REPORT:

Program Description:

This math matching game uses 16 cards. Eight of them being equations and the other eight being the answers of those equations. The cards are randomized at the start and the user proceeds to try and match them. Until the game ends, the user has the option to play again or quit.

Challenges:

- ***Cassidy's***
- The cards matching. At first I had them match in a different way because my shuffling was different but, after changing the shuffling method I began to struggle in getting the cards to match. In order to overcome this I debugged the matching code, used breaks, and figured out the problem. The problem was not considering, for when two cards of the same type were picked.
- The shuffling. At first I tried to use the shuffling algorithm we used in project #2 but, I began having trouble trying to implement that into this project. So, I decided to just do the shuffling in a different way. The new shuffling code uses a random number generator to randomize the shuffling, which I think works a lot better than I had first in mind.
- ***Chloe's***
- For the timer, I'm going to preface this by saying I read the instructions and discussion board wrong. I thought we had to display the time after each turn and that the timer *had* to be in real time. Because of these two misconceptions many of my problems stemmed from this such as linking the main and timer files.
- The hardest part of the timer was displaying it correctly in conjunction with the main.asm file. Before testing it with main.asm, I created the timer logic in order to work within the timer.asm file in order to make sure my idea of how to increment the seconds and minutes and how to stop the timer was working.But because of this the file was reliant on itself. Once I began to implement it with main.asm it worked but not in the way I wanted it to. Ideally, it would count the time once the game starts and only display the time once the user gave the program their second input (once again this is because I thought it had to be in real time). However, since the timeLoop is recursive, it kept calling itself, stalling the game to the point it doesn't display the board until after the time limit is up (a time limit was used to debug). In order to fix this, instead of calling the following function needed in timer.asm, I decided to use jump register to its return address in order to go back to main.asm, this worked once I switched all my registers to those that would be saved across calls.
- At this point I had everything working properly and Cassidy gave me the final files for the project. From here I realized we have been using the same saved registers, so I decided to go through each of our files to see which registers I'm able to use, specifically which s0-s7 registers that aren't being used. Once I was able to figure that out it began to display the timer after each turn and at the end the win time.

- Sound. The first challenge is seeing if MIPS has any Syscalls that correspond with sound output. There was such as syscall 33 (MIDI out synchronous); from here I had to understand how it works as well as its parameters. There are 4 parameters, one for pitch,duration,instrument and volume. Once I was able to make a tune in a separate testing file, it was a problem of implementing it with a match or try again outcome. My first attempt to do this was creating a routine inside main.asm that would use a temporary register to set as if it was incorrect or correct guesses and it would branch to its corresponding sound. This didn't work since temporary registers aren't preserved across calls. If I wanted to keep this, I had to use a save register; however, all of them were in use. I then decided to just jump immediately to its corresponding sound to the routines Cassidy already created.

Learned:

Throughout this entire project I believe my understandment on registers and how they work along programs grew. Before I got the basic idea of them but now I think I understand them way more than I did before.

Also, I used an array in this program more than once and I believe my understandment of arrays grew. Before, I honestly didn't really understand them but now I do. I now find them a lot more simpler.

Overall, through this project my knowledge of Mips Assembly and Mars grew.

Discussion:

Display Board:

To display the board, it uses a loop. It first checks if the position is marked by the matched array; if not then it continues to print the number of the card. If the position is marked then it goes on to print the respective equation or answer for the card (this is only when the user has already made a match). It uses an index to keep track of the number card that it is printing. After each 4th card it prints a newline. It knows when to print a new line because it uses a modulo (%) to determine if the number is divisible by 4. It prints a | after each number card. It is pretty simple.

Shuffle Logic:

It first loads the base address of the cards array. It starts at the end of the array and then generates a random index between the index counter and 0. It then does modulus between the random number and the index it is currently in. It then swaps the numbers between the current index and the random index it just generated. It does this loop until all the array values are visited.

Game Logic:

This first gets the user's input of the card they chose, then it uses it to get the value that it represents in the card arrays. Once it gets the value it figures out if it is an equation or answer (answers are typically bigger than 12). It then prints the respected stuff and saves the values in respected registers. After getting two cards from the user, it goes to the checkMatch function where it uses the values that were previously saved to check if they match. But before entering this function the main checks if the two cards picked were of the same type, if they were it goes to the tryAgain function. Moving on, if the cards are not the same type it checks its values, and if they match they go to the cardsMatch function which marks a matched array with a 1 for their respective index. After all this, it goes back to the main and does all the process again until all cards are matched which is checked by checking if the entire matched array is 1.

Timer Logic:

This is how we are "tracking" the time the user takes to solve the game. Within the timer.asm file there is an initialization code (startTime) that initializes 2 registers,one for minutes, one for seconds, both that are save-across call registers. This is called into main at the start of the game under 'gameStart'. Then in 'gameLoop' after the user picks a card incrementTime is called.

The way incrementTime works is that we load a temporary register (there were no more save-across call registers that we could have used) that loads 60, this is the limit for our seconds. The next thing it does is increment the register that contains the seconds, if the register equals 60, it goes to addMinute, which as you can guess adds a minute and resets the seconds counter. Additionally, we added a 'penalty'(adds 3 seconds) when the player makes an incorrect guess to 'match' with the sound. The way this is implemented is just calling incrementTime 3 times in main after the sound is played.

Sound Logic:

In order to understand the sound logic, we need to understand the use of syscall 33. Syscall 33 uses 4 parameters in the order of registers $a0-$a3 they are: pitch, duration (milliseconds), instrument and volume. Pitch works in a range of 0-127 where 0 is the lowest and 127 is the highest. Each number is a half step in the chromatic scale, if a number is outside the range it defaults to 60. For the duration, it is in milliseconds. For the instrument the range is from 0-127. In this range there are 16 families (8 instruments in each family). The only ones we will be needing to know is that 48-55 are ensemble, 96-103 are synth effects and 112-119 are percussion. Lastly for volume it is also a range of 0-127 with 0 being the quietest and 127 being the loudest.

In our program there are 3 arrays in our data section (correctMatch,incorrectMatch and win). These arrays are 16 integers that consist of 1 or 0 which correspond to different sound signals. There are 3 different routines: winSound,correctSound and incorrectSound. For all 3, the address of the array is loaded into $t8. Then in $t2, an integer which corresponds to an instrument (the ones used were: 52 ,114,100). In all of their respective routines, they then jump

and link to another routine (correctLoop or incorrectLoop) which goes through a loop for their arrays. We also set the pitch here using $a0 (incorrectSound has a lower pitch than winSound and correctSound). For both routines, we set the volume ($a3) to 127. Within these routines, another routine (playSound) is called when the integer in the array equals 1. In playSound, it checks what $t2 is, depending on whether it will go to its respective label (matchDuration/winDuration). The only difference between playSound, which acts as our default (an incorrect match), matchRoutine and winDuration, is how long the tune will play. The duration is set by parameter $a1, (after converting from milliseconds) our default playSound is 3 seconds, matching is 2 seconds and winning the game is 5 seconds. The reason the default is longer than the match, we wanted the user to feel the 'penalty' that applies to their time. In these duration routines syscall 33 is called which makes the tone play. Then the routine goes back to the caller. We keep returning to the caller after each routine.

The way this is implemented in main.asm is that depending on the outcome (match was found, user has to try again or user won) it will jump and link to its respective routine of: correctSound,incorrectSound or winSound. Once it jumps to the routine, it will go through the process stated above.

Peer Evaluation:
I did this project with Chloe and she was amazing to work with. She was considerate and got her part of the project done in a timely manner. Her part was doing the timer.asm and the sound.asm which she completed. Communication was never difficult with her and overall we worked well together. I have nothing bad to say.

Link to Video:
FINAL PROJECT VIDEO.mp4

Suggestions:
none