

**School of Computer Science and Engineering**

**Faculty of Engineering**

**UNSW Sydney**

**COMP2121 Project:**

**Monorail Emulator Design Manual**

by

Omar Al-Ouf & Ihor Balaban

Report submitted as a requirement for the Microprocessors and Interfacing course (COMP2121)

Submitted: October 28, 2018 Student ID: z5037275

z5229133

# Contents

[1. Introduction 1](#_Toc527566640)

[2. Data Structure 2](#_Toc527566641)

[3. Functions and Algorithms 3](#_Toc527566642)

[3.1 Procedure 3](#_Toc527566643)

[3.2 Algorithms 4](#_Toc527566644)

[3.3 Macros 4](#_Toc527566645)

[3.4 Interrupts 4](#_Toc527566646)

[References 6](#_Toc527566647)

Chapter 1

# Introduction



**Figure 1. Flow chart of the functions.**

Chapter 2

# Data Structure

The registers are defined as below, using .def:

**Table 1. Data structure.**

|  |  |  |
| --- | --- | --- |
| Name | Register | Function |
| to\_stop | R2 | Flag register to check if the monorail stops at the next station |
| rolling | R3 | Flag register to check if the motor is running |
| speed | R4 | To control the speed of the motor |
| to\_stop2 | R5 | Flag register to check if the monorail stops at the next station |
| need\_stop | R6 | To check if motor should start running at beginning |
| index | R7 | To count the characters in the station names |
| trans\_time | R8 | Temporary register that stores the travel time between two stations |
| stop\_time | R9 | Register that stores the stop time at any stations |
| keys\_entered | R10 | Calculate number of the keys entered, also works as a counter |
| stat\_num | R11 | Status register that stores the number of stations |
| stat\_count | R12 | Counter that helps to represent the current station |
| step | R13 | To show the step the program reaches |
| lcd\_temp | R14 | Temporary register that stores the ASCII value of a number or character |
| flag | R15 | Flag register to check if the value has been displayed |
| temp | R16 | Temporary register |
| row | R17 | The current row number |
| col | R18 | The current column number |
| mask | R19 | Mask for column/row during scanning |
| temp2 | R20 | Temporary register |
| stop\_now | R21 | Flag register to check if the monorail needs to stop immediately |
| Timertemp | R22 | Temporary register for Timer0 interrupt |
| shift | R23 | Status register to represent which character is needed |
|  | R24 | Lower byte of the counter used in Timer0 interrupt |
|  | R25 | Higher byte of the counter used in |

The addresses are defined as below, using .equ:

**Table 2. Addresses.**

|  |  |  |
| --- | --- | --- |
| Name | Address | Function |
| PORTLDIR | 0xF0 | PL7-4 for the output, PL3-0 for the input |
| INITCOLMASK | 0xEF | To scan from the leftmost column |
| INITROWMASK | 0x01 | To scan from the top row |
| ROWMASK | 0x0F | To obtain input from Port L |
| LCD\_RS | 7 | Different pins for the LCD |
| LCD\_E | 6 |
| LCD\_RW | 5 |
| LCD\_BE | 4 |
| HOB\_LABEL | 0b00100000 | High order bit for symbols |
| HOB\_NUM | 0b00110000 | High order bit for numbers |
| HOB\_CHAR | 0b01000000 | High order bit for characters |

The data segment is defined as below, using .byte:

**Table 3. Data segments.**

|  |  |  |
| --- | --- | --- |
| Name | Bytes | Function |
| name\_list | 100 | To store the names of ten stations, 10 bytes each, with a maximum number of 10 |
| time\_list | 10 | To store the travel time between two stations, 1 byte each, with a maximum number of 10 |
| TempCounter | 2 | The time counter in Timer0 |

Chapter 3

# Functions and Algorithms

## 3.1 Procedure

**Step 1: Entering the number of stations.**

This step obtains the number of stations. At first, the function ‘step0’ is called to display the instruction on the LCD screen. The main function continuously scans for a key press. When a key is pressed, the program will check the status and handle it. Since the data type required in this step is of a numeric type, all other keys do not work. After entering the value, the hash key indicates the end of input.

It is also important to handle any errors, such as inputs under 2, above 10 and any other irregularities. When the hash key is pressed, the function will check whether the value is valid. If nothing is entered or if the station number is 0 or 1, 10 will be stored. On the other hand, if the value is larger than 10, the last digit will be read as the input. For example, the input would be 6 if 16 is entered. The maximum number of digits that can be entered is 7, after which the hash key must be pressed to store the number into the variable *stat\_num*. The program will then automatically move to the next step, with the function ‘step1’ being called to display the next instruction.

**Table 4. Number of stations.**

|  |  |
| --- | --- |
| Button | Function |
| 0-9 | Enter number of stations |
| # | Finish inputting and store the number of stations |
| A | Will not work |
| B | Will not work |
| C | Will not work |
| D | Will not work |
| \* | Will not work |

**Step 2: Entering the name of each station.**

This step obtains the names of the stations. The chosen procedure for this step follows that of a phonetic keyboard. This enables the keys A, B, C and D to each represent a character on a number key (A for the 1st character, B for the 2nd, C for the 3rd and in some cases D for the 4th). After pressing one of these keys, followed by a numeric key, the desired character will be displayed on the LCD screen. For example, if the letter ‘D’ is required to be outputted, the A key must first be pressed, followed by 3. Similarly, for output ‘Y’, C must be pressed followed by 9. When a key is pressed, the program will check the status and handle it. In this step, since the data type required is of a characteristic value, key D will only work when keys 7 or 9 (letters S and Z) are pressed. Keys 0 and 1 will not work in this step. White spaces are inserted by pressing the star key (\*). Every time a character is inputted, it will be stored into the variable *namelist*. After entering the station name and pressing the hash key, the name of the next station can be entered. The maximum number of characters for any name is 10. If the length of the name is less than 10 characters, white spaces are used to complete the name. The function will increment the variable *stat\_count* and call the function ‘step1’ to display the instruction. Once the name of the last station has been inputted, the program will automatically move to the next step and ‘step2’ will be called. The detailed algorithm for the character inputs will be introduced later.

**Table 5. Station names.**

|  |  |
| --- | --- |
| Button | Function |
| 2-9 | Enter characters |
| # | Finish inputting and store the name of station |
| A | Represents the first character on the number key |
| B | Represents the second character on the number key |
| C | Represents the third character on the number key |
| D | Represents the fourth character on the number key |
| \* | White space |
| 0/1 | Will not work |

**Step 3: Entering the travel time from one station to another.**

This step obtains the time for the monorail to travel from one station to the next excluding stops. As in the previous step, the main function constantly scans for a key press. When a key is pressed, the program will check the status and handle it. Since the data type required for this step is of a numeric value, pressing other keys will not work. After entering a value, the hash key indicates the end of input.

As per the specifications, the maximum travel time is 10, thus any values greater than this must be handled accordingly. Upon pressing the hash key, the function will check whether the value is valid. If nothing is entered before pressing the hash key, the travel time will also be stored as 10 seconds. Entering 0 will store a travel time of 10 seconds and multiples of 10 will also be considered as 10 seconds; otherwise, the last digit will be regarded as the input. The maximum number of digits that can be entered is 7, after which the hash key must be pressed. The value will be stored into the variable *time\_list* and the function will be recalled for the next travel time. After storing all travel times, the program will continue to the next step by calling the function ‘step3’.

**Table 6. Travels times.**

|  |  |
| --- | --- |
| Button | Function |
| 0-9 | Enter number of stations |
| # | Finish inputting and store the number of stations |
| A | Will not work |
| B | Will not work |
| C | Will not work |
| D | Will not work |
| \* | Will not work |

**Step 4: Entering the stop time.**

This step acquires the stop time of the monorail at any station. As with steps 3 and 4, t the main function constantly scans for a key press. When a key is pressed, the program will check the status and handle it. Since the data type required for this step is of a numeric value, pressing other keys will not work. After entering a value, the hash key indicates the end of input.

The minimum stop time is 2 seconds while the maximum is 5 seconds. When the hash key is pressed, the function will check whether the value is valid. . If nothing is entered, or if the value entered is either 0 or 1, the stop time will be stored as 2 seconds. If a single digit entered, and is larger than 5, 5 seconds will be stored. If the value is larger than 9, only the last digit will be considered as the input. The maximum number of digits that can be entered is 6, after which the hash key must be pressed to store the input. The value will be stored into the variable *stop\_time* and the function ‘step4’ will be called to display the instruction.

**Table 7. Stop times.**

|  |  |
| --- | --- |
| Button | Function |
| 0-9 | Enter number of stations |
| # | Finish inputting and store the number of stations |
| A | Will not work |
| B | Will not work |
| C | Will not work |
| D | Will not work |
| \* | Will not work |

**Step 5: Monorail emulation.**

During this step, the monorail emulation begins. The initialisation for the Timer0 interrupt will start and a few variables will be reset. After a waiting time of 5 seconds, the motor will begin spinning at a speed of 60 revolutions per second, which means the monorail has started travelling. During the emulation, the screen continually displays the name of the next station and the travel time.

PB0 and PB1 are to simulate if a tourist wants to get off or get on at the next station. If you press one of them, the monorail will stop at the next station with the stop time you entered.

PB0 and PB1 simulate a tourist getting off or on at the next station. If one of them is pressed, the monorail will stop at the next station with the stop time registered before continuing.

The hash key simulates the case where the monorail stops half way between two stations. Whilst waiting, the motor ceases to move and two LEDs will blink at a frequency of 3 Hz. When the hash key is pressed again, the monorail will continue travelling and the LEDs will discontinue blinking.

## 3.2 Algorithms

### 3.2.1 Character Generation

**Keypad scanning:**

* Pressing the star key (\*) will store white space into the variable *namelist* and display it on the LCD screen.
* Pressing the character keys will change the value of the variable shift (A:1, B:2, C:3, D:4)
* Pressing the number keys will have a range of functions.
  + shift = 4 => more\_key
  + shift = 0 => skip\_key
  + 0, 1 pressed => skip\_key
  + 8 pressed => more\_key
  + Others => normal\_key
* Normal/more key: convert the column and row value into ASCII code of the character.
* End\_key: store the character into variable *namelist* and increase the counter.
* Convert\_end: display the character on the screen.

### 3.2.2 Emulating the behaviour of the monorail:

This is implemented in the following Timer0OVF procedure:

1. Start Timer0OVF.
2. Load and increase the temporary counter.
3. Check if need to stop immediately.
4. Check if the motor is spinning.
5. If the motor is not spinning, check if one third of a second has passed. If it still hasn’t moved, 2 LEDs will blink at a frequency of 3 Hz
6. Check if 1 second has passed.
7. Check if need to display ‘wait’ and wait for 5 seconds.
8. Initialise the monorail emulation and check if it needs to start.
9. Print the name of the next station, counting travel time. Advance to the next station if the travel time has been reached. Check if need to stop at next station – if it does then the monorail will stop for the saved stop time.
10. Reset the pointers of the variables *namelist* and *timelist* and some other variables if the monorail has travelled from the final station to the first station.
11. End Timer0OVF.

### 3.2.3 Number Operations

1. Check which step the program has reached.
   1. If step 0:
      1. Handle error inputs
      2. Store the number of stations.
      3. Call the function ‘step1’ to display the instruction.
      4. Increment the step.
   2. If step 1:
      1. Check if the length of the name is less than 10 characters. If it is, use white spaces to complete the word.
      2. Call the function ‘step1’ to display the instruction.
      3. Check if all station names have been entered.
      4. Call the function ‘step2’ to display the instruction.
      5. Increment the step.
   3. If step 2:
      1. Handle error inputs.
      2. Call the function ‘step2’ to display the instruction.
      3. Check if all travel times have been entered.
      4. Call the function ‘step2’ to display the instruction.
      5. Increment the step.
   4. If step 3:
      1. Handle error inputs.
      2. Store the stop time.
      3. Call the function ‘step4’ to display the instruction.
      4. Increment the step.
   5. If step 4:
      1. Change the flag for stopping immediately.

## 3.3 Macros

Macros were used for LCD commands and for clearing 2-byte variables.

## 3.4 Interrupts

**EXT\_INT0:**

1. Get status from SREG.
2. Change flag variable for stopping at the next station.
3. Write status back to SREG.

**EXT\_INT1:**

1. Get status from SREG.
2. Change flag variable for stopping at the next station.
3. Write status back to SREG.

# References

1. Wu, H. (2018). COMP2121 Microprocessors and Interfacing Notes. Accessed 17 Oct. 2018, from WebCMS3 CSE UNSW.