COM1031: Computer Logic Coursework Report | Group 13

Overview of the implementation

Our implementation of the problem involved using two tables to record how each segment should be output on the 7 segment display; one table corresponding to the 21 of the characters of the alphabet, and another corresponding to the rest and an error message.

Secondly, we devised a method (alternative to the one proposed in the instructions) to store the inputs in only one register and getting them from only one table. It was based on giving dots a value of one and dashes a value of two, which were multiplied by a larger number depending on whether it was the first / second/ … / fifth signal (i.e. we multiply 1(dot) or 2(dash) by 1, 2, 4, 8 or 16) and added the result to a register. The register would then be storing a number unique representing a character, some values do not correspond to any letter in international morse code (e.g. A tilde) so, in its place in the table we wrote an error message. This, we think is easier and permitted to easily implement errors, however we never got to use it nor get_digit.S after ET failed to work with the morse generator.

Unfortunately, we only managed to code the characters **E** and **T**, this was a result of lack of understanding and will be elaborated upon in the *Challenges of Implementation*. Thus, characters which use more signals will be decoded as a succession of E's and T's (e.g. A will output an E and a T)

The implementation runs based on interrupts, waiting for either a short button press (less than 200 ms) or a long button press (more than 200 ms). We output characters on the 7-segment display by loading the segment values (declared in ***7_segment.S***) into a register, and outputting that register on PORT D where the 7-segment display is connected.

When starting implementation, we made the decision to split the code into several different files. The reason behind this being that it would make the code easier to understand and edit if the code was split into several different files depending on their purpose.

"group_13.S" is our main file where we have coded the interrupts, from this file we call the use of the other files in our project. In this file we output the characters in response to long and short button presses.

"7_segment.S" represents how each character is displayed on the 7 segment display. Each segment is defined (e.g. the top segment is referred to as "SEG_A") and

connected to a corresponding pin of port D. Each character is then defined using the bit values of each segment that is to be used. For example the character ‘r’ (CHAR_R) is defined as ~(_BV(SEG_E) | _BV(SEG_G)) where the ‘~’ symbol inverts the value. We invert the value because the 7-segment display uses inverse logic.

Challenges of the implementation

Unfortunately we only managed to successfully implement characters E and T, this is because we had many issues with the morse code generator (because of the way we implemented these letters). We wrote character definitions for all other letters into ***7_segment.S*** as we had initially planned to implement these characters also, however after much trial and error we couldn’t achieve this.

One of the challenges of implementing the morse decoder was dealing with memory space. Initially, we tried to include all of our characters within one table (including many error messages and numbers), however we found that even if this compiled, it didn’t work. As a solution to this we divided the characters into two tables of no more than 21 CHAR definitions and deleted the numbers table, which then compiled correctly.

Another issue we faced was that no members in our group had any prior experience with assembly language. This meant that we only had a short period of time to learn the syntax and instructions of the language.
Finally, our code generator did not work correctly, making us believe our code was not correct when it actually was.

Description of group contribution and member engagement

All members of the group were equally engaged with the project and contributed to the project.

Callum Seymour (URN 6426813)
Contributed by working on this report, and helping with implementing the code in general areas and comments. He also got the button to work with the 7 segment display.

Luis A. Saavedra del Toro (URN 6406355)
Contributed to the report and came up with an alternative method to store the characters. Also with implementing the code in general.

Zhi Cheng (URN 6380013)
Contributed by working on the inter-letter gap, and helping with implementing the code in general. He also commented the code.

Harry Ware (URN 61413040)
Contributed by working on the written report in writing up details of the implementation and challenge. Helped with trying to understand how the inputs and outputs work.