**Gomoku Team Project Report**

Danny Lee

CS 2340-001

Computer Architecture

December 01, 2022

Contents

* Program Description
* Challenges Our Team Had & How We Overcame Them
* What I Learned
* Discussion of Algorithms and Techniques Used in the Program
* Peer Evaluations
* Program Description

The program reproduces the Gomoku game written in MIPS assembly language. The game is played between the user and the computer where both the players take turns marking their pieces on positions of the Gomoku board. The first player to get 5 of their pieces in a row on the board (either orthogonally or diagonally) wins the game.

* Challenges Our Team Had & How We Overcame Them

Board Display

A couple of challenges we faced were related to the board display. Our first challenge with the display was that the board was not aligned. We overcame this challenge by modifying the main, createBoard1, createBoard2, printRow procedures and then adding a new procedure called addSpace. The second challenge with the board display was modifying the board so that the row number would start with ‘1’ at the bottom. Initially, our board would have row number ‘1’ at the top, but we wanted it to be the same as in the video where ‘1’ would be the bottom row and ‘19’ would be the top row of the board. With this, we also had to change the logic of the input to match with the change in row numbers.

Error and Input Validation Checking

Validating our input took a lot of time and work too as we wanted to integrate in with the existing methods to avoid repetition. We eventually decided to use a specific register as a true or false type value and assigned it to 1 during an error state and 0 during a normal state making error check a lot easier. Input validation required us learning the ASCII table again and also looking much deeper into understanding indexes within a string. There was a lot of time spent into ASCII values and eventually it just came down to conversion and running the program step by step to debug

Win Conditions

Often index values are a headache to deal with as well and with this game it was no different. Lots of time was spent running the program step by step to assure index values worked and we were able to check the “P” at the index to assure that the surrounding indexes of the win condition worked. Particularly calculating values for diagonal win conditions was much more difficult and required a lot of time to mathematically think about the association of a diagonal win in 361 individual indexes

* What I Learned

I was surprised by the efficieny the MIPS had. When I first came across this languange, I felt like there are limitation on what this language can do and only certain way can be done but after coding and brainstorming on this project, I was very wrong. When creating win condition and validation for this game. Abis looked at win condition which I thought it was simple but realized his way of validation was completely different from others and I had. From that point on, I went through the code to be the most efficient as possible with better algorithms.

* Discussion of Techniques & Algorithms Used in the Program

Board

The best way to keep the board consistent was by utilizing the labels as constant strings and spacing them based off that and for the board itself we did 19 x 19 bytes that would correspond to 361 index values and thus we would be able to get an index value on the board and store the byte in it

Win Check

When it came to win checking, we utilized the index values and just checked each individual one, branching off to the next potential win check if any of the indexes surrounding the center one did not match the pattern. This allowed us easily to check for wins

Valid and Error Check

Branching was pivotal in our program. Error check and valid number checking relied completely on ASCII values and the one Boolean value in the case of error checking. This made it easy to work with and essentially allowed us to jump from place to place to handle errors.

Sound

As an extra feature we implemented sound. To generate a sound when the user inputs, we implemented MIDI output allowing Mars to produce sound. To achieve this, li $v0, 31. What this means is that service 31 will generate a certain tone depending on the parameters the code has. So, the first parameter is a pitch which is labeled as $a0. The pitch ranges from 0 to 127 (60 being the middle C). Next parameter is the duration ($a1). The time is in milliseconds which defaults at 1000 milliseconds (1 seconds) if value is invalid. Instrument is the next parameter ($a2) which allows user to pick the instrument the tone will play on. For our code, we decided on 10. The last parameter is volume ($a3) ranging from 0 to 127.