# ECSE 324
# Computer Organization
# Fall 2019

Date: November 16 2019

## Lab 4 Report

**Group 1**

Preyansh Kaushik : 260790402
Elie Elia : 2607959306

**Part 1: VGA - Use the VGA controller to display pixels and characters**

When fully functional this section of the lab displayed a series of numbers and characters from our board onto the connected monitor depending on the state and colored pixels.

Our main.c function begins with provided code that includes test_char(), test_byte(), test_pixel(). The method test_char covers the entire monitor display with ASCII characters, test_byte() similarly covers the entire monitor display with hexadecimal digits and finally test_pixel() iterates through the canvas to display a series of colors. The function test_char() calls upon VGA_write_char_ASM in each iteration of its nested for loop. The purpose of this subroutine is to write integer values of characters into the character buffer using the x,y values given. The subroutine uses logic derived from given sections of the boards manual to convert given values to accessible memory locations and stores the byte there.

This subroutine caused us surprisingly little issue as it is short and concise allowing the little debugging that took place to go smoothly. The method test_byte() called upon VGA_write_byte_ASM in each iteration of its nested for loop through the display. This specific subroutine served the purpose of converting the input into its hexadecimal representation and writing it into the character buffer at a specific address. It does so by separating the input byte into two divisions of 4 bytes, uses ASCII convention to convert each segment into hexadecimal and stores the two characters at the converted memory addresses obtained with logic from the instruction manual.

Our biggest challenge when writing this subroutine was the discovery and familiarization with ARM instructions we had not previously seen like “AND”, “LDRB” and “STRB”. Once we understood their function we could achieve implementation of our solution with greater efficiency. The test_pixel() method calls upon the subroutine VGA_draw_point_ASM that is meant to write the half word input into a given address in memory. Once again, we faced challenges with respect to the ARM assembly instruction functions, specifically realizing that “STRH” needed to be used took us quite some time of troubleshooting and reading the instruction manual.

Within main we also call upon VGA_clear_pixelbuff_ASM and VGA_clear_charbuff_ASM. The clear_pixelbuff serves the function of clearing all spaces in the pixel buffer through setting each space to 0. The subroutine iterates through 320x240 spaces and stores a half word of zeroes to the offset address obtained through logic found in manual. If we were looking to improve upon this section we could iterate through every fourth location and store a full word rather than a half to increase efficiency. The clear_charbuff similarly clears all the spaces in the character buffer through the setting of each space to 0 but iterates through a 80x60 frame while storing a full byte of zeroes to the effective address. Our biggest obstacle for both clear functions was implementing the loops on x and y within the clear functions themselves and understanding where to branch back to and where to do so in order to stay within the screen bounds.Once that was sorted our methods were rather straightforward to code.

**Part 2 : Keyboard - Use the PS/2 port to accept input from a keyboard**

Ps2_keyboard.s : This subroutine takes input from the keyboard of the user by checking the PS/2 data register. If this register has a valid input it returns a value of 1 and stores the input inside a char pointer. The biggest issue we faced is figuring out how to check for a wrong input as the input is a 16 bit binary number.

Main.c: The main function starts by clearing the character buffer, eliminating any previous character displayed on the screen then checks continuously for a keyboard input in correspondence to the return value of the subroutine above. This allows us to only display characters on the screen when a key is pressed after that we reference the variable “input” to get the typed character. We use the Vga_write_byte_ASM subroutine to write the input bytes on the screen at the coordinates x and y.

We increment the values of x and y after writing process by following these three rules:

-if  x>81 then this means that x is larger than the width of the image so we set it back to 0 and increment the row y.
-if y>60 then this means that we filled all the rows and we have to clear the screen and set x=0 and y=0
-if none of the first two checks applies then an incrementation of x by 3 is performed, which is the distance between the printed characters on the screen.

This part proved to be challenging because we faced issues regarding figuring out how to use the PS/2 data register and linking it to our main method.