ARM Assembly Language: Digital Clock

Nelson Lincecum, Arye Mindell, Evan Mow

The objective of this lab is to create an assembly program that runs on a Raspberry Pi and asks the user for the current time. IT then displays the time in the terminal in HH:MM:SS form. The code has been written, and sections have been tested to ensure functionality. In the lab the code will be uploaded to a Raspberry Pi and tested. The program will be tested and evaluated at multiple times.

I. INTRODUCTION

The objective of this lab is to create an assembly program that prompts the user for the current military time. The program is to run on a Raspberry Pi and accept inputs from a keyboard. The time will be displayed as text in the terminal and will update each second. It is relevant to any website or program that displays time. The user input can be seen as the computer asking a server for the current time and updating it from there. [1]

The code cannot be written in C and converted to assembly. The time must be displayed as a single line that updates every second, rather than a new line each second. [1]

II. SYSTEM DESCRIPTION

The algorithm to display the time is shown in the flowchart Figure 1.

The main code file will immediately branch to the subroutine, setClock. This subroutine will prompt the user for the current time in HHMM format. It will then read the user input and add 00 to the seconds portion of HHMMSS. Next it converts from ascii to integer values and branches back to the main function. The pointer now enters a loop and immediately branches to the second subroutine, delay. This subroutine performs repetitive subtraction for a period that is dependent on the processors clock speed. The number of repetitions must be adjusted to reach a delay of one second. It then adds 1 to the second counter and checks if it is 60, if so it resets it to 00 and adds 1 to the minute counter. If SS is not 60, or after it has been reset the program checks if the minute counter is equal to 60. The same step is repeated for the minute and hour counters, with the hour counter checking if it is 24. If it is 24 then every counter resets to 00. After the check and potential reset the subroutine branches back to the main file. It immediately branches to the final subroutine, display. The integer values for HH, MM, and SS are converted to ascii and displayed on the terminal. The subroutine then branches back to the main function which loops back to the start of the loop. This loop continues indefinitely.

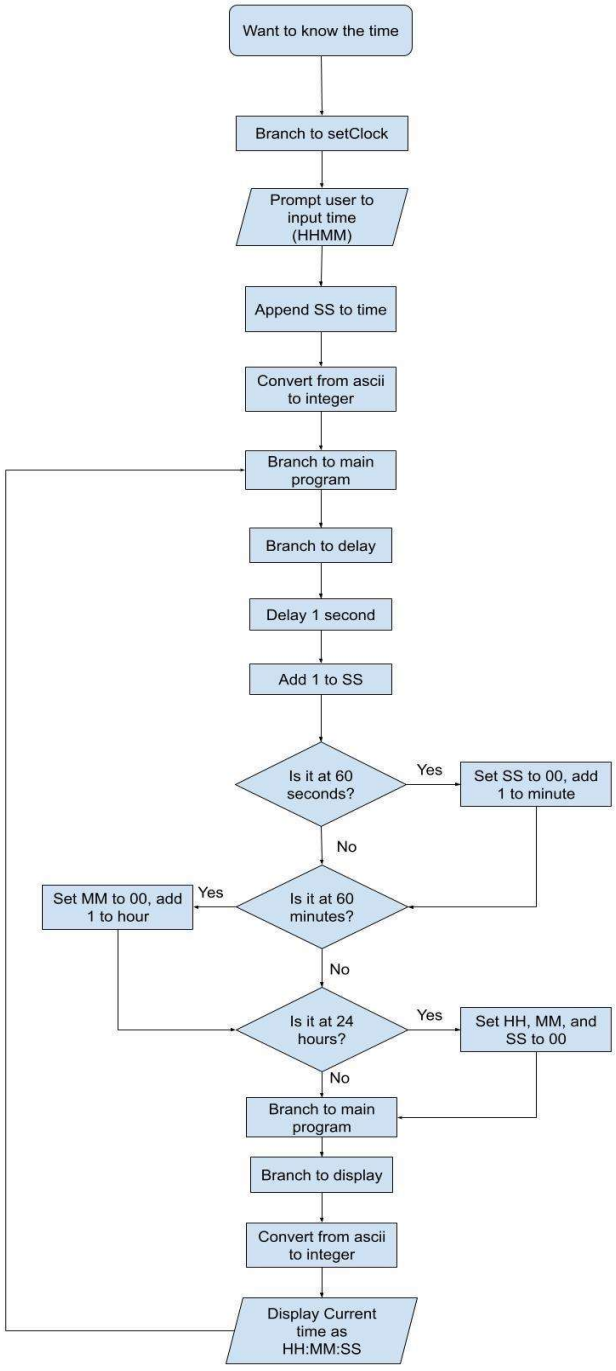

Figure 1. flowchart of the algorithm [2]

To test the clock progression, simulations were performed at times 12:59:59, 13:59:59, and 23:59:59 per the lab instructions. The hour, minute, and second values are stored in r1, r2, and r3 respectively. Figures 2 and 3 are screenshots of 12:59:59 test case.

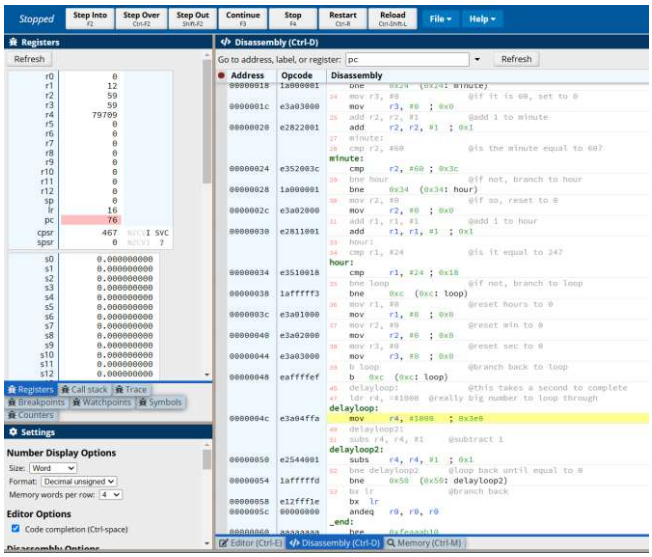

Figure 2. Simulation at 12:59:59

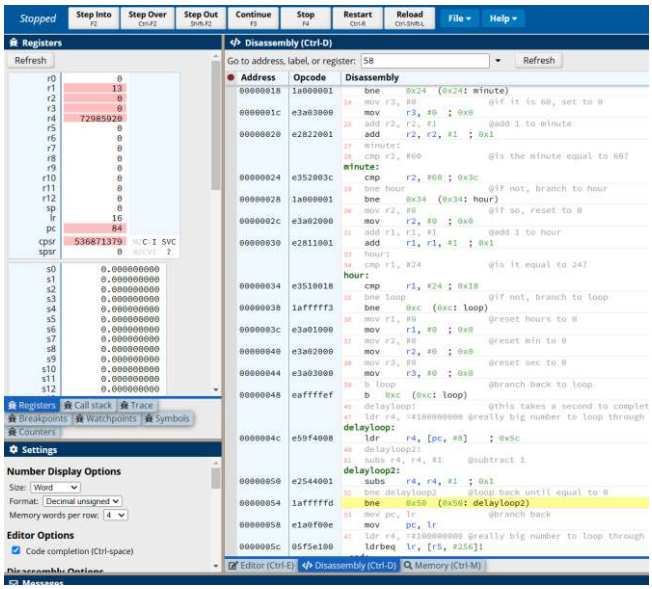

Figure 3. Simulation at 13:00:00

Figures 4 and 5 are screenshots of the 13:59:59 test case.

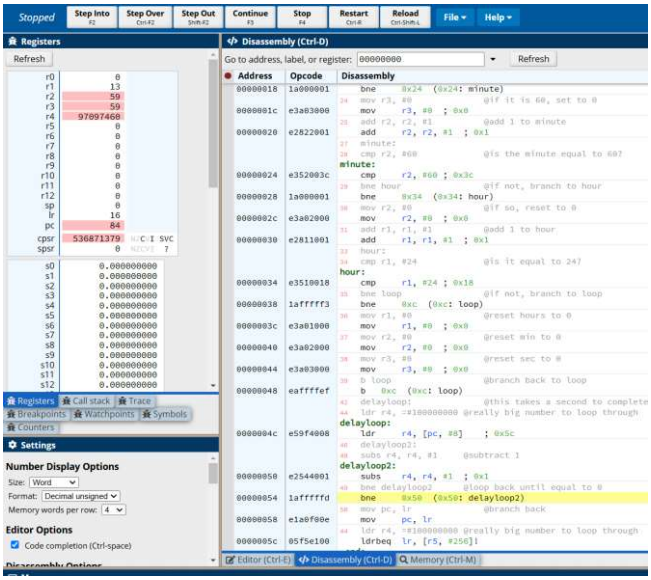

Figure 4. Simulation at 13:59:59

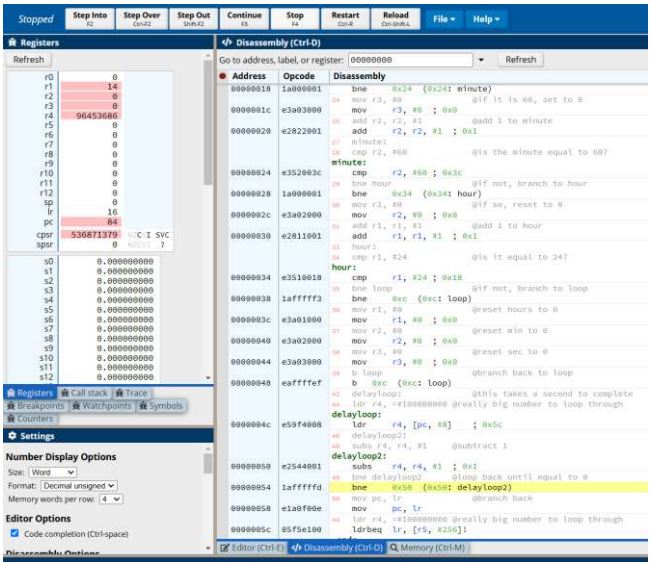

Figure 5. Simulation at 14:00:00

Figures 6 and 7 are screenshots of the 23:59:59 test case.

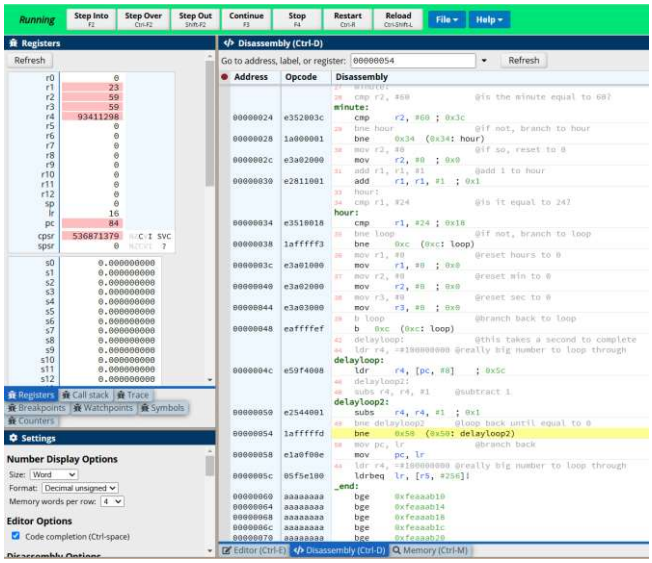

Figure 6. Simulation at 23:59:59

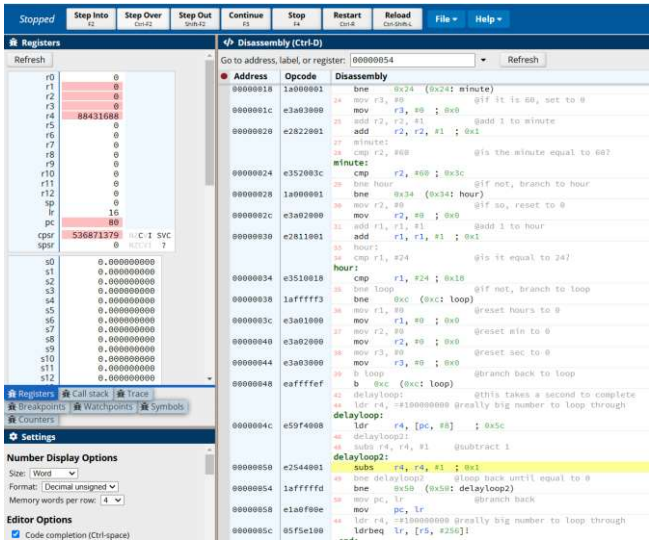

Figure 7. Simulation at 00:00:00

The main file and the three subroutines will be uploaded alongside this document as .s files.

III. EXPERIMENTAL EVALUATION

Performance Metrics:

There is no mathematical equation that can describe the performance of the clock. Instead, the performance was measured by how accurate the clock is over a five-minute period. It was also tested to ensure that it properly transitioned from 12:59:59 to 13:00:00 and from 23:59:59 to 00:00:00.

List of Materials:

- Raspberry Pi
- Monitor
- Keyboard
- Mouse
- Stopwatch
- The four .s files and the make file

Experimental Procedure:

In the lab the code was first downloaded to the Raspberry Pi and compiled with the make file. A test was then conducted to see if the delay function delays by the intended second. To test this a stopwatch was started at the same time as the clock. The results are shown in Table 1. The delay loop was then adjusted based on that data. The clock was then run for five minutes and timed with a stopwatch. The results are shown in Table 2. The 12:59:59 and the 23:59:59 test cases were then tested and are visible in Figures 8 and 9 respectively.

Results:

Table 1: Test Case Used for Adjusting Clock

Real Time Elapsed (s)	Time Elapsed on Clock (s)
40.08	30

Table 2: Time on Clock vs Real Time after 5 Minutes

Real Time Elapsed (min:sec)	Time Elapsed on Clock (min:sec)
05:01	05:00

```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ ./clockNelson
Enter the current time in the HHMM format: 2359
^C:59:10
pi@raspberrypi:~/Desktop $ ./clockNelson
Enter the current time in the HHMM format: 2359
^C:59:23
pi@raspberrypi:~/Desktop $ ./clockNelson
Enter the current time in the HHMM format: 2359
^C:59:55
pi@raspberrypi:~/Desktop $
pi@raspberrypi:~/Desktop $ ./clockNelson
Enter the current time in the HHMM format: 2359
^C:59:32
pi@raspberrypi:~/Desktop $ make
as -o delayNelson.o delayNelson.s
delayNelson.s: Assembler messages:
delayNelson.s: Warning: end of file in comment; newline inserted
gcc -o clockNelson clockNelson.o delayNelson.o setClockNelson.o displayNelson.o
pi@raspberrypi:~/Desktop $ ./clockNelson
Enter the current time in the HHMM format: 2359
^C:04:07
pi@raspberrypi:~/Desktop $ ./clockNelson
Enter the current time in the HHMM format: 1259
13:00:00
```

Figure 8: Verifying Test Case 12:59

```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ ./clockNelson
Enter the current time in the HHMM format: 2359
^C:59:10
pi@raspberrypi:~/Desktop $ ./clockNelson
Enter the current time in the HHMM format: 2359
^C:59:23
pi@raspberrypi:~/Desktop $ ./clockNelson
Enter the current time in the HHMM format: 2359
^C:59:55
pi@raspberrypi:~/Desktop $
pi@raspberrypi:~/Desktop $ ./clockNelson
Enter the current time in the HHMM format: 2359
^C:59:32
pi@raspberrypi:~/Desktop $ make
as -o delayNelson.o delayNelson.s
delayNelson.s: Assembler messages:
delayNelson.s: Warning: end of file in comment; newline inserted
gcc -o clockNelson clockNelson.o delayNelson.o setClockNelson.o displayNelson.o
pi@raspberrypi:~/Desktop $ ./clockNelson
Enter the current time in the HHMM format: 2359
^C:04:07
pi@raspberrypi:~/Desktop $ ./clockNelson
Enter the current time in the HHMM format: 1259
^C:00:04
pi@raspberrypi:~/Desktop $ ./clockNelson
Enter the current time in the HHMM format: 2359
00:00:00
```

Figure 9: Verifying Test Case 23:59

Several factors can affect the accuracy of the digital clock implemented on the Raspberry Pi. The system clock relies on a crystal oscillator, which has an inherent frequency tolerance that may lead to slight variations in timekeeping. Temperature

fluctuations can also cause the oscillator frequency to shift, affecting precision. Additionally, software execution delays due to instruction cycle variations may result in timing inaccuracies, as the delay subroutine might not be precisely 1 second. System interruptions and background processes running on the Raspberry Pi can further contribute to minor deviations in execution timing. Power supply stability is another factor, as voltage fluctuations may impact oscillator performance, leading to clock drift over time. If the clock loses 10 microseconds per second, this will result in a total loss of 0.864 seconds per day and approximately 5.26 minutes per year. To ensure the clock remains accurate to within 1 second per year, it would need to maintain an accuracy of 0.0317 microseconds per second. In percentage terms, the current clock drift is 0.001%, or 10 parts per million (PPM), while the required accuracy for 1-second precision per year is 0.00000317%, or 0.0317 PPM. These calculations highlight the level of precision needed to maintain long-term accuracy in the digital clock's implementation.

IV. Conclusion

Upon running the program on the Raspberry Pi, the clock's accuracy was assessed by comparing it against a stopwatch over a five-minute period. Initial tests indicated a discrepancy between real-time elapsed and clock time due to an inaccurate delay function. Adjustments were made to the delay loop to better approximate a one-second interval. The updated implementation showed improved accuracy, with the clock maintaining consistent time within a second over the five-minute test window.

Further analysis revealed potential factors affecting long-term accuracy, such as oscillator drift, temperature variations, and background processes. These findings highlight the limitations of software-based timing and the importance of external synchronization for long-term precision.

Overall, the lab successfully demonstrated the feasibility of implementing a functional digital clock in ARM assembly on a Raspberry Pi. The program accurately updates time in a single-line format, responds to user input, and performs real-time updates. Future improvements could include implementing hardware timers for greater accuracy, refining the delay function, or integrating an external time source for synchronization.

VI. References

[1] Dr. Nazmul Ula, "ARM Assembly Language: Digital Clock", Accessed Jan. 2025

[2] Wikipedia, "Flowchart", Accessed Jan. 2025. [Online] Available: https://en.wikipedia.org/wiki/Flowchart