Part 1

Brief description

The task for this part was to write 4 drivers for the VGA device. Draw point, which displays a colour on a specific pixel, clear pixelbuff which draws a black background over the whole VGA display, write character, which draws a character given its ascii code at specific coordinates and finally a clear charbuff which erases all the characters on the screen by writing 0 on all the addresses for the character buffer.

Approach taken

The draw point and draw character drivers were quite short to write, the first thing to do is check the two coordinate arguments passed to ensure they are within the correct range, calculate the correct address to store the information using the x coordinate, y coordinate and the starting address for the buffer, then store either the halfword or byte associated with each program.

The clear drivers were also both very similar, essentially the driver sets up the starting information then enters a loop where the x value is incremented each time, and if surpasses the number of pixels for the width of the vga, resets the x and increments the y. At the end of each loop, it calls either draw point or draw character and displays either black or 0, which is no character.

Challenges faced

This part was made more difficult by not reading the instructions properly and figuring out the logic to calculate the address of the points without the instructions, so a different technique was figured out using the manual.

Possible Improvements

Drivers work perfectly, nothing to be improved

Part 2

Brief description

The instructions for part 2 were to write a single driver that reads data from the PS/2 keyboard.

Approach taken

The driver first loads the value of the button pressed from the address of the ps2 register. It then checks if info in the register was valid by grabbing the rvalid bit in the data through an AND and a logical shift right. If it was a valid input, it will store the last 2 bytes associated with the button pressed in the address passed in the argument register. The driver will return whether the data read was valid

Challenges faced

The Rvalid bit was initially not getting calculated correctly because the logical shift right was forgotten, so it was keeping the valid bit in the 16^{th} bit in the register.

Possible Improvements

No improvements to be made.

Part 3

Brief description

The goal of part 3 is to design a game of Tic-Tac-Toe that takes inputs from the PS/2 keyboard, displays the moves on the VGA driver and can determine a winner or a tie, with the 0 key- restarting or starting the game.

Approach taken

This part was undertaken with a very modular approach, every part was done in a subroutine and the main loop just called the subroutines. First, 3 registers were created in memory to hold one hot encoded positions for the whole board, and just the pieces each player had made.

The first subroutine that was called was draw_game, which sets the background to black and draws 4 rectangles in order to create the 9 squares for the game.

Next was the wait_to_start subroutine that only returned to the main program when it read a 45 from the PS/2 keyboard ie a 0 is input.

The next three were the message functions that were all fairly similar. Draw_tie, player_turn and player_win all used the write_char driver to write the three messages the program can output: “draw”, “Player 1/2 wins!” and “Player 1/2 turn”, with the second two taking a 1 or 0 as an argument in r0 to determine which player’s turn it was.

There were two more draw subroutines, draw_x and draw_o, which actually only drew a diamond, but it was close enough to a circle. Both functions are passed an integer between 1 and 9 to determine in which square they draw. They follow and algorithm where they draw two lines. X simply draws two diagonal lines, stopping after the y value has decreased by 59 pixels. The diamond was a bit trickier, it started on the middle-y and leftmost pixel in its square and drew diagonally up 29 pixels, then continued down 29 pixels. It then did the same thing resetting its position except drawing down then up.

The read subroutine is where most of the program happened. It is first passed a 1 or 0 in r0 to determine if it is player 1’s turn. It polled the PS/2 keyboard until the rvalid bit was 1, where it then got “translated” by the translate subroutine. Translate turned the keyboard input numbers into numbers between 0 and 10, with ten the value being returned if the input was not a proper input. If the input was not proper, it would continue to poll the PS/2 keyboard, otherwise it would continue to the next step, determining if the chosen square was already taken. It would do this by comparing the translated value to the one hot encoded register for the whole board. If it was taken already, it would go back to polling the PS/2 keyboard, if it wasn’t it would update both the register for the whole board and the register for the player whose turn it is. The subroutine finally returns a number from 0 to 9 to either restart the game or go to the draw subroutine.

Finally, the last two functions were determine_winner and determine_tie. The tie subroutine was very simple; it just compared the register that holds the board to 0x1FF, i.e., the first nine bits to check if all nine squares were taken. A draw is only checked for after player 1’s turn because player 2 cannot be the player to play the 9th square. The determine winner was a bit longer. It took the player whose turn it was

as an input and loads that player's one hot encoded register. It then compared that register to the 8 winning combinations (three columns, three rows and two diagonals) by performing an AND operation with the three bits needed then doing a comparison to see if all three were taken. Both subroutines return a 1 if it is a draw or victory and return a 0 if not.

Challenges faced

Planning how the game would work at the beginning was a bit of a challenge because it was always going to be a bit tricky determining a winner or a tie. Making it impossible for both players to place a block on the same square was also a bit trickier than expected

Possible Improvements

This program could become far more efficient if it used interrupts instead of just polling the PS/2 keyboard, otherwise, the translate subroutine could have been dropped since the number it outputs has to be compared again, so it's a bit useless.