**CompE-271**

* I declare that all material in this assignment is my own work except where there is clear reference to the work of others.
* I have read, understood and agree to the SDSU Policy on Plagiarism and Cheating on the university website at <http://go.sdsu.edu/student_affairs/srr/cheating-plagiarism.aspx> , the syllabus and the student-teacher contract for the consequences of plagiarism, including both academic and punitive sanctions.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Remark\*. By submitting this assignment report electronically, you are deemed to have signed the declaration above.*

4/25/2021

[Final Project Report]

[COMPE 271]

**Introduction**

In this program the user was be given the option to preform actions that revolve around the MIDI syscall. The first section of my program will give the user the ability to pick a root note and play either a Major or Natural Minor scale. After the user has chosen a root note and which scale, they want to hear, the MARS will use MIDI syscalls to output the scale that was chosen. The second part of my program will test the user on their pitch recognition. How this will be accomplished is that the user will first enter a root note. The program will then play the major scale for the root note. The catch is that one of the notes in the scale will be a semi tone off pitch and it is the user’s task to correctly guess which note in the scale is off pitch. If the correct note is guessed, then the user is prompted that they guessed correctly however if they get it wrong, they are prompted that they guessed incorrectly. The last section of my program will allow users too pick a root note and build chords. The chord structures available in my program are Major Chords, Minor Chords. Major 7th Chords, Minor 7th Chords and Dominant 7th Chords. After the user chooses a root note and a chord structure, the program will play each note in the chord separately in ascending order and then play the entire chord at once. This is only possible using the additional MIDI syscalls included in MARS++. After each section in my program, the user will have the option to choose to restart the program from the main menu or exit the program. Depending on what the user inputs will determine what option occurs. The entirety of my program was written in Assembly. The reason why I did not choose to write pseudo code for my program is because the MIDI syscall is unique to assembly and not native to C/C++. This means that most of my functions that I wrote in Assembly would have not been possible to write in C/C++.

**Source Code**

######################################

# Name: Rylan Bumbasi

# Date: 3/7/2021

# COMPE 271 Personal Project

#######################################

.data

#Arrays

rootNotes: .word 60 61 62 63 64 65 66 67 68 69 70 71 72 # Create an Array Contaning all the pitches from C4 to B4

#Strings for the the main program

myIntro: .asciiz "Welcome to MIDI Madness! By Rylan Bumbasi\n\n"

menuOptions: .asciiz "Enter 1 to Play Scales\n\nEnter 2 to take Pitch Recognition Test.\n\nEnter 3 for Chord Madness.\n\n"

requestInput: .asciiz "Please enter a number: "

displayError: .asciiz "\*\*\* INVALID INPUT \*\*\*, try again.\n\n"

rootNoteOptions: .asciiz "C4 = 0\nC# = 1\nD = 2\nD# = 3\nE = 4\nF = 5\nF# = 6\nG = 7\nG# = 8\nA = 9\nA# = 10\nB = 11\n"

askRestart: .asciiz "\nWould your like to return to the main menu?\n\n1 = Yes.\n2 = No.\n\n"

halfCount: .word 500

wholeCount: .word 1000

#Chord Madness Data Section

chordMadnessIntro: .asciiz "In this section of the program you will choose a root note and a chord stucture.\nThe MIDI player will then play you the corelating chord.\n\nPlease choose a root note.\n"

chordStructureOptions: .asciiz "\nPlease select a chord structure.\n\nMajor Chord = 1\nMinor Chord = 2\nMajor 7th Chord = 3\nMinor 7th Chord = 4\nDominant 7th Chord = 5\n"

#playingScales Data Section

playScalesIntro: .asciiz "In this section of the program You will select a root note and choose to play either a major or minor scale.\n\nPlease select a root note.\n\n"

scaleOptions: .asciiz "\nPlease select a Scale.\n1 = Major Scale\n2 = Minor Scale\n"

#pitchRecognitionTest Data Section

requestGuess: .asciiz "Which number note in the scale was off pitch?\n\n"

correctGuess: .asciiz "\nCongratulations, you guessed correctly :D\n"

incorrectGuess: .asciiz "\nSorry, you got it wrong :(\n"

pitchRecognitionTestIntro: .asciiz "In this section of the program you will first choose a root note.\nAfter a root note is chosen, the Major Scale for that Root note will be played.\n\nThe Catch is that one of the notes in the scale will be played off key and your goal is to identify which key is off pitch\n\n"

.text

# Display intro Message

main:

li $v0, 4

la $a0, myIntro

syscall

# Display Menu Options

li $v0, 4

la $a0, menuOptions

syscall

j getMenuInput

###############################################################################################################################

# Below are functions that can be universally used by every section in my program

displayRootNotes:

addi $sp,$sp,-4 # Allocate one spot for the stack

sw $ra, 0($sp) # push $ra onto the stack

li $v0, 4 # Display the Root Note Options

la $a0, rootNoteOptions

syscall

jal requestUserInput

lw $ra, 0($sp) # restore value of $ra

addi $sp, $sp, 4 # Decallocate the space createad for the stack

jr $ra

#########################################################################################################################################

# The nature of the getRootNote function written below is based off an an algorithim written by "The Simple Engineer" on YouTube

# A link to his original code will be included in my Final Lab Report

#########################################################################################################################################

getRootNote:

bgt $t4, $t7, exitGetRootNote # The loop will break at the index of the array that the user requested

sll $t5, $t4, 2 # 4 \* I

addu $t5, $t5, $s1 # rootNotes[I]

lw $t6, 0($t5) # $t6 = rootNotes[I]

addi $t4, $t4, 1 #Iterator + 1

j getRootNote

exitGetRootNote:

add $t1, $zero, $t6 #t1 hold's the pitch of the root note

lw $t3, halfCount($zero) # t3 = halfCount

addi $t7, $zero, 0

jr $ra

# Display error message in user inputs a number that isn't 1,2,3

errorMessage:

li $v0, 4

la $a0, displayError

syscall

getUserInput:

# Get users input for menu option

li $v0, 5

syscall

# Store result in t7

move $t7, $v0

jr $ra

requestUserInput:

li $v0, 4

la $a0, requestInput

syscall

jr $ra

#########################################################################################################################################

# The nature of the generateRandomInt function written below is based off an an algorithim written by "Ayman Jajja" on YouTube

# A link to his original code will be included in my Final Lab Report

#########################################################################################################################################

generateRandomInt:

# Generate a random number from 0 to 9

addi $a1, $zero, 10

addi $v0, $zero, 42

syscall

# If the Number generated is not 3,4,5 or 6, then regenerate the number

beq $a0, 3, grabRandomInt

beq $a0, 4, grabRandomInt

beq $a0, 5, grabRandomInt

beq $a0, 6, grabRandomInt

j generateRandomInt

grabRandomInt:

# Take the randomly generated number and send it to register $t0

add $t0, $zero, $a0

jr $ra

requestUserGuess:

# Prompt the user to enter a guess

li $v0, 4

la $a0, requestGuess

syscall

jr $ra

###############################################################################################################################################################

getMenuInput: # Ask for user Input

li $v0, 4

la $a0, requestInput

syscall

# Get users input for menu option

li $v0, 5

syscall

# Store result in t7

move $t7, $v0

beq $t7, 1, playScales # If the user inputs 1, program will jump to PlayScales

beq $t7, 2, pitchRecognitionTest # If the user inputs 2, the program will jump to pitchRecognitionTest

beq $t7, 3, chordMadness # If the user inputs 3, the program will jump to chordMadness

j errorMessage

playScales:

la $s1, rootNotes # Load array into register $s1

# Display play Scales Intro Message

li $v0, 4

la $a0, playScalesIntro

syscall

jal displayRootNotes # Jump and link to the displayRootNotes function

jal getUserInput

jal getRootNote # Jump and link to the displayRootNotes function

# Display Scale options

li $v0, 4

la $a0, scaleOptions

syscall

jal getUserInput

# Play a major or minor scale depending on the user input

beq $t7, 1, majorScale

beq $t7, 2, minorScale

j playScalesEnd

majorScale:

# Play the root note of the scale

li $v0, 31

la $a0, ($t1)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 2 #t2 = t1 + 2

# Play the second of the Scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 4 #t2 = t1 + 4

# Play the 3rd of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 5 #t2 = t1 + 5

# Play the 4th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 7 #t2 = t1 + 7

# Play the 5th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 9 #t2 = t1 + 9

# Play the 6th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 11 # t2 = t1 + 11

# Play the 7th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 12 #t2 = t1 + 12

# Play the tonic

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

j playScalesEnd

minorScale:

# Play Root Note of Scale

li $v0, 31

la $a0, ($t1)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 2 #t2 = t1 + 2

# Play the second of the Scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 3 #t2 = t1 + 3

# Play the 3rd of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 5 #t2 = t1 + 5

# Play the 4th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 7 #t2 = t1 + 7

# Play the 5th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 8 #t2 = t1 + 8

# Play the 6th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 10 #t2 = t1 + 10

# Play the 7th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 12 #t1 = t1 + 12

# Play the tonic

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

playScalesEnd:

j end

pitchRecognitionTest:

# Display Pitch Recognition intro

li $v0, 4

la $a0, pitchRecognitionTestIntro

syscall

la $s1, rootNotes # Load array into register $s1

jal displayRootNotes # Jump and link to the displayRootNotes function

jal getUserInput

jal getRootNote # Jump and link to the displayRootNotes function

jal generateRandomInt

lw $t3, wholeCount($zero) # t3 = wholeCount

# Depending on the random number generated, play a certain scale that is off pitch

beq $t0, 3, majorScaleOffPitch3

beq $t0, 4, majorScaleOffPitch4

beq $t0, 5, majorScaleOffPitch5

beq $t0, 6, majorScaleOffPitch6

majorScaleOffPitch3:

li $v0, 31

la $a0, ($t1) # Play Root Note of Scale

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 2 #t2 = t1 + 2

# Play the second of the Scale off pitch, (Semitone lower)

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 3 #t2 = t1 + 3

# Play the 3rd of the scale off pitch (semitone down)

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 5 #t2 = t1 + 5

# Play the 4th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 7 #t2 = t1 + 7

# Play the 5th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 9 #t2 = t1 + 9

# Play the 6th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 11 # t2 = t1 + 11

# Play the 7th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 12 #t2 = t1 + 12

# Play the tonic

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

add $t2, $zero, $t1 # reset t2 back to the root note

j guessNote

majorScaleOffPitch4:

li $v0, 31

la $a0, ($t1)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 2 #t2 = t1 + 2

# Play the second of the Scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 4 #t2 = t1 + 4

# Play the 3rd of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 6 #t2 = t1 + 6

# Play the 4th of the scale off pitch (semitone higher)

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 7 #t2 = t1 + 7

# Play the 5th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 9 #t2 = t1 + 9

# Play the 6th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 11 # t2 = t1 + 11

# Play the 7th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 12 #t2 = t1 + 12

# Play the tonic

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

add $t2, $zero, $t1 # reset t2 back to the root note

j guessNote

majorScaleOffPitch5:

li $v0, 31

la $a0, ($t1)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 2 #t2 = t1 + 2

# Play the second of the Scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 4 #t2 = t1 + 4

# Play the 3rd of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 5 #t2 = t1 + 5

# Play the 4th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 6 #t2 = t1 + 6

# Play the 5th of the scale off pitch (semi tone down)

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 9 #t2 = t1 + 9

# Play the 6th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 11 # t2 = t1 + 11

# Play the 7th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 12 #t2 = t1 + 12

# Play the tonic

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

add $t2, $zero, $t1 # reset t2 back to the root note

j guessNote

majorScaleOffPitch6:

li $v0, 31

la $a0, ($t1)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 2 #t2 = t1 + 2

# Play the second of the Scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 4 #t2 = t1 + 4

# Play the 3rd of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 5 #t2 = t1 + 5

# Play the 4th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 7 #t2 = t1 + 7

# Play the 5th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 10 #t2 = t1 + 10

# Play the 6th of the scale off pitch (semitone highter)

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 11 # t2 = t1 + 11

# Play the 7th of the scale

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

addi $t2, $t1, 12 #t2 = t1 + 12

# Play the tonic

li $v0, 31

la $a0, ($t2)

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

add $t2, $zero, $t1 # reset t2 back to the root note

j guessNote

guessNote:

jal requestUserGuess # Prompt the user to input a guess

jal requestUserInput # Ask the user for their input

jal getUserInput # Get user input

bne $t7, $t0, incorrect # If user guesses incorrectly, display a message

li $v0, 4

la $a0, correctGuess # If a user guesses correctly, display a message

syscall

j pitchRecognitionTestEnd

incorrect:

# Display a message that user was incorrect.

li $v0, 4

la $a0, incorrectGuess

syscall

j pitchRecognitionTestEnd

pitchRecognitionTestEnd:

j end

chordMadness:

# Channel 1 contains piano sound

initilizeChannelSounds:

li $v0, 38

la $a0, 1

la $a1, 0

syscall

# Channel 2 contains piano sound

li $v0, 38

la $a0, 2

la $a1, 0

syscall

# Channel 3 contains piano sound

li $v0, 38

la $a0, 3

la $a1, 0

syscall

# Channel 4 contains piano sound

li $v0, 38

la $a0, 4

la $a1, 0

syscall

# Channel 5 contains piano sound

li $v0, 38

la $a0, 5

la $a1, 0

syscall

la $s1, rootNotes # Load array into register $s1

# Display chord Madness Intro Message

li $v0, 4

la $a0, chordMadnessIntro

syscall

jal displayRootNotes # Jump and link to the displayRootNotes function

jal getUserInput

jal getRootNote # Jump and link to the displayRootNotes function

getChordStructure:

# Display Chord Structure Options

li $v0, 4

la $a0, chordStructureOptions

syscall

# Request input from user

li $v0, 4

la $a0, requestInput

syscall

li $v0, 5 # Get User's input for Chord Sturcture

syscall

move $t7, $v0 # Store user's input into $t7

beq $t7, 1, brokenMajorChord

beq $t7, 2, brokenMinorChord

beq $t7, 3, brokenMajor7thChord

beq $t7, 4, brokenMinor7thChord

beq $t7, 5, brokenMinor7thChord

j end

# Play each note in the chord individually first

brokenMajorChord:

li $v0, 31

la $a0, ($t1) # Play the Root note of the Chord

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

li $v0, 31

addi $t2, $t1, 4 # Increment the root note by 4 semitones to reach the 3rd of the scale and set that note to $t2

la, $a0, ($t2) # Play the 3rd of the scale

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

li $v0, 31

addi $t2, $t1, 7 # Increment the root note by 7 semitones to reach the 5th of the scale and set that note to $t2

la, $a0, ($t2) # Play the 5th of the Scale

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

j majorChord

majorChord:

li $v0, 37

la, $a0, ($t1) # Play the Root Note of the scale

la $a1, ($t3) #Play the note for the length of the value stored in $t3

la $a2, 1

la $a3, 127

syscall

# Play the 3rd of the scale

li $v0, 37

addi $t2, $t1, 4 # Increment the root note by 4 semitones to reach the 3rd of the scale and set that note to $t2

la, $a0, ($t2) # Play the 3rd of the scale

la $a1, ($t3) #Play the note for the length of the value stored in $t3

la $a2, 2

la $a3, 127

syscall

# Play the 5th of the scale

li $v0, 37

addi $t2, $t1, 7 # Increment the root note by 7 semitones to reach the 5th of the scale and set that note to $t2

la, $a0, ($t2) # Play the 5th of the Scale

la $a1, ($t3) # Play the note for the length of the value stored in $t3

la $a2, 3

la $a3, 127

syscall

j end

# Play each note in the chord individually first

brokenMinorChord:

li $v0, 31

la $a0, ($t1) # Play the Root note of the Chord

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

li $v0, 31

addi $t2, $t1, 3 # Increment the root note by 3 semitones to reach the flat 3rd of the scale and set that note to $t2

la, $a0, ($t2) # Play the flat 3rd of the scale

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

li $v0, 31

addi $t2, $t1, 7 # Increment the root note by 7 semitones to reach the 5th of the scale and set that note to $t2

la, $a0, ($t2) # Play the 5th of the Scale

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

j minorChord

minorChord:

li $v0, 37

la, $a0, ($t1) # Play the Root Note of the scale

la $a1, ($t3) #Play the note for the length of the value stored in $t3

la $a2, 1

la $a3, 127

syscall

# Play the 3rd of the scale

li $v0, 37

addi $t2, $t1, 3 # Increment the root note by 3 semitones to reach the 3rd of the scale and set that note to $t2

la, $a0, ($t2) # Play the 3rd of the scale

la $a1, ($t3) #Play the note for the length of the value stored in $t3

la $a2, 2

la $a3, 127

syscall

# Play the 5th of the scale

li $v0, 37

addi $t2, $t1, 7 # Increment the root note by 7 semitones to reach the 5th of the scale and set that note to $t2

la, $a0, ($t2) # Play the 5th of the Scale

la $a1, ($t3) # Play the note for the length of the value stored in $t3

la $a2, 3

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

j end

brokenMajor7thChord:

li $v0, 31

la $a0, ($t1) # Play the Root note of the Chord

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

li $v0, 31

addi $t2, $t1, 4 # Increment the root note by 4 semitones to reach the 3rd of the scale and set that note to $t2

la, $a0, ($t2) # Play the 3rd of the scale

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

li $v0, 31

addi $t2, $t1, 7 # Increment the root note by 7 semitones to reach the 5th of the scale and set that note to $t2

la, $a0, ($t2) # Play the 5th of the Scale

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

li $v0, 31

addi $t2, $t1, 11 # Increment the root note by 11 semitones to reach the 7th of the scale and set that note to $t2

la, $a0, ($t2) # Play the 5th of the Scale

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

j major7thChord

major7thChord:

li $v0, 37

la, $a0, ($t1) # Play the Root Note of the scale

la $a1, ($t3) #Play the note for the length of the value stored in $t3

la $a2, 1

la $a3, 127

syscall

# Play the 3rd of the scale

li $v0, 37

addi $t2, $t1, 4 # Increment the root note by 4 semitones to reach the 3rd of the scale and set that note to $t2

la, $a0, ($t2) # Play the 3rd of the scale

la $a1, ($t3) #Play the note for the length of the value stored in $t3

la $a2, 2

la $a3, 127

syscall

# Play the 5th of the scale

li $v0, 37

addi $t2, $t1, 7 # Increment the root note by 7 semitones to reach the 5th of the scale and set that note to $t2

la, $a0, ($t2) # Play the 5th of the Scale

la $a1, ($t3) # Play the note for the length of the value stored in $t3

la $a2, 3

la $a3, 127

syscall

# Play the 7th of the scale

li $v0, 37

addi $t2, $t1, 11 # Increment the root note by 11 semitones to reach the 7th of the scale and set that note to $t2

la, $a0, ($t2) # Play the 7th of the Scale

la $a1, ($t3) # Play the note for the length of the value stored in $t3

la $a2, 3

la $a3, 127

syscall

j end

brokenMinor7thChord:

li $v0, 31

la $a0, ($t1) # Play the Root note of the Chord

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

li $v0, 31

addi $t2, $t1, 3 # Increment the root note by 3 semitones to reach the flat 3rd of the scale and set that note to $t2

la, $a0, ($t2) # Play the flat 3rd of the scale

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

li $v0, 31

addi $t2, $t1, 7 # Increment the root note by 7 semitones to reach the 5th of the scale and set that note to $t2

la, $a0, ($t2) # Play the 5th of the Scale

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

li $v0, 31

addi $t2, $t1, 10 # Increment the root note by 10 semitones to reach the flat 7th of the scale and set that note to $t2

la, $a0, ($t2) # Play the flat 7th of the Scale

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

j minor7thChord

minor7thChord:

li $v0, 37

la, $a0, ($t1) # Play the Root Note of the scale

la $a1, ($t3) #Play the note for the length of the value stored in $t3

la $a2, 1

la $a3, 127

syscall

# Play the flat 3rd of the scale

li $v0, 37

addi $t2, $t1, 3 # Increment the root note by 3 semitones to reach the flat 3rd of the scale and set that note to $t2

la, $a0, ($t2) # Play the flat 3rd of the scale

la $a1, ($t3) #Play the note for the length of the value stored in $t3

la $a2, 2

la $a3, 127

syscall

# Play the 5th of the scale

li $v0, 37

addi $t2, $t1, 7 # Increment the root note by 7 semitones to reach the 5th of the scale and set that note to $t2

la, $a0, ($t2) # Play the 5th of the Scale

la $a1, ($t3) # Play the note for the length of the value stored in $t3

la $a2, 3

la $a3, 127

syscall

# Play the flat 7th of the scale

li $v0, 37

addi $t2, $t1, 10 # Increment the root note by 10 semitones to reach the flat 7th of the scale and set that note to $t2

la, $a0, ($t2) # Play the 7th of the Scale

la $a1, ($t3) # Play the note for the length of the value stored in $t3

la $a2, 3

la $a3, 127

syscall

j end

brokenDominant7thChord:

li $v0, 31

la $a0, ($t1) # Play the Root note of the Chord

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

li $v0, 31

addi $t2, $t1, 4 # Increment the root note by 4 semitones to reach the 3rd of the scale and set that note to $t2

la, $a0, ($t2) # Play the 3rd of the scale

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

li $v0, 31

addi $t2, $t1, 7 # Increment the root note by 7 semitones to reach the 5th of the scale and set that note to $t2

la, $a0, ($t2) # Play the 5th of the Scale

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

li $v0, 31

addi $t2, $t1, 10 # Increment the root note by 11 semitones to reach the flat 7th of the scale and set that note to $t2

la, $a0, ($t2) # Play the 5th of the Scale

la $a1, ($t3)

la $a2, 0

la $a3, 127

syscall

#sleep

li $v0, 32

la $a0, ($t3)

syscall

j dominant7thChord

dominant7thChord:

li $v0, 37

la, $a0, ($t1) # Play the Root Note of the scale

la $a1, ($t3) #Play the note for the length of the value stored in $t3

la $a2, 1

la $a3, 127

syscall

# Play the 3rd of the scale

li $v0, 37

addi $t2, $t1, 4 # Increment the root note by 4 semitones to reach the 3rd of the scale and set that note to $t2

la, $a0, ($t2) # Play the 3rd of the scale

la $a1, ($t3) #Play the note for the length of the value stored in $t3

la $a2, 2

la $a3, 127

syscall

# Play the 5th of the scale

li $v0, 37

addi $t2, $t1, 7 # Increment the root note by 7 semitones to reach the 5th of the scale and set that note to $t2

la, $a0, ($t2) # Play the 5th of the Scale

la $a1, ($t3) # Play the note for the length of the value stored in $t3

la $a2, 3

la $a3, 127

syscall

# Play the 7th of the scale

li $v0, 37

addi $t2, $t1, 10 # Increment the root note by 10 semitones to reach the flat 7th of the scale and set that note to $t2

la, $a0, ($t2) # Play the flat 7th of the Scale

la $a1, ($t3) # Play the note for the length of the value stored in $t3

la $a2, 3

la $a3, 127

syscall

j end

end:

# Ask the user if they would like to restart.

li $v0, 4

la $a0, askRestart

syscall

# Get User Input

jal requestUserInput

jal getUserInput

beq $t7, 1, main # If user inputs 1, restart

beq $t7, 2, endOfProgram# if the user inputs 2, end program

endOfProgram:

**Tools**

My program was written using a version of MARS named MARS++. This version of MARS introduces more MIDI syscalls that allow the MARS to output up to 15 MIDI tones at once. I will link a download to MARS++ within my project submission on Blackboard.

**User Instructions**

To run the program the user first must download MARS++. After that, simply open 822563190\_BumbasiRylan\_FinalProject.ASM in MARS++ and compile and run the .ASM file. The program will start, and instructions will be given to the user on when and what to input.

**Functions Explained**

**majorScale/minorScale**: How this function works is that it takes the root note stored in register one and progresses the value of $t1 with an addi instruction on register $t2. The reason why I opted to increment register $t2 instead of register $t1 is so that I always have access to the root note pitch stored in register $t1. By creating my scale functions this way, it allowed me to easily make separate scales that were off pitch when I coded my random pitch scales in my pitchRecognitionTest section of my code.

**displayRootNotes:** How this function works is that when it is initially called it will first push $ra into the stack. Then the function will print a message that displays all the root notes, then message that requests user input using a nested jal function. After the program returns to displayRootNotes from the function it jal’d to, the function will pop the stack and restore the value of register $ra and deallocate the stack. Finally the function will return to the address where it was originally called.

**getRootNote:** This function behaves like a for loop where the condition is that the function will continue to repeat until the condition is met that the number of times that it loops is strictly less than or equal too the value that the user inputted when they were asked to select a rootNote. The reason it loops this way is because the user is able to select a root note within the rootNote array by entering the number that correlates with the index in the array that the root note is stored in. After the array is finished traversing to the index that it needs to satisfy the user’s input, it will store the value at the current index into register $t6. Lastly, when the loops break, it will jump to the exitGetRootNote function.

**exitGetRootNote:** This function will take the root note value stored in register $t6 and store it in register $t1. It does this so that when the majority other functions that plays scales get called, they will use treat register 1 as the root Note. This function will also initialize $t3 to a word “halfCount” which has a value of 500. This is done so that the MIDI instructions can use $t3 to specify the duration that they will play the requested MIDI sound.

**generateRandomInt:** This function will first use two addi instructions on register $a1 and register $v0 to generate a random number between 0-9 which gets sent to be held is register $a0. This function will loop infinitely under the condition that the random number generated is no in the inclusive range from 3 to 6. Once a number in the range is generated, the function will jump to grabRandomInt

**grabRandomInt:** This function will take the randomly generated number from the generateRandomInt function and store the value generated into register $t0. This function will then return to the $ra value that was stored when generateRandomInt was called.

**pitchRecognitionTest:** This section of my code will work by first asking the user to pick a root note. Once a root note is chosen, the program will then generate a random number. Once the random number is generated it will play a scale with a note that is off pitch that corelates to the number that was randomly generated. For example, if the randomly generated number was 4, then the program will play the scale of the root note inputted but the 4th note in the scale will be off pitch. After the program plays the scale for the user, they are prompted to guess which number note in the scale they think is off pitch and the program will tell them if they are right or wrong.

**chordMadness:** This section of my code will work by first initializing MIDI channels 1 – 5 with a piano instrument sound. After this is done the program will ask the user to input a root note. Once the root note is choosing, the user will input which chord structure they want to be played. Depending on which chord structure the user chose to play, the program will then first play each note in the chord individually in ascending order before playing all the notes all at once to simulate a chord sound.

(The rest of the functions in my program are straightforward and well commented that they do not warrant an in-depth explanation)

**Test Results**

The program works perfectly fine under that condition that the user correctly inputs values that are valid when the program prompts them to input a value. The program begins to break once the user starts to input values that are not in the range of what is requested of them. The only part of my program that properly identifies, catches and scraps invalid number inputs would be the beginning of the program under the “main” label. The program will automatically exit if a user were to input any value that was not a number value.

The rest of the 3 sections of my program all do not properly work when number values are inputted that are not within the range of values that were displayed to the users. In the Playing Scales section of my program if the user inputs a number that is not either a #1 or a #2 then the program will not play any sounds back to the user. This is because the way the code is written is that the program will try to jump to either a function that will play a major scale if the input is 1 or a function that will play a minor scale if the input is 2. If a number were to be inputted that was out of that range the program will unfortunately be able to identify which function to jump to so it will inevitably just jump to the end of the section without playing any sounds at all.

Another bug in my program occurs when the user is given the task to select a root note via user input. If the user were to input values that were outside of 0-13, then the program will break as it will try to access index points in my rootNotes array that do not exist. When the user inputs a number greater than the bounds of 13, then the program will continuously play the same MIDI sound repeatedly. When the user inputs a number value that is less that 0 then the program will play a MIDI which pitch sounds on the low end and nowhere near the pitch values in my rootNotes array.   
  
Lastly, the program will also break in section 3 of my program if the user were to input a number outside of the range when selecting a chords structure. If the user inputs a number that is not in the range from 1 to 5 then the program will break and play the same note repeatedly until it eventually stops and then jumps to the end of the program.

**References**

(Below are links to YouTube videos which helped me write some of the functions used in my program)

YouTube video that helped me write my getRootNote function

<https://youtu.be/ls4QpZD2Cow>

YouTube video that helped me write my generateRandomNumber function

<https://youtu.be/s6snbrJ3Exk>

**Development Tools**

In addition to the references above, here is a webpage on a MIDI program where I found the version of MARS++ that I used for my program

<https://karnbianco.co.uk/portfolio/mips-assembly-music-sequencer/>

**Tests Ran**

The way I tested my program to make sure it was working as intended was mainly by trying inputs in the range and inputs outside of the range. I first made sure that every input in the range of inputs that would be requested by the program would work properly. I tested this by repeatedly putting different inputs within the range of my program and I fortunately did not run into any problems. Problems started to arise when I began to run input tests with values outside of the range of the requested inputs. More detail about to results of these tests can be found under the “Test Results” section of this documentation. In summary when testing inputs that were outside of the number range the program would often break or not output MIDI sounds as intended. The last types of inputs I tested were non-number inputs in which in every case where the program requested an input, if a non-number input was entered the program would break and automatically exit.

**Demonstration**

The following link is to a YouTube video demonstrating my program with various working inputs

<https://youtu.be/1BohukF8JmQ.com>

**Conclusion**

If I had more time to work on my project, I would try to smooth out the various inputs bugs that I discussed earlier in the “Test Results” and “Tests Ran” section. I probably would done something like how I coded the main section of my program to properly identify and discard numbers that are outside of the number range. As far as discarding non-number inputs, I probably would need to spend more time learning about what tools MARS provides me to deal with situations like that.

Overall, I learned a lot about the nature of assembly programming. This project made me appreciate the tools that I have access to when coding in higher level languages such as C/C++. This program also gave me insight into how lower-level languages real reflect more closely what is going on at a machine or CPU level. When I started this class, I was very lost in the way that Assembly programs were written and the concepts of registers. Going through and finishing this project really made my understanding of assembly solid.

**Hours**

I approximately spent 30 hours programming my project.