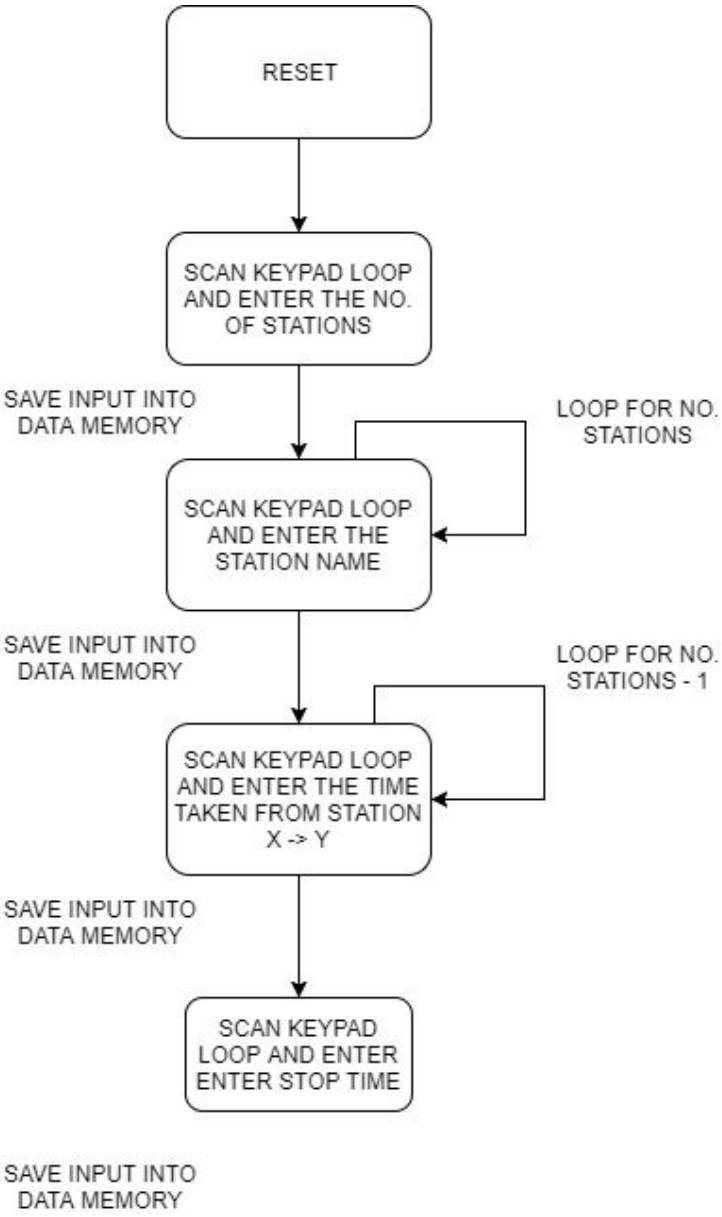
COMP2121 Design Manual *By Jeremy Chen (z5016815) and Ting Chen (z5023744)*

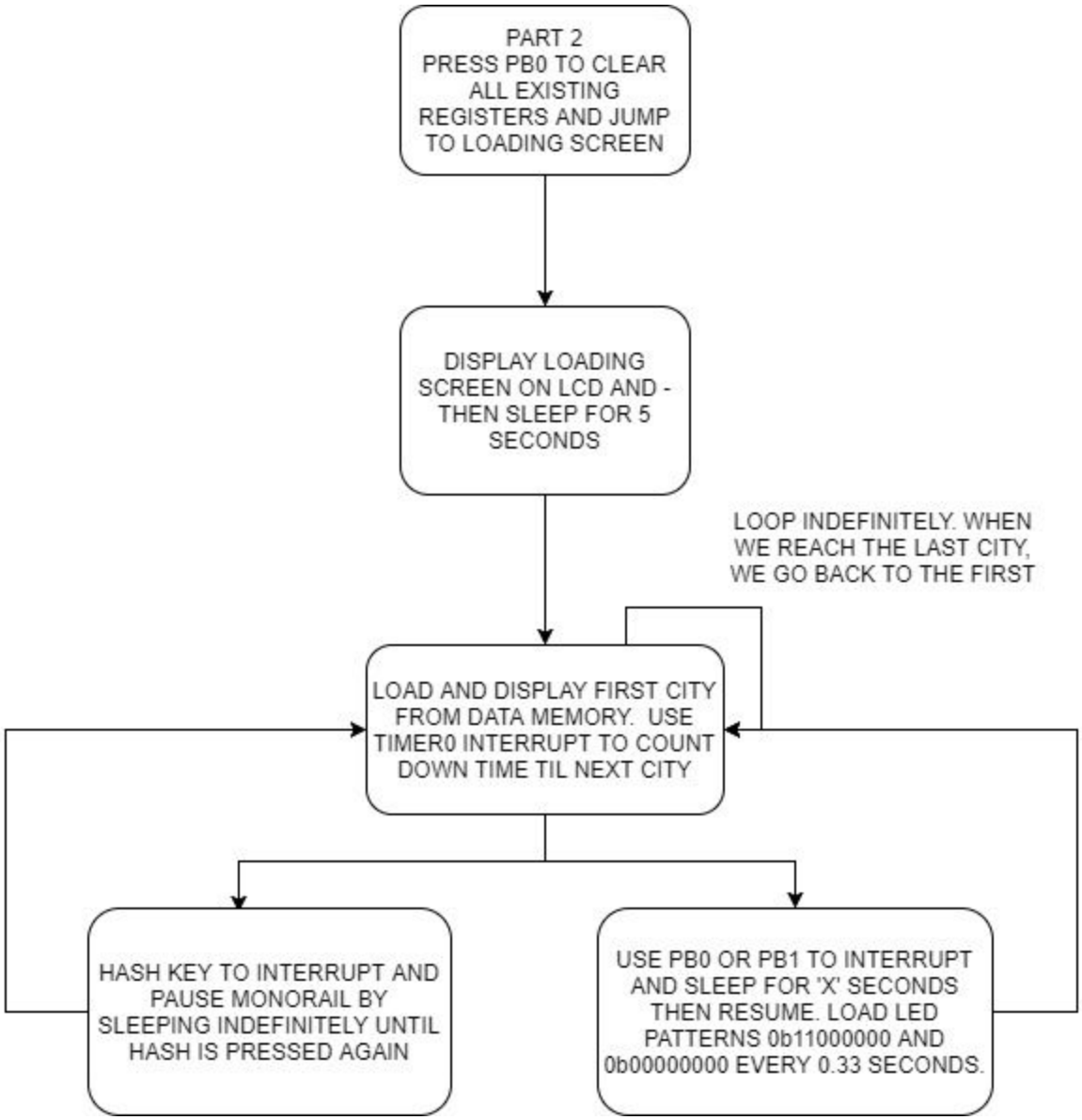
Flow Diagram

We can split the monorail emulator into two separate components. The first component primarily involves the data input / setting up the emulator, while the second component would include the execution of the emulator. Monorail Emulation begins after System Configuration.

System Configuration



Monorail Emulation



Algorithms

Some elements of our program do not work fully as intended, but we have written algorithms for what we believe should be the correct implementation for this type of emulation.

When the emulator first starts, users are required to enter keys through the keypad to input data. For the keypad reading algorithm, the keypad is constantly being scanned by monitoring voltage changes to determine what key has been pressed. RowLoop is used to find which row the key that was pressed was in, and a complementary ColumnLoop is used to find which

column the key was pressed. To account for the limitations of the smaller keypad, some modifications were made so that the keypad could express all the letters, numbers and symbols necessary for input. A more in-depth explanation can be found in the User Manual.

Pseudo Code in C For Keypad

```
#DEFINE true 1
#DEFINE false 0

while (true){
	If (Key == star){
		displayNumbers = true; //here displayNumbers indicates what keys 1-9 will do
		 Letters = 0;
		}
	Else if (key == 0){
		If (displayNumbers == false){
			Letters++;//if letters is 0, then 1-9 shows A-I, then 1 is J-R, 2 is S-Z
		Else {
			displayNumbers = false; //stop showing numbers
			}
		}
	Else if (key == D){
		Letters = 0;
		ClearDisplay(); //function for clearing lcd
		}
	Else if (key == C){
		SaveInput(); //function for saving input data
		Counter++; //counter to count how many cities data saved
		If (Counter == Total){ //if we have got all the city and time data, then break
			break;
			}
		}
}
```

After obtaining all the input data, we are then required to start the emulator and then use an algorithm to start the motor.

Pseudo C Code for Transitioning from System Configuration to Emulation

```
If (PB0 && Start == 0){ //if startflag is enabled, then pb0 does something else (let passenger
off)
	Start = 1;
	ClearRegisters(); //clear all registers used for the system configuration
	printf(“starting in 5s\n”);
	Sleep_5s(); //sleep for 5 seconds
	StartEmulation(60); //start the motor
	}
```

After we have transitioned the emulator from system configuration mode to emulation mode, we will then start running the motor / monorail.

Pseudo C Code for Starting Emulation

```
#DEFINE true 1
#DEFINE false 0

Void startEmulation(int speed){
	While (true){
	//enter a while loop, that does not break until pb1, pb0 or hash key is pressed
		If (Hash){ //if hash key is pressed, break instantly.
			break;
		}
		If (Pb0 || Pb1){ //if pb0 or pb1, only break after reaching next destination
			startMotor (speed); //start the motor at 60 rps
			Display (city[counter]); //display current city
			wait (city_distance[counter]);
			break;
		}

		startMotor (speed); //start the motor at 60 rps
		Display (city[counter]); //display current city
		wait (city_distance[counter]);//wait for monorail to arrive at next city
		counter++;
		If (counter == max_cities){  //reset to the first city after cycling through all
			Counter = 0;
		}
	}

	while (Hash == true){ //if hash key is pressed
		If (HashKey){
			Hash = false; //if hash key is pressed again, we break out of loop
			startEmulation(60); //resume the motor at 60rps
		}
		loadPattern(0b11000000); //load pattern to flash
		loadPattern(0b00000000); //clear it immediately
		Sleep (0.333);  //wait 0.333 seconds then do it again

	}

	While ((PB0 == true) || (PB1 == true)){  //if pb0 or pb1 was pressed

		While (Counter < (Stop_Delay * 3)){
			loadPattern(0b11000000); //load pattern to flash it
			loadPattern(0b00000000); //clear immediately
			Sleep (0.333);  //wait 0.333 seconds then resume
			Counter++; //increment counter until we reach stop delay x 3
		}
		PB0 = false; //reset pb0
		PB1 = false;  //reset pb1
		startEmulation(60); //resume the motor at 60 rps
	}
```

```
}
```

As shown in the code above, what would happen is that we start off with a loop that constantly cycles through and displays all the cities that have been inputted. After the monorail has run for the required time, it will then reach the next city and display that city name. This will continue until the monorail has reached the last city, which would then cause the monorail to resume at the first city.

If a hash key, pb0 or pb1 is pressed, then we break out of the loop and enter another one. In the case that the hash button was pressed, we load the pattern 0b11000000 followed by 0b00000000 and then wait for 0.333 seconds. We then repeat this indefinitely until the hash key has been pressed again. The patterns simulate a flashing effect every 0.333 seconds (3 Hz). When the hash key is pressed again, the hash flag will be set back to false and we will then recursively call the function startEmulation again.

If a push button has been pressed, then it will keep the motor running until it reaches its next destination. Upon reaching the next destination, it will exit the loop and enter another one. Similar to the hash key loop, it will load the pattern 0b11000000 followed by 0b00000000 and then wait for 0.333 seconds, this will continue until a counter reaches the stop delay x 3. Afterwards, the pb0 and pb1 flag will be set back to false and it will recursively call the function startEmulation again.

In our actual implementation, we had issues where we were unable to display the city names correctly or use the hash key. For the hash key, I believe we could have followed advice from Lecturer Wu and modified our keyboard.asm to remove the loop so that the hash key could then act as an interrupt. We could not display our city names correctly as planned, this could have been addressed by reading from an array as shown in the above pseudo code.

Data Structures

Arrays / Linked Lists

Ideally, the main data structures that would have been used for this emulator would have been arrays or linked lists. Using an array / linked list we would have stored all the data that had been inputted into the system. If using an array, our array size would be dependent on the maximum city value that would be inputted first.

When we would be required to output the data, we would read from the array / linked list, we would increment through the array after each delay period and then restart at the beginning of the array when we had reached the end. We would use a string array / linked list to store the

city names and a separate int array / linked list to store the time taken to travel between each city data.

Registers

Registers are one of the fundamental aspects of many assembly programs and ours was no exception. Since our emulator required more registers than that was provided, we split our emulator into two components (system configuration and emulation). After the system configuration component was over, we cleared all the registers and reused them for the emulation component of our program. For the system configuration component, the majority of the registers were used for functions to read the key input, others were used as counters for certain elements of the program. For the emulation, all registers were then reset and used for other purposes. These registers were used in the timer interrupt, counters, loading data from data memory,

Data Memory

A lot of variables and inputted data were stored in data memory. These included the city names and the time taken to traverse between cities as well as the stop time. While we did not fully succeed at this, the data would have then be read and then displayed on the LCD screen, cycling through the cities based on the time difference between them. We also used the data memory storage for a few byte counters and temporary counters used in the timer interrupt. Stations were stored as 8 byte data and time between stations was stored as 1 byte data.

Other Variables

We also stored many other constant variables (.equ) such as the flashing patterns (0b11000000 and 0b00000000) as well as LCD constants. Some other numbers were given names, so that the code would be easier to understand between partners. We also used them for testing purposes when trying to test out our code with placeholder values. We also used variables such as F_CPU to set the sleep duration for 1ms.

Module Specification

Module	Function	Input / Output
Timer Interrupt	The Timer 0 interrupt is used to count the number of seconds that have passed to detect when we need to transition to the next city. We also use timer interrupt to	Run Motor at 60 RPS Count Seconds Change LCD Display

	count falling edges for the motor to run at 60 rps and measure the speed.	
Push Button Interrupt (PB0 and PB1)	The Push buttons were used to interrupt from the loop that was cycling through all the cities. The push buttons would trigger a flash sequence that lasted for a specific delay period specified in the initial system configuration	Pause Motor Flash LEDs at 3 Hz
RESET Interrupt	The Reset Interrupt was used to initialize the program and all the pre-existing variables.	Clear R16 - R24 Initialize Interrupts Initialize Stack Pointer Load LCD Information Start LEDs
LCD Macros (do_lcd_data, do_lcd_command etc.)	The LCD Macros were used to display and call functions that would convert and display data onto the LCD screen. It would take a character as @0 or it would take a register value as @0 and display that.	Registers or Single Characters. Output would be the values displayed on the LCD.
Clear Macro	The Clear macro was used to clear data memory variables.	Data Memory variables were used as Input, and the output would be a cleared data memory variable.
Sleep Functions	The Sleep functions were used to sleep and pause the motor for a brief moment. They were also used to address debouncing issues. This sleep duration could range from 1ms to 5 seconds. They were often combined with a flashing functionality on the LED Display.	Pauses the Motor Temporarily Flashes LED