, s9, 1
addi s1, s1, 1
bgt s9, a6, next21
j print22
```

```
nop

next21:
li a7, 4
la a0, msg3
ecall
la a2, array
addi s9, zero, 0
li a7, 4
la a0, msg4
ecall

print23:
lb s8, (a2)
jal HEX
nop
li a7, 4
la a0, comma
ecall
addi s9, s9, 1
addi a2, a2, 4
bgt s9, a5, next22
j print23
nop

next22:
lb s8, (a2)
jal HEX
nop

li a7, 4
la a0, msg5
ecall

li a7, 4
la a0, msg2
ecall
addi s8, zero, 0

print24:
lb a0, (s3)
li a7, 11
ecall
addi s8, s8, 1
addi s3, s3, 1
bgt s8, a6, endisk2
j print24
nop
```

```
endisk2:
li a7, 4
la a0, msg3
ecall
li a7, 4
la a0, enter
ecall
beq t3, zero, exit1

#-------------------------------
block3:
# Funtion block2: Next 2 4-byte blocks are stored in disk2, disk3; parity is stored in
disk1
la a2, array
la s2, disk2
la s3, disk3
addi s0, s0, 4
addi t0, zero, 0
print31:
# Print '[['
li a7, 4
la a0, msg4
ecall
b32:
# Byte stored in Disk 2
lb t1, (s0)
addi t3, t3, -1
sb t1, (s2)
b33:
# Store in Disk 3
addi s5, s0, 4
lb t2, (s5)
addi t3, t3, -1
sb t2, (s3)

b31:
# Store XOR result into disk1
xor a3, t1, t2
sw a3, (a2)
addi a2, a2, 4
addi t0, t0, 1
addi s0, s0, 1
addi s2, s2, 1
addi s3, s3, 1
bgt t0, a6, reset3
j b32
nop
reset3:
# Reset disks
```

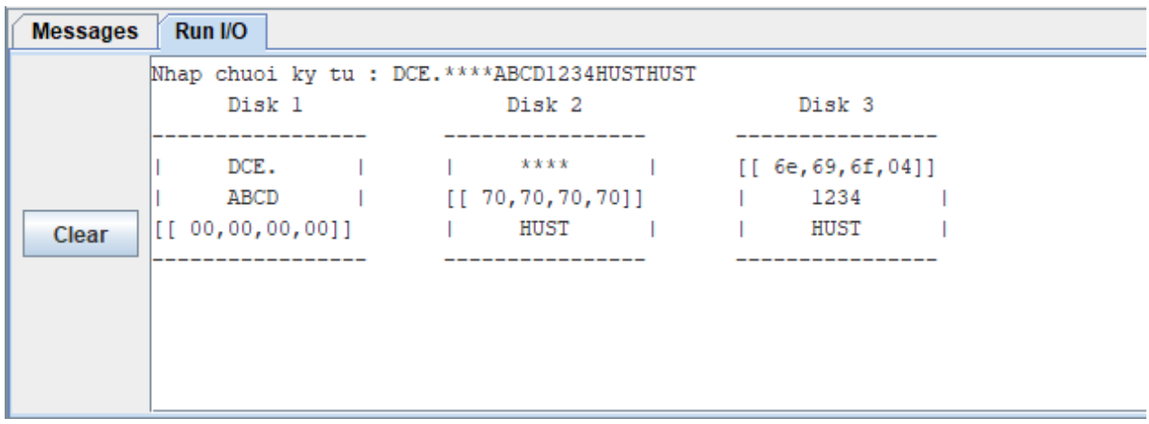
```
la s2, disk2
la s3, disk3
la a2, array
addi s9, zero, 0                # Index

print32:
lb s8, (a2)
jal HEX
nop
li a7, 4
la a0, comma
ecall

addi s9, s9, 1
addi a2, a2, 4
bgt s9, a5, next31
j print32
nop

next31:
lb s8, (a2)
jal HEX
nop

li a7, 4
la a0, msg5
ecall
li a7, 4
la a0, msg2
ecall
addi s9, zero, 0

print33:
lb a0, (s2)
li a7, 11
ecall
addi s9, s9, 1
addi s2, s2, 1
bgt s9, a6, next32
j print33
nop

next32:
addi s9, zero, 0
addi s8, zero, 0
li a7, 4
la a0, msg3
ecall
li a7, 4
```

```
la a0, msg2
ecall
print34:
lb a0, (s3)
li a7, 11
ecall
addi s8, s8, 1
addi s3, s3, 1
bgt s8, a6, endisk3
j print34
nop

endisk3:
li a7, 4
la a0, msg3
ecall

li a7, 4
la a0, enter
ecall
beq t3, zero, exit1


#-----------End first 6 4-byte blocks-----------------------------
#-----------Next 6 4-byte blocks----------------------------------

nextloop: addi s0, s0, 4              # Skip 4 consider characters
j block1
nop

exit1:
# Print ------ and end RAID simulation
li a7, 4
la a0, msg1
ecall
j ask
nop

#--------------------END RAID 5 SIMULATION-------------------------


#--------------------TRY ANOTHER STRING----------------------------
ask:    li a7, 50                     # Ask if wanna try
la a0, message
ecall
beq a0, zero, clear           # a0 :     0 = YES;  1 = NO;  2 = CANCEL
nop
j exit
nop
```

```
# clear function: Return string to original state
clear:
la s0, string
add s3, s0, t5          # s3: last byte's address used in string
li t1, 0                # Set t1 = 0

goAgain:
# Return string to empty state to start again
sb t1, (s0)
nop
addi s0, s0, 1
bge s0, s3, input
nop
j goAgain
nop

#-----Exit program----------
exit:   li a7, 10
ecall
```

5. Simulation results

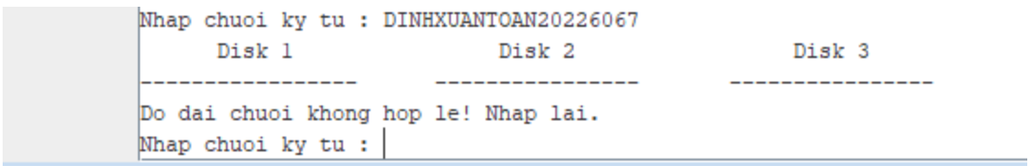
INPUT: DCE.****ABCD1234HUSTHUST



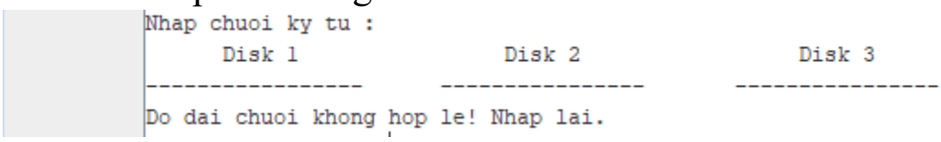
INPUT: DINHXUANTOAN20226067

=> Do dai chuoi khong hop le

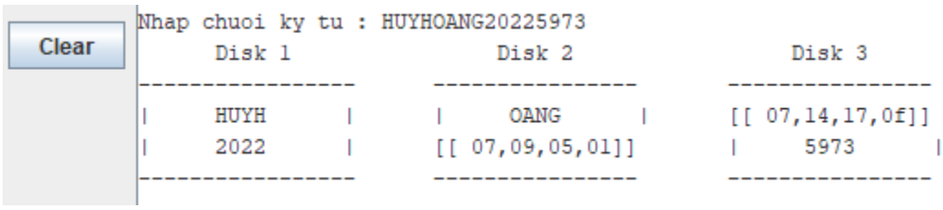
Cause string length is not multiple of 8



INPUT:
=> Do dai chuoi khong hop le
Cause we input nothing



INPUT: HUYHOANG20225973



SUBJECT 16: PLAY MUSICAL SCRIPT

1. Requirement

+ Research about the system call to play a sound.

+ A musical script is a string containing sets of four values. Each set includes pitch, duration, instrument type, and volume. For example: “60, 1200,1, 120, 73, 220,1, 125, ...”

+ Prepare at least 4 scripts.

+ When the program is running, the user will choose which script will be played by pressing the corresponding key from 1 to 4 in the key matrix. Press 0 to pause.

2. Methods

- Music Input Handling
  - Purpose: Receive user input to select a song and validate its validity.
  - Steps:
    - Display the song menu and prompt the user to choose a song between 1 to 4.
    - Validate the input (1-4). If invalid, prompt the user to re-enter.
- Sound Wave Generation
  - Purpose: Play the sound for the selected music notes.
  - Steps:
    - Define the frequency and duration for each music note.
    - Use the syscall li a7, 31 to play the sound, with parameter a0 as frequency, a1 as duration, and a3 as volume.
    - Play the sound for each note in the song, with a delay between each note.

3. Algorithms

- Song Input Validation

1. Display the song menu and accept input from the user.
2. Check if the input is valid (1-4); if not, prompt for re-input.

- Play Song

1. Fetch Notes: Each song contains a series of notes with frequency and duration.
2. Generate Sound:
   - Use the syscall li a7, 31 to generate sound:
     - a0: frequency of the note.
     - a1: duration of the note.
     - a3: volume (from 0 to 100).
   - Example: li a0, 61 (Do frequency), li a1, 1000 (duration 1000 ms), li a3, 50 (volume 50).
3. Handle Delay: After each note, call the Sleep function to create a delay between notes.
4. Repeat: Continue playing the notes until the song is completed.

- Music Note Management

  - Each note is defined with its frequency (e.g., Do = 61 Hz, Re = 62 Hz, etc.) and duration (time to play each note).
  - Notes can be extended or modified to suit the song's needs.

4. Source code

```
.data
menu:     .asciz "Choose a song (1-Twinkle, 2-Happy, 3-Jingle, 4-Ode): "  # Menu for song selection
invalid:   .asciz "Invalid selection!\n"  # Error message when an invalid option is chosen
delay_value: .word 1  # Delay time for creating pauses between notes

.text
.globl main

main:
    # Display the song selection menu
    la a0, menu          # Load the address of the menu string
    li a7, 4             # Syscall: Print string
    ecall

    # Read the user input for song choice
    li a7, 5             # Syscall: Read integer
    ecall
    addi t0, a0 , 0      # Store the input value in t0

    # Check the user input and jump to the corresponding song
    li t1, 1             # T1 = 1
    beq t0, t1, music1    # If input is 1 -> Twinkle song

    li t1, 2             # T1 = 2
    beq t0, t1, music2    # If input is 2 -> Happy Birthday song

    li t1, 3             # T1 = 3
    beq t0, t1, music3    # If input is 3 -> Jingle Bells song

    li t1, 4             # T1 = 4
    beq t0, t1, music4    # If input is 4 -> Ode to Joy song

    # Display error message if the selection is invalid
    la a0, invalid       # Load the address of the error message
    li a7, 4             # Syscall: Print string
    ecall
    j main               # Return to the menu

# Song 1: Twinkle Twinkle Little Star
music1:
    li a3, 100     # a3 sets the volume: 100
    li a2, 2       # a2 sets the instrument type: piano
    li a7, 31      # Make sound syscall
```

```
    # Play the melody for Twinkle Twinkle Little Star
    # To change the melody, modify the pitch (a0) and duration (a1) for each note
    jal Do
    jal Do
    jal Sol
    jal Sol
    jal La
    jal La
    jal Sol
    jal Sleep

    jal Fa
    jal Fa
    jal Mi
    jal Mi
    jal Re
    jal Re
    jal Do
    jal Sleep
    j main  # Return to the menu

# Song 2: Happy Birthday
music2:
    li a3, 50       # a3 sets the volume: 50
    li a2, 24       # a2 sets the instrument type: guitar
    li a7, 31       # Make sound syscall

    # Play the melody for Happy Birthday
    # Modify pitch (a0) and duration (a1) for each note as needed
    jal Do
    jal Do
    jal Re
    jal Do
    jal Fa
    jal Mi
    jal Sleep

    jal Do
    jal Do
    jal Re
    jal Do
    jal Sol
    jal Fa
    jal Sleep

    jal Do
    jal Do
    jal Do
    jal La
```

```
    jal Fa
    jal Mi
    jal Re
    jal Sleep

    jal Si
    jal Si
    jal La
    jal Fa
    jal Sol
    jal Fa
    jal Sleep
    j main

# Song 3: Jingle Bells
music3:
    li a3, 70       # a3 sets the volume: 70
    li a2, 11       # a2 sets the instrument type: Chromatic Percussion
    li a7, 31       # Make sound syscall

    # Play the melody for Jingle Bells
    # Modify pitch (a0) and duration (a1) for each note as needed
    jal Mi
    jal Mi
    jal Mi
    jal Sleep
    jal Mi
    jal Mi
    jal Mi
    jal Sleep

    jal Mi
    jal Sol
    jal Do
    jal Re
    jal Mi
    jal Sleep

    jal Fa
    jal Fa
    jal Fa
    jal Fa
    jal Fa
    jal Mi
    jal Mi
    jal Sleep

    jal Mi
    jal Mi
```

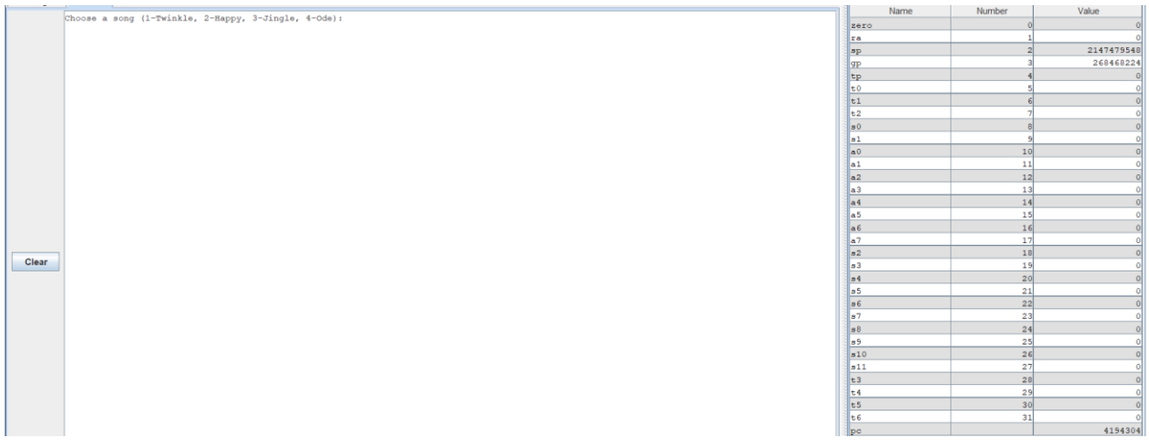
```
    jal Re
    jal Re
    jal Mi
    jal Re
    jal Sol
    jal Sleep
    j main

# Song 4: Ode to Joy
music4:
    li a3, 80       # a3 sets the volume: 80
    li a2, 33       # a2 sets the instrument type: Bass
    li a7, 31       # Make sound syscall

    # Play the melody for Ode to Joy
    # Modify pitch (a0) and duration (a1) for each note as needed
    jal Mi
    jal Mi
    jal Fa
    jal Sol
    jal Sol
    jal Fa
    jal Mi
    jal Re
    jal Do
    jal Do
    jal Re
    jal Mi
    jal Mi
    jal Sleep

    jal Re
    jal Re
    jal Mi
    jal Mi
    jal Fa
    jal Sol
    jal Sol
    jal Fa
    jal Mi
    jal Mi
    jal Sleep

    jal Mi
    jal Mi
    jal Fa
    jal Sol
    jal Sol
    jal Fa
```

```
    jal Mi
    jal Re
    jal Do
    jal Do
    jal Re
    jal Mi
    jal Mi
    jal Sleep
    j main

# Sleep function for creating a delay between notes
Sleep:
    la t0, delay_value
    lw t1, 0(t0)
Delay:
    addi t1, t1, -1
    bnez t1, Delay
    jr ra

# Note functions for musical notes
Do:
    li a0, 61    # Pitch of Do
    li a1, 1000  # Duration of Do
    ecall
    jr ra

Re:
    li a0, 62    # Pitch of Re
    li a1, 1000  # Duration of Re
    ecall
    jr ra

Mi:
    li a0, 64    # Pitch of Mi
    li a1, 1000  # Duration of Mi
    ecall
    jr ra

Fa:
    li a0, 65    # Pitch of Fa
    li a1, 1000  # Duration of Fa
    ecall
    jr ra

Sol:
    li a0, 67    # Pitch of Sol
    li a1, 1000  # Duration of Sol
    ecall
    jr ra
```

```
La:
    li a0, 69    # Pitch of La
    li a1, 1000  # Duration of La
    ecall
    jr ra

Si:
    li a0, 71    # Pitch of Si
    li a1, 1000  # Duration of Si
    ecall
    jr ra

# Long note functions for extended durations
Dolong:
    li a0, 61    # Pitch of Do
    li a1, 2000  # Duration of Do (longer duration)
    ecall
    jr ra

Relong:
    li a0, 62    # Pitch of Re
    li a1, 2000  # Duration of Re (longer duration)
    ecall
    jr ra

Milong:
    li a0, 64    # Pitch of Mi
    li a1, 2000  # Duration of Mi (longer duration)
    ecall
    jr ra

Falong:
    li a0, 65    # Pitch of Fa
    li a1, 2000  # Duration of Fa (longer duration)
    ecall
    jr ra

Sollong:
    li a0, 67    # Pitch of Sol
    li a1, 2000  # Duration of Sol (longer duration)
    ecall
    jr ra

Lalong:
    li a0, 69    # Pitch of La
    li a1, 2000  # Duration of La (longer duration)
    ecall
    jr ra
```

```
Silong:
    li a0, 71    # Pitch of Si
    li a1, 2000  # Duration of Si (longer duration)
    ecall
    jr ra

# Sharp note functions for higher pitches
Dothang:
    li a0, 62    # Pitch of D#
    li a1, 1000  # Duration of D#
    ecall
    jr ra

Rethang:
    li a0, 63    # Pitch of D#
    li a1, 1000  # Duration of D#
    ecall
    jr ra

Mithang:
    li a0, 65    # Pitch of F#
    li a1, 1000  # Duration of F#
    ecall
    jr ra

Fathang:
    li a0, 66    # Pitch of F#
    li a1, 1000  # Duration of F#
    ecall
    jr ra

Solthang:
    li a0, 68    # Pitch of G#
    li a1, 1000  # Duration of G#
    ecall
    jr ra

Lathang:
    li a0, 70    # Pitch of A#
    li a1, 1000  # Duration of A#
    ecall
    jr ra

Sithang:
    li a0, 72    # Pitch of B#
    li a1, 1000  # Duration of B#
    ecall
    jr ra
```

5. Simulation results



- Choose a song (1-Twinkle, 2-Happy, 3-Jingle, 4-Ode)
- Listen and enjoy
- After each song, you can choose another song.