Introduction

The purpose of the emulator that we created is in order to demonstrate the functionality of an elevator in a hospital scenario. Everything that has been coded is an attempt to serve the community in health sector as efficiently as possible, by providing a reliable travelling method where visitors and patient can arrive to their destination but also giving hospital staff such as doctors the ability to take control of movements in crucial moments. The code can find its depth of functionality in a hospital environment given the fact that it not only administers to the normal functionality of an elevators but goes beyond that by implementing functions catered towards hospitals.

Code structure

The code that is written can be summarised into 3 major categories, and these are; initialisation. Interrupts and functions. Each of these modules have different properties that will be required to allow the functioning of the elevator.

- Initialisers – These modules are only present at the start of the code base and is responsible for create an ecosystem of pre-loaded values for the system to start on. This is an important feature as majority of the code are heavily reliant on loaded values, and not implementing these values at the start could possibly create unwarranted behaviour which could be dangerous in a real-life scenario, and a perfect conundrum is when traveling floors. The code that is implemented works on the assumption that first the numbers present in the array will be between 0 and 10 inclusive, with 0 signifying either end of the list or an empty space. If the normal garbage values are lift, the code will not be detecting the garbage value and may attempt to travel to a floor that does not exist.
- The next major code block are the interrupts, which is responsible for executing a specific code list depending of the event that has occurred, whether that is on the hardware induced or based on signals from the software side. Interrupts are vital as they act as a separate code base and run in parallel with the main code, and interfere at the appropriate time. The 2 interrupts that the elevator code takes advantage of are the timers and the number pad interface. To summarise the timer interrupt is responsible for keeping track a certain duration and executing code after a certain period, for example changing the floor of the elevator of appropriate after 2 seconds and also for to execute certain code in a 5 second timeframe when the elevator is waiting at the loop (opening and closing the doors). Furthermore, the most obvious interrupt will quite possibly be the keypad that is responsible for detecting any input from the user and executing the appropriate code based on the input that was just taken. Interrupts are the back bone of the code as they are major contributors of requesting function calls at the appropriate time

and also for detecting crucial events and executing the most suitable functions at that event.

- The final major code block can be summarised as functions and can be described as the nervous system of the code, as these blocks of code are responsible in taking information form the interrupts and executing the appropriate task. Throughout the entire code base majority of the memory changing, calculations and condition changing is conducted within in the functions, and what makes these functions quite intricate is their interdependence with each other. This is particularly obvious for a function such as “insert_in_array”, which role is to place in the appropriate floor request in an array. This simple function which is editing the memory, relies on 2 functions called insert_in_array_up and insert_in_array_down, which in return relies on the functions shift_to_right.

Component Mapping

We can link all the code to the design specification of the project and how they interact each other provides a solution for the user in a hospital scenario.

Moving

The first thing we need to look at is how we make the elevator move in the intended manners. First a timer interrupt under the name Timer0OVF has been setup in order keep track of the 2 second interval that is required for the elevator to travel. After the 2 seconds the timer function will issue a timer interrupt which will be responsible for calling multiple functions such as change_floor function, check_floor_reached and create_binary_patterns. These 3 functions are important when changing the floor levels but also displaying the floor levels. By calling the change_floor function and create_binary_patterns functions right after each other ensure the user gets the most updated information on led panel, and thus also creating the illusion of the bar filling up or down when the elevator is moving. The change_floor function utilises movement value and the values that are stored in the request arrays in order to determine in which direction to move, and which direction to give priority to.

Furthermore, the in order for the elevator to operate(move) in the manner of the users wish, a second interrupt in the form of the keypad has been setup in the self-looping function named pin_main. This main function takes in all inputs from the hardware side and identifies what the exact input was using the convert function. If a valid number this subfunction will eventually call the function convert_end_continiue. Convert_end_continiue is important as it will call the function insert_in_arrays, which will place all concerning numbers in their respective array which the change_floor and check_floor_reached will utilise in order to move in the appropriate direction.

Stopped

The previous stated function of change_floor is important in the regard of what the elevator will do if there are no more request left. In the case there is no requests the change_floor will skip the entire process and thus will not change the current floor thus ensuring that the elevator does not change as long no one has requested for the elevator. At the end of 2 second travel interrupt counter just like how we call the function to change the floors and update the LED lights, we also call a function called check_floor_reached which starts a lot of functions when we have determined that the lift has reached a certain floor. This is important as it first imitates a lot of certain functions, firstly the check_floor_reached will initiate the 5 second timer that will pause the elevator at the floor, then it will remove the requested floor and finally will call the open-door function.

The check_floor_reached will initiate the floor reached flag, which in turn initiates the 5 second counter. During the first second the open-door function will be called, after the first second we check the time and initiate the 3 second holding period where the motor is turned off. Finally, after four seconds the close door close door function will be called again which will start the motors again for 1 second. After the 5 second timer is finished the motor will be switched off and also switches off the floor reached flag and switched off the motor, thus leading the timer function to start using the 2 second timer again.

Finally at the end of every change_floor function, this respective function will call the create_movement_message which always provide an updated information of the current floor and the next stop.

Emergency

In order to enact the emergency situation, we will still use the keypad interrupt pin_main. The pin_main function will identify whether symbol or number has been pressed. After it is confirmed that a symbol has been entered the function will branch into the symbols function, and then using the number rows and columns we can confirm what symbol has been entered. Once the symbol function confirms that a star has been entered, we further branch out to the function star. The star function will then first clear the entire array using the set_0_request_array, then calls the insert_in_arrays function which will be responsible for placing the 1st floor in the request arrays. Finally, the emergency mode flag is set off which will alert the insert_in_array function. With this function insert_in_array seeing this flag will stop any additional floors from being added into the arrays. This status flag also alerts the create_movement_message which will make message to display the emergency message. If the star is pressed again the emergency mode is switched off which allow the elevator to operate normal.

Bonus Feature

The bonus feature we have implemented is for code blue. Once the hash is pressed, the status flag for code blue is set off. The pin_main will detect the hash and branch to hash function which first clears all request and then first number that is pressed is

added to the request array. Once the code blue is flagged the insert_in_arrays function will skip all other requests. This ensures that the elevator travels immediately to required destination without any interrupts. Finally, the create_movement_message detecting this flag will display a specialised code blue emergency message. After the elevator reached the particular floor the flag switched off automatically allowing the elevator to operate normally.

Key Functions

There are very key functions that have been implemented through out the code in order to allow the behaviour of an elevator be emulated on the ATMEGA board. The key functions are:

- Timer0OVF – This function is very important as it is responsible for executing all functions and displaying any behaviours that is related to time. For example, Timer0OVF when not at a requested floor will continuously activate a 2 second timer, which at the end of this 2 seconds will call the appropriate functions that will change the floor if there are any requests, check if it currently on a requested floor and update the current floor LED light so that it will always corresponds to the current floor. Furthermore, this function is also responsible of initiating the second timer when the elevator has reached a desired floor level,
- Pin_main – Pin_main is a function that is in constant loop waiting for an input to be received from the keypad hardware end. This would be considered a major function as it is responsible of enacting the most crucial section of elevator operation and that is, for a user to be able give inputs through the keypad. The pin_main's job will then identify what type of key was pressed and then branch accordingly. For example, if a number was pressed branch to the branch called "number" and call a function that will add the request into an array. If a symbol is pressed, pin_main will branch into the symbols and determine which symbol was pressed (hash or star), branch to the appropriate branch and execute code in that branch.
- Create_binary_patterns – A simple function, but gives the program the ability to display the current floor level on the LED lights. This function relies on constant values of all the binary patterns which are preloaded in the order of their respective levels under the name LEDvalues. By adding the current floor of the elevator with the initial address of the LEDvalues we can get the appropriate pattern to display for the current. Because 9 and 10 cannot be displayed with 8 bits we call special functions to display these particular LED pattern.
- Insert_in_arrays – A very important function that has 2 major roles, which are placing request in the intended order and also determining whether a number should be placed in request array. This function is highly fragile as it is highly dependent on other functions working properly for it to also perform as expected. The first step the function takes is determining whether there are

any special conditions or flags active that require us to skip insertion, and this includes whether elevator is in emergency mode or whether the specialised code blue is active. If the function determines that these flags are active it will immediately exit as a manner of not inserting the request. If the function sees that there are no status flags, it will then proceed deciding whether to play request in the up array or the down array. This is done by comparing the current floor and the request, and if the request is determined to be higher than the current floor, we place it in the up array and if the request is lower place, it in the down array. Depending on the which array the request is to be placed will call a certain function. These functions are:

- Shift_right – The shift_right function is called by the functions stated below. This function requires the functions insert_for_Arrayup and Insert_for_Arraydown to have replaced at the certain position and already have the address of that position to be stored in the Y pointer. The right function is called when a request needs to be inserted in the middle of an array. The key to this function is to always have 2 registers where the register 18 will acts as a storage for the previous value of the array and 19 will act as the value after the aforementioned value in r18. The values in r19 are stored in the next address. This loop ends when the array reaches the end which is marked by 0.
- Insert_for_Arrayup - Called when the request needs to be placed in the upper array. The function will loop through the entire array, until it finds the appropriate position. The appropriate position is whether the number is at the end or whether the request number is between 2 numbers. If the position of request is in the middle of 2 numbers this function will call the shift_right function.
- Insert_for_Arraydown - Called when the request needs to be placed in the down array. The function will loop through the entire array, until it finds the appropriate position. The appropriate position is whether the number is at the end or whether the request number is between 2 numbers. If the position of request is in the middle of 2 numbers this function will call the shift_right function.

- Check_floor_reached – This function is responsible for checking whether the elevator has reached a requested floor. This is done by loading both the up array and down array and check whether the first value of either of these arrays is equal to the current floor. If function has determined that a floor has not been reached it will continue as normal but if has reached a requested floor it will call a certain function. If it is the request is in the up array it will call the shift_up_values or if the request is in the down array shift_down_values which will be responsible in removing the said request floors. Furthermore, after the function ahs determined that a floor has been reached it will initiate the open-door sequence and also switch on the flag for the floor_reached.

        - shift_up_values – Used to remove a requested floor in he up array, which is done by using 2 pointers Y and Z. One pointer will (Z) will always be 1 data address higher than Y. The function works by moving data form the Z pointer to the Y pointer. The function will also reset the second counter for the timer to use for the 5 second delay.
        - shift_down_values – Used to remove a requested floor in he down array, which is done by using 2 pointers Y and Z. One pointer will (Z) will always be 1 data address higher than Y. The function works by moving data form the Z pointer to the Y pointer. The function will also reset the second counter for the timer to use for the 5 second delay.
- Change_floor – This function is responsible for changing the current floor according to the situation. The function will first check whether there any flags set such as the emergency_clearance_mode to know whether not to move. After this if there are no requests in the up array or down array the function will skip through to end otherwise it will continue. The direction of movement is determined by the stored value movement. Using this information, the change_floor function will either call the increment floor function or the decrement floor function.
    - increment floor – Goal to increase the floor level by 1, by loading the current floor increasing the value and storing it back in memory.
    - decrement floor – Goal to decrease the floor level by 1, by loading the current floor decreasing the value and storing it back in memory.
- Create_movement_message - This function is responsible by displaying the appropriate message on the LED screen. To display the appropriate message the function will first check the appropriate flags. For example, if we need to display the emergency message, the function will first check the emergency mode status flag and display it if appropriate. If none of special status are flagged, we continue with the normal display of current floor and net floor.
- Finally, we have a set of 3 functions that are quite similar in functionality. These are:
    - Set_open_doors: Sets the appropriate flags for the door's status open doors (1), close doors (0) and hold doors (0). Starts the motor.
    - Set_close_doors: Sets the appropriate flags for the door's status open doors (0), close doors (1) and hold doors (0). Starts the motor.
    - Set_hold_doors: Sets the appropriate flags for the door's status open doors (0), close doors (0) and hold doors (1). Stops the motor.

Bug Fix

The first thing that has to be done is to identify and isolate the error or bug that is occurring in the code.

There are 3 major ways we can detect these edge cases, and these will be:

- By creating a checklist which allow us to identify any easy occurring error. This way the fix can be implemented almost immediately. But the hardest problem will be identifying the edge cases that cannot be identified as easily.
- Secondly will be running the code in a controlled environment and allowing different people with different perspectives. This sorting of testing can allow us to different combinations of input and usage which may help reveal deadly bugs that could usually go undetected.
- Finally, another way we can test any bugs we is inputting extreme values that may reveal other bugs that usually won't appear when using expected values.

Extending the Code

Extending the codebase for new features or functionality requires careful planning, below is a detailed guide and best practices to help you navigate the process smoothly. Firstly, you need to understand the existing codebase, it is highly recommended that you read through the Code Structure part in this document and the main.asm file. Secondly, define the new requirement clearly which means you understand the new requirement well, it can be done by writing user stories and acceptance criteria. Thirdly, if you are using github, it is worth making a new branch to avoid contamination code. Fourthly, implementing the new feature, it should follow the code rule and code structure. Fifthly, it is time to do code reviews and documentation. After that is testing the code to check if the new feature works and will not influence the other function. What's more, check if it satisfies the acceptance criteria.

If the above guidelines still leave you confused, please have a look at the following example.

Example: It adds the new feature about using '*' button for emergency, pushing the emergency button will let the LCD show the 'Emergency call 000' message. Below is how this new feature was done successfully.

1. Understand the existing codebase: Review the existing codebase, focusing on the three parts which are initialisers, interrupts and function.
2. Define requirements: Make requirements for pressing '*' button the LCD will show an emergency message.
3. Create new branch: Create new github branch.
4. Implement the new feature: Basically, set up the interrupt for '*' button, after that set up the port for the LCD, write the interrupt function for the LCD to display an 'Emergency call 000' message.
5. Code review and Documentation: Maybe ask other team members to review your code or review by yourself, improve the code and add comments to the code.

6. Testing: Build up the code to flash the board, press the ‘*’ button to check if the LCD shows the emergency message.

Optimizing Performance

Considerations:
It is important that to avoid overly optimising the code, code should be readable so that it can be maintained and understood easily. The optimisation should focus on the code which is hard to read and time complexity is large. In other words, we should optimise the truly needed code.

Recommendations:

1. Choose the low time complexity algorithm. For array sorting, the quick sort is better than the bubble sort.
2. Apply the low time complexity data structures.
3. Minimise the .dseg and .cseg memory usage.
4. Store frequently used data.
5. Use more efficient instruction. For example, it is quicker to do xor eax, eax than do mov eax, 0