Assignment 2 Design Document

Abstract:
The task of assignment 2 is to build sequencers, In part II, we are asked to build a more advanced sequencer than that of part 1. Therefore, instead of just playing one fixed sequence of notes which the sequencer in part 1 is capable of, I built a more advanced sequencer which is able to play multiple sequences of notes and with dynamic changes in terms of many features of the melody. The music I used for sample is the 'coffin dance' meme music that went viral recently,

1. General Structure

1.1 Data structure

The registers I used for part 2 are from r4 to r11. However, it is clear that this amount of registers is not sufficient for the need of building a more advanced sequencer. Therefore, the concept of stack and memory are used in my design.

1.1.1 The form of data:

Each note of music has a certain frequency, however these frequencies cannot be used unless they are transferred into the data form that our program recognizes. By using the equation $x = {48000}/{note\ frequecy * 2}$, I am able to find out the number of plots in half period of triangle wave of each pitch. The number of half period plots is convenient to program design as the number will be a two digit hexadecimal number, and this helps to have a faster search in memory since each offset of memory stores two bytes of data.

1.1.2 The location of data

After getting each note translated into two bytes of data, I use .data keyword to store the sequences into memory, since the memory is capable of storing long data chains. The more advanced sequencer I build for part 2 is able to play multiple sequences in memory, therefore I stored two arrays of data into memory, and named each of them.

1.2 Code Structure

1.2.1 Main body

There is a main body of sequencer in the program that executes the main logics of the sequencer.

1.2.2 Functions

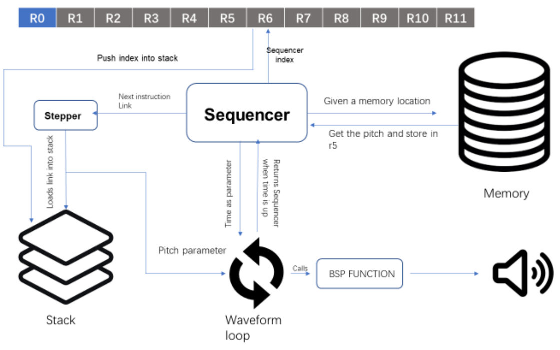
Besides sequencer, there are also other functions that outputs values we need.

1.2.3 Wave Loops

Wave loops produce waves of each specific pitch in the array. There are two wave loops where one produces a triangle wave and the other produces a square wave.

2. The Sequencer

Sequencer is the main part of the code. Sequencer is devised to search each element of the array and play each pitch out by a certain order. In the sequencer, different aspects such as time, frequency or amplitude of the sound wave is determined. Sequencer interacts with memory to get the correct pitch of array, then it inputs the pitch into the loops to play it. The sequencer is designed in a way that it can loop to itself to continuously play different pitches in an array, and each iteration of the sequencer is trackable with an index stored in R6. Here is a diagram showing how sequencer works.



3. Functions

There are several functions that achieve different tasks.

- calculate_increment: This function gets the difference in amplitude depending on the value of peak amplitude
- Sequence_changer: This function determines when to change the array by setting another base address.
- No_sound: This function gets called when sequencer determines when to pause, and this function produces quiet sound during this period.
-

4. More Advanced Features

4.1 Changing the music sequence dynamically

By storing several arrays of note sequences, the sequencer is able to arrange these pitch sequences by the composer's will. This is done by using the sequence index that is stored in R6. Each time the sequence is executed, this means that one note has been played, and therefore the index of sequence will increment by 1. With use of sequence_changer function, composer can change the address of array that sequencer loads at certain time stamps. In the code I wrote, I used the sequence changer to play the main melody (stored as section_1 in data) for 3 times, then followed by the hook of the music(stored as section_2 in data), then play the main melody again. This feature will help piece the music parts together with a more ordered way.

4.2 Changing the time duration of pitches dynamically

Since the time duration is set in the sequencer and can be changed each time the sequence runs, the sequencer can change play time of pitches freely. In the code I implemented a condition which checks the sequence index and if the sequence index reaches at a certain point, the melody will have a faster pace.

4.3 Changing the softness of sound dynamically

In the sequencer, the amplitude can be changed as well each time the sequencer iterates. This is done by checking the sequence index and change the value in r8. In the code I implemented a change in amplitude during the first time the main melody plays, as when it plays each pitch will gradually increase its amplitude.

4.4 Changing the waveform of music dynamically

Different waveforms exhibit different sound experiences. Sometimes it is useful to put different waveforms into one music to output a better sound. The more advanced sequencer can do this too. This is done by changing the stepper call in certain sequence indexes. In my code I implemented a squared wave of main melody and made it play after the hook is played.