Jordan University of Science and Technology

Faculty of Computer and Information Technology

Department of Computer Engineering

SOFTWARE DESIGN AND DEVELOPMENT PROJECT

MIPS Simulator

Supervised by: Eng-Shatha Alhassan

Cookie Bytes Team

Team Leader: Sadan Nsour
Team Members: Juman Alhayajneh, Roaa Batta

Table of Contents

Document Overview

This document will include all team members' main information (Member name, ID, and email) We will also be defining our meeting and documenting structure for the project.

Contact Information

Name: Sadan Fayez Nsour (primary contact)

University ID: 137019

E-mail: sfnsour19@cit.just.edu.jo

Name: Roaa Ziad Batta
University ID: 137307
E-mail: rzbatta19@cit.just.edu

Name: Juman Wasfi Alhayajneh
University ID: 136705
E-mail: jwalhayajeh19@cit.just.edu.jo

Team Structure

- Author: Juman,Sadan,Roaa
- Leader: Sadan
- Analyzer: Juman,Sadan,Roaa
- Designer: Roaa
- Timekeeper: Sadan
- Explainer: Juman
- Clarifier: Juman,Roaa
- Accuracy coach: Juman
- Integrator: Sadan,Roaa

Meeting information

Format:

- Online:

  Saturday at 9:00 PM

- Face to face

  Tuesday at 9:00 AM

Project tracker: ZOHO projects

Minutes Documentation: documented by team leader and sha

Implementation process

This application is implemented using Dart programming language, and Flutter environment, because it provides a simple platform, Beautiful UI, Quicker time to market, and faster code development.

Overview of the system:

The project entails building a CPU simulator based on a small subset of the MIPS instruction set. instead of digital design.
The implementation involves a simulator that reads MIPS assembly code from a file and progresses through the five stages of a processor (fetch, decode, execute, memory, and writeback).
A distinctive feature is the graphical representation of register and memory updates with options for both complete and step-by-step code execution.

Advantages of the system:

- *Educational advantages*: help both professors and students by providing a practical understanding of processor architecture and implementation.

- *Modularity:* dividing the processor into separate modules for each stage, making it easier to understand and manage.

- *Visualization*: the graphical updates enhance comprehension by providing a visual representation of changes in registers and memory during code execution.

- *Step-by-step Execution*: the option for step-by-step code execution facilitates a detailed examination of each stage, aiding in debugging and comprehension.

Disadvantage of the system:

- *Limited instruction set*: the project only implements a small subset of the MIPS instruction set, limiting the scope and functionality of the simulator.

- *No digital design implementation:* the absence of digital design components might limit exposure to hardware-level implementation and troubleshooting.

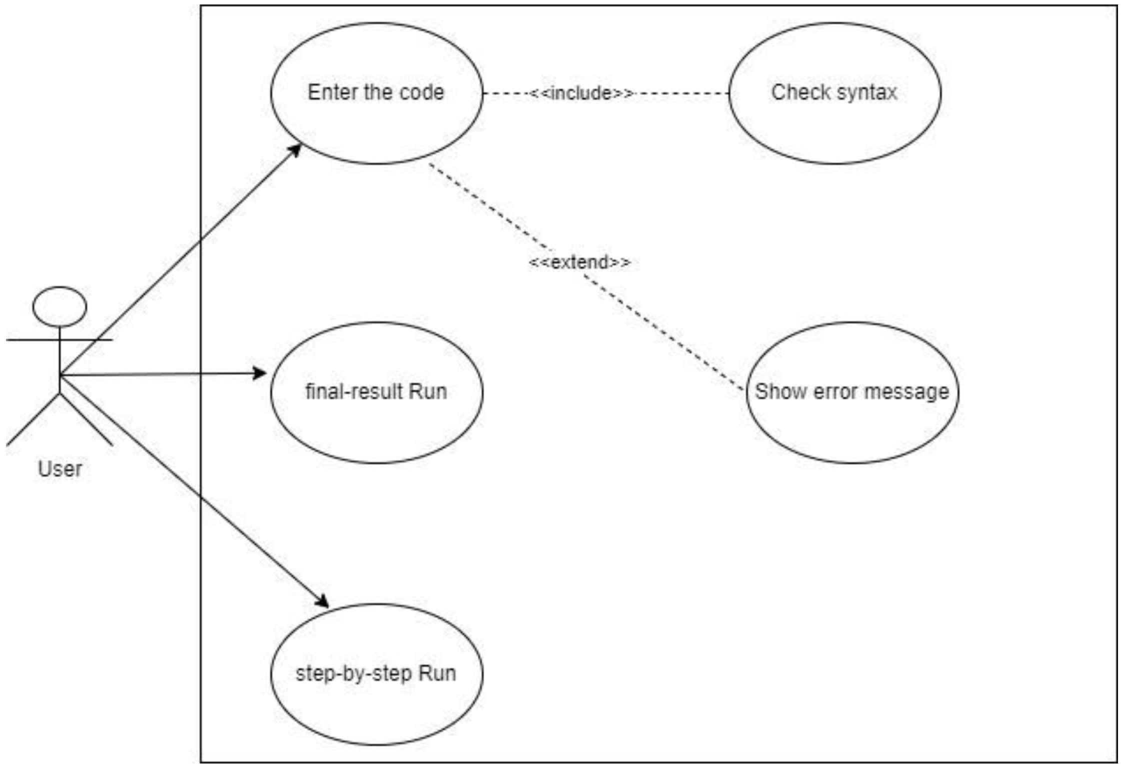
Modules

1. **user:** This module allows users to perform some operations to specific set of instructions entered by the user
2. **text editor:** This module allows users to start the coding process by writing the required set of instructions.
3. **register files:** This module is responsible for tracking the register's values and managing the processor's register set, storing, and retrieving data for processing.
4. **pipeline module:**

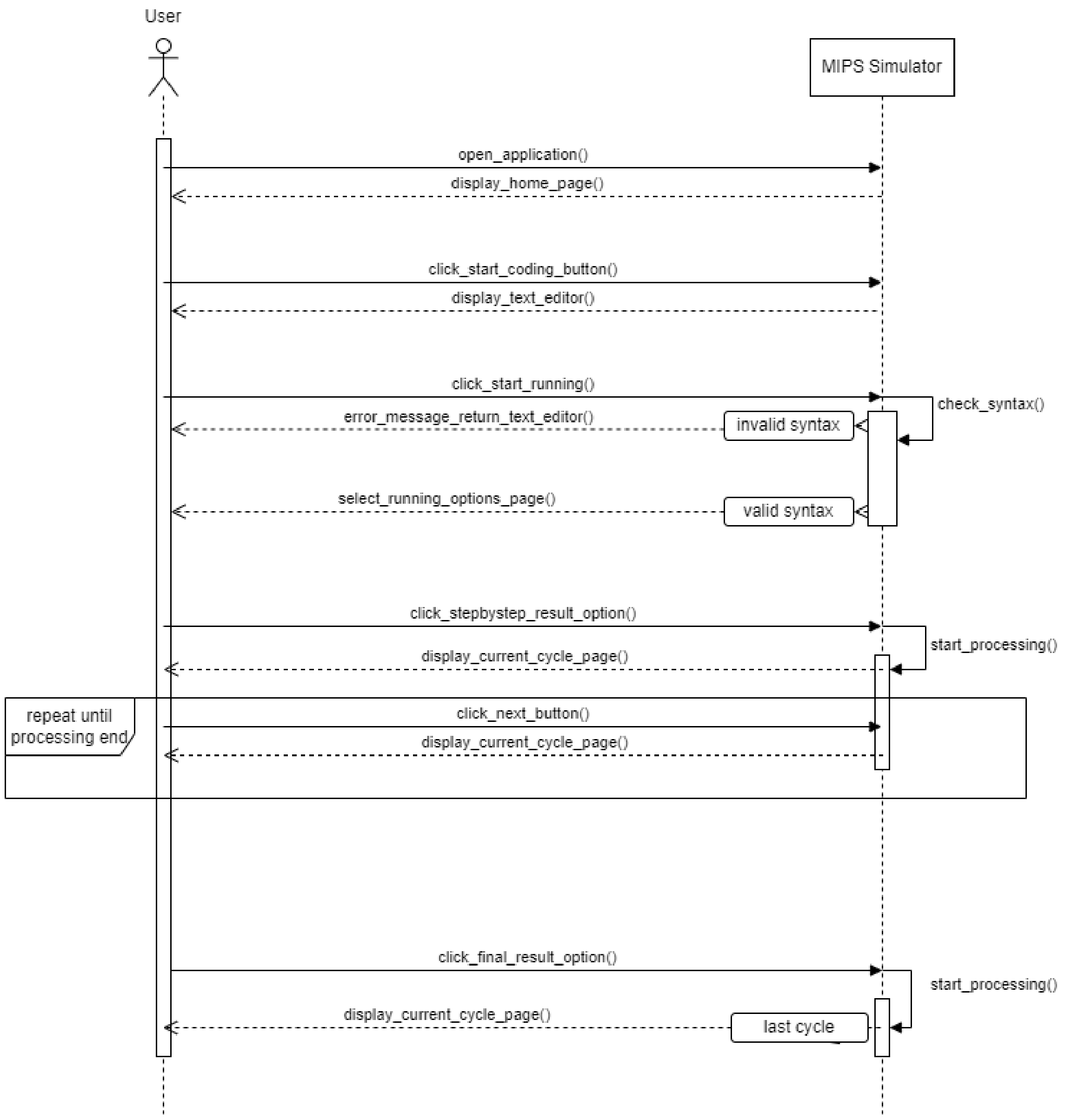
1) **Instruction Fetch (IF):** Responsible for fetching the next instruction from memory
2) **Instruction Decode (ID):** Decodes the fetched instruction, extracting the opcode and operands
3) **Execution Unit (EX):** Performs the actual execution of arithmetic and logic operations specified by the instruction.
4) **Memory Access (MEM):** If the instruction involves memory operations, this module is responsible for accessing the memory (read or write).
5) **Write Back (WB):** Write the results of the execution back to the register file.

5. **Running module:** this module allows users to view their code results.

Use Case Diagram



Sequence Diagram


User
MIPS Simulator
open_application()
display_home_page()
click_start_coding_button()
display_text_editor()
click_start_running()
check_syntax()
error_message_return_text_editor()
invalid syntax
select_running_options_page()
valid syntax
click_stepbystep_result_option()
start_processing()
display_current_cycle_page()
repeat until processing end
click_next_button()
display_current_cycle_page()
click_final_result_option()
start_processing()
display_current_cycle_page()
last cycle

Project Closure Report:

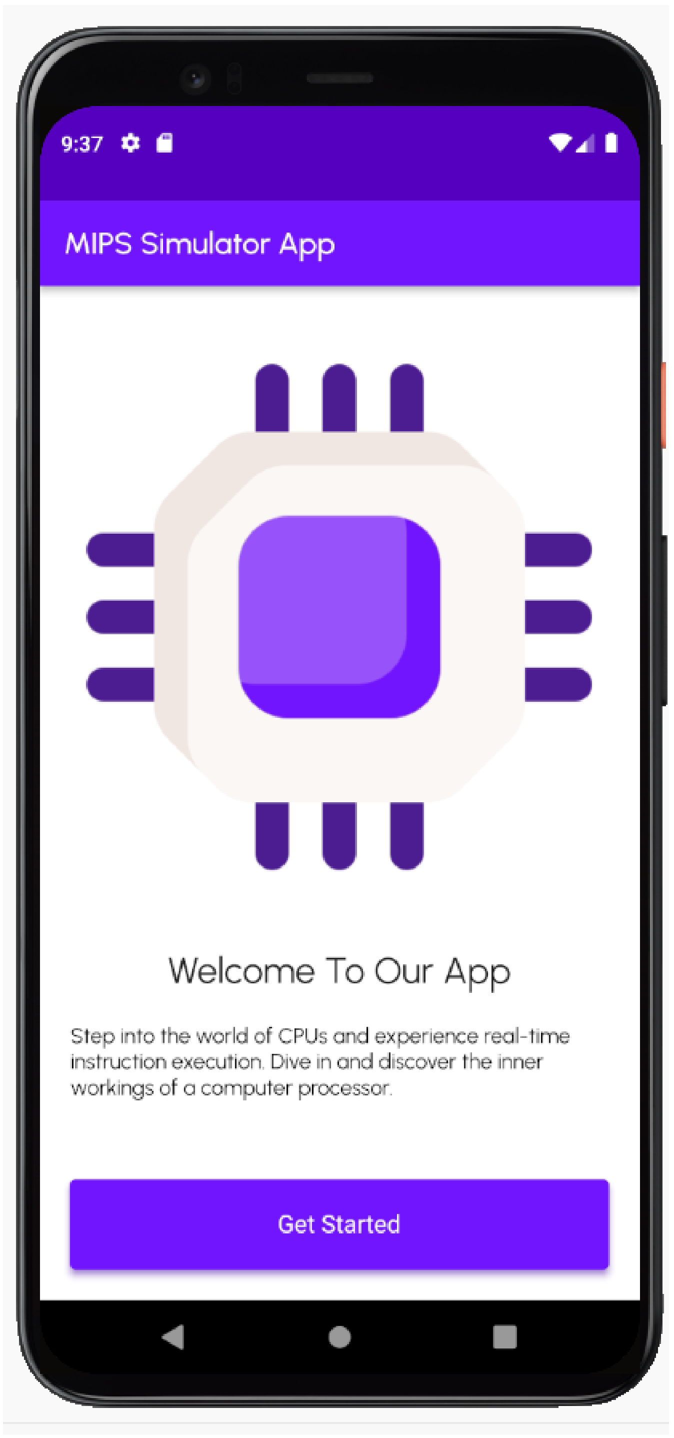
We accomplished the requirements specification of the application in a specific time but in a difficult way. We face problems throughout the duration of working on the project. We find the MIPS processor complex and difficult to work on. Working on this project using a new programming language for all of us has made the process more difficult.

however, regardless of all the difficulties we faced and the problems we solved together, we all were satisfied with our end product.

System Documentation:

To use our application (address), first of all, you need to download it from Play Store (for Android systems) or App Store (for iOS systems), and have a good internet connection to search.

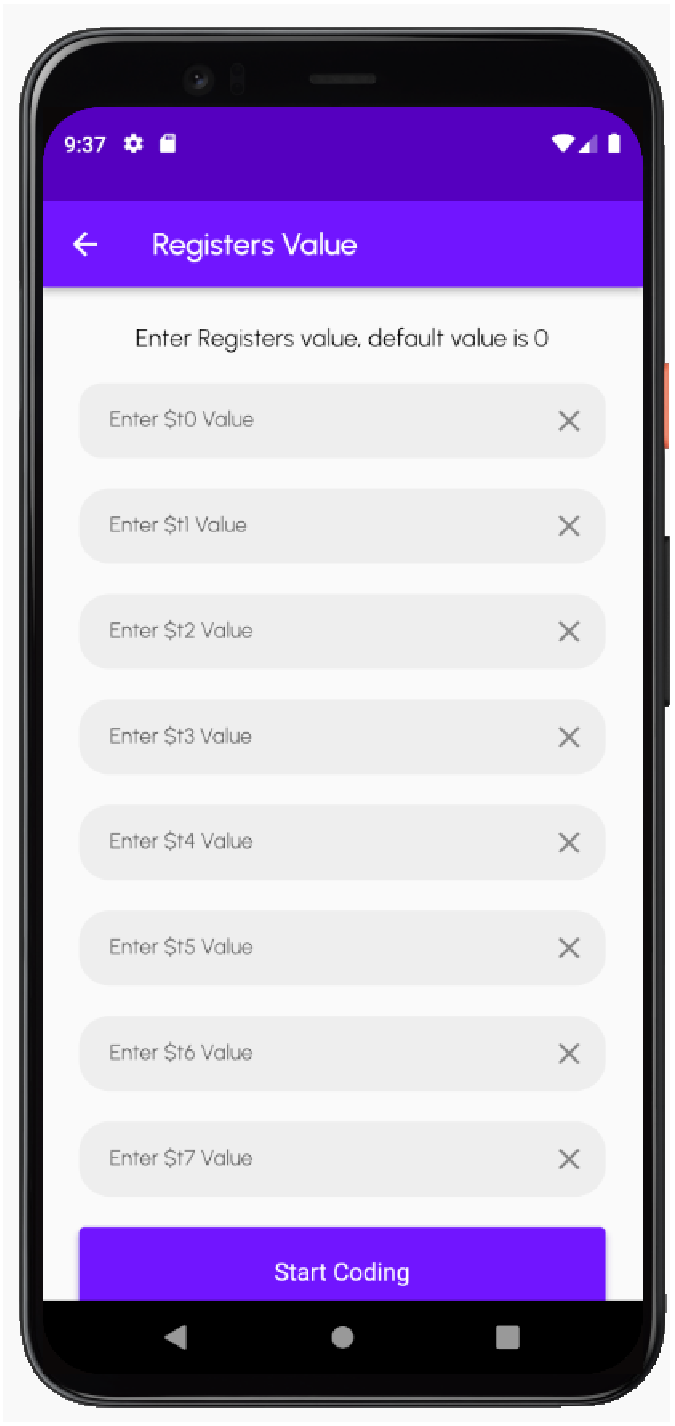
Welcoming screen :



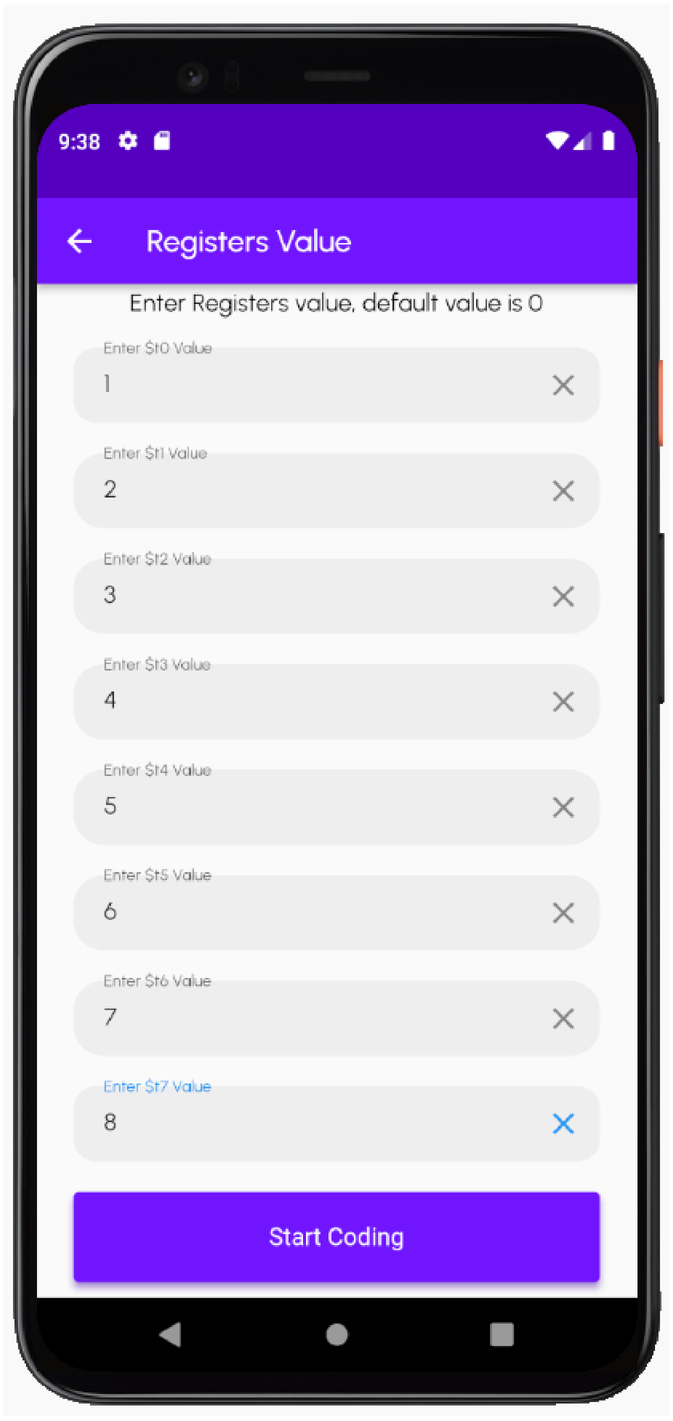
Welcome To Our App

Step into the world of CPUs and experience real-time instruction execution. Dive in and discover the inner workings of a computer processor.

Second screen ask the user to enter register values
it's not necessary to fill all the values.



We fill the values with decimal numbers.
(For now, it only uses decimal numbers)



Enter Registers value, default value is 0

Enter $t0 Value
1

Enter $t1 Value
2

Enter $t2 Value
3

Enter $t3 Value
4

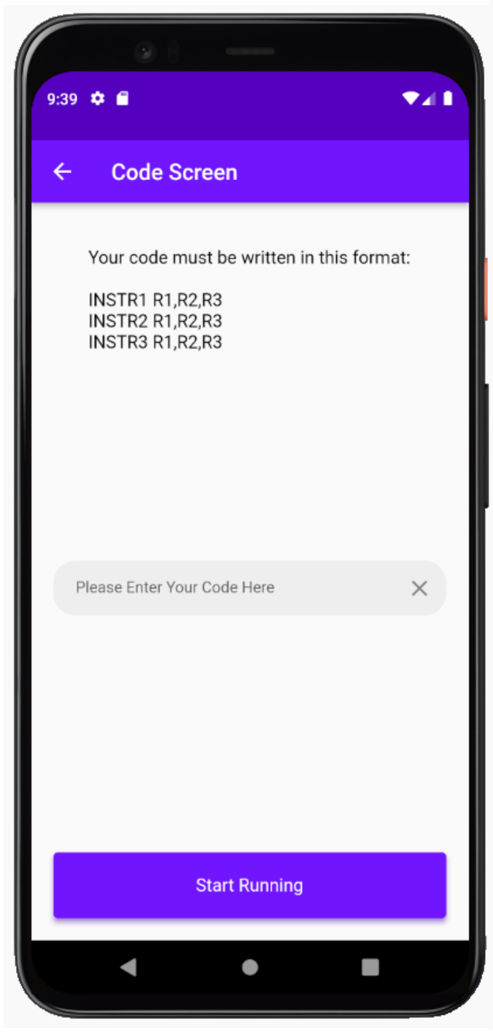
Enter $t4 Value
5

Enter $t5 Value
6

Enter $t6 Value
7

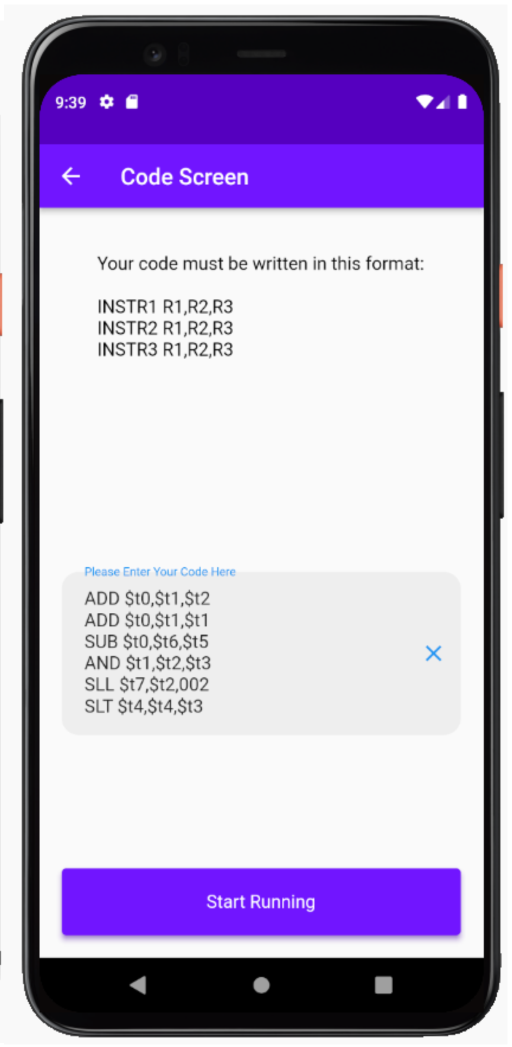
Enter $t7 Value
8

The next screen ask the user to enter the assembly code.


Code Screen

Your code must be written in this format:

```
INSTR1 R1,R2,R3
INSTR2 R1,R2,R3
INSTR3 R1,R2,R3
```


Code Screen

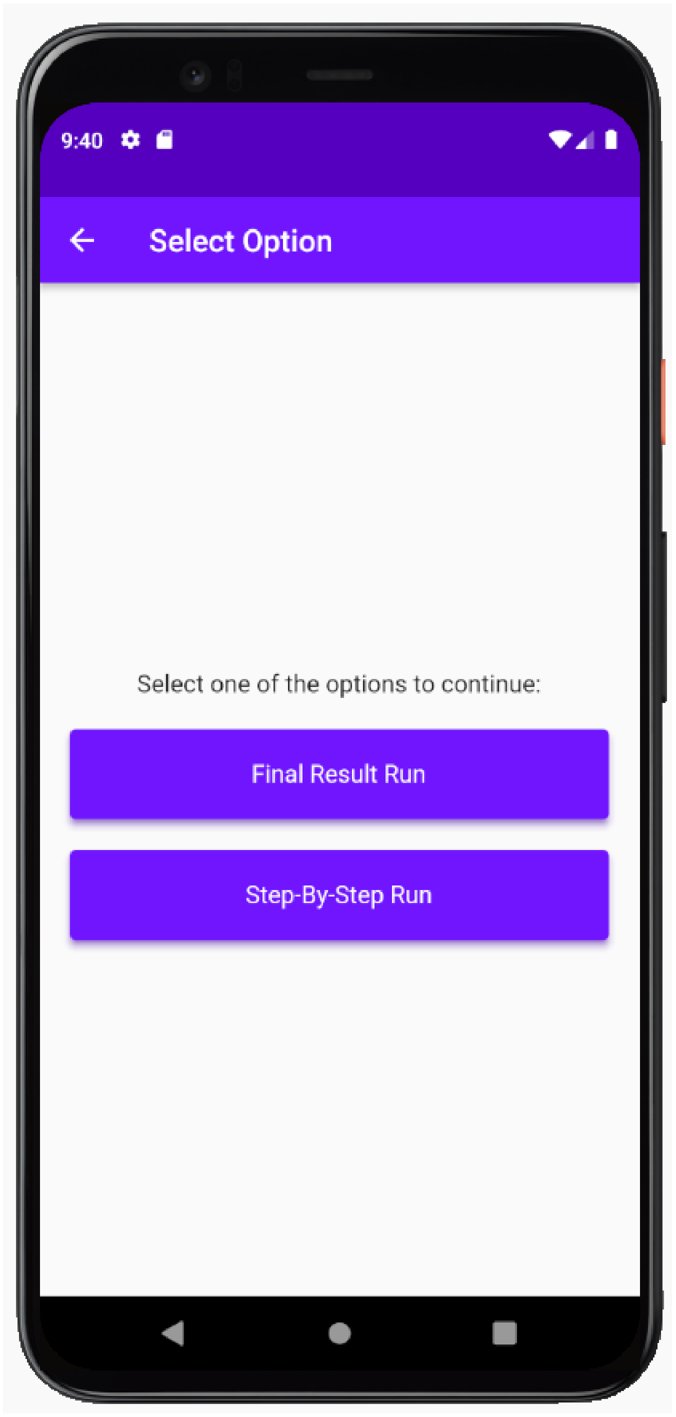
Your code must be written in this format:

```
INSTR1 R1,R2,R3
INSTR2 R1,R2,R3
INSTR3 R1,R2,R3
```

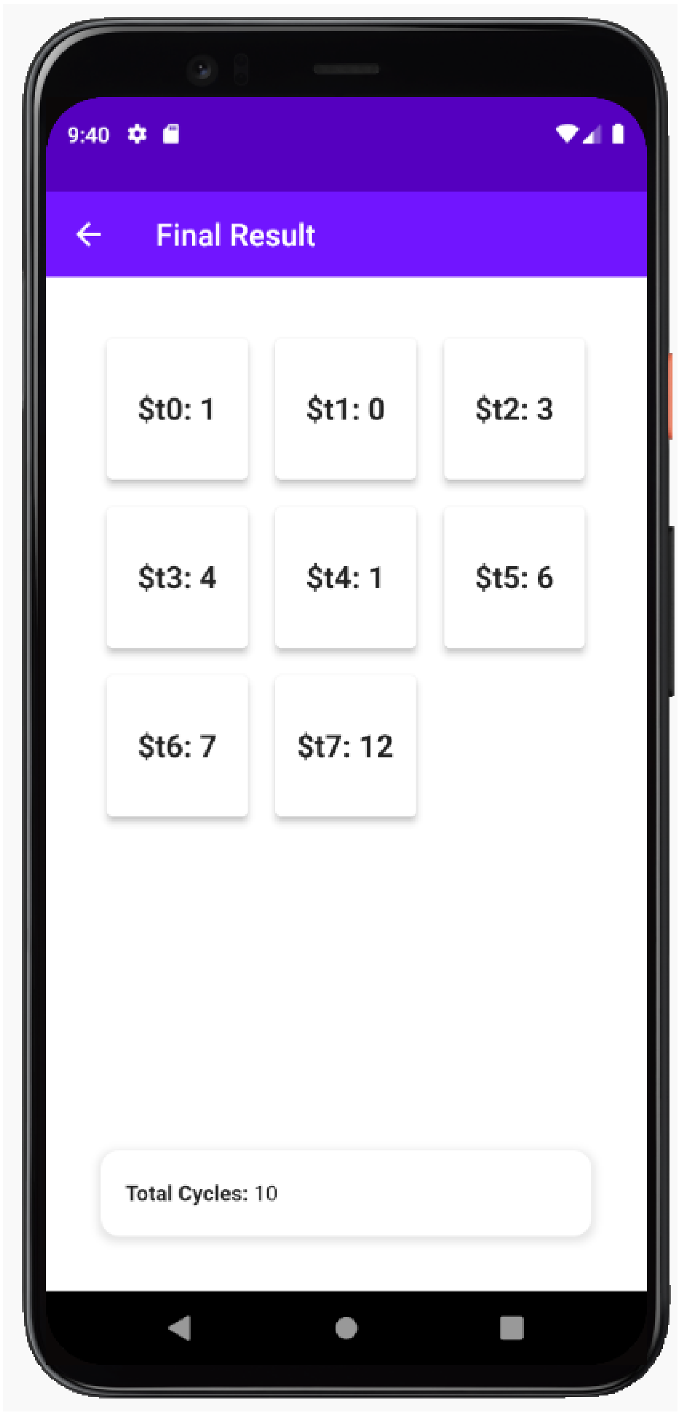
Please Enter Your Code Here

```
ADD $t0,$t1,$t2
ADD $t0,$t1,$t1
SUB $t0,$t6,$t5
AND $t1,$t2,$t3
SLL $t7,$t2,002
SLT $t4,$t4,$t3
```

After that, the application asks the user to choose between two options to run : Final result run or Step-by-step run.



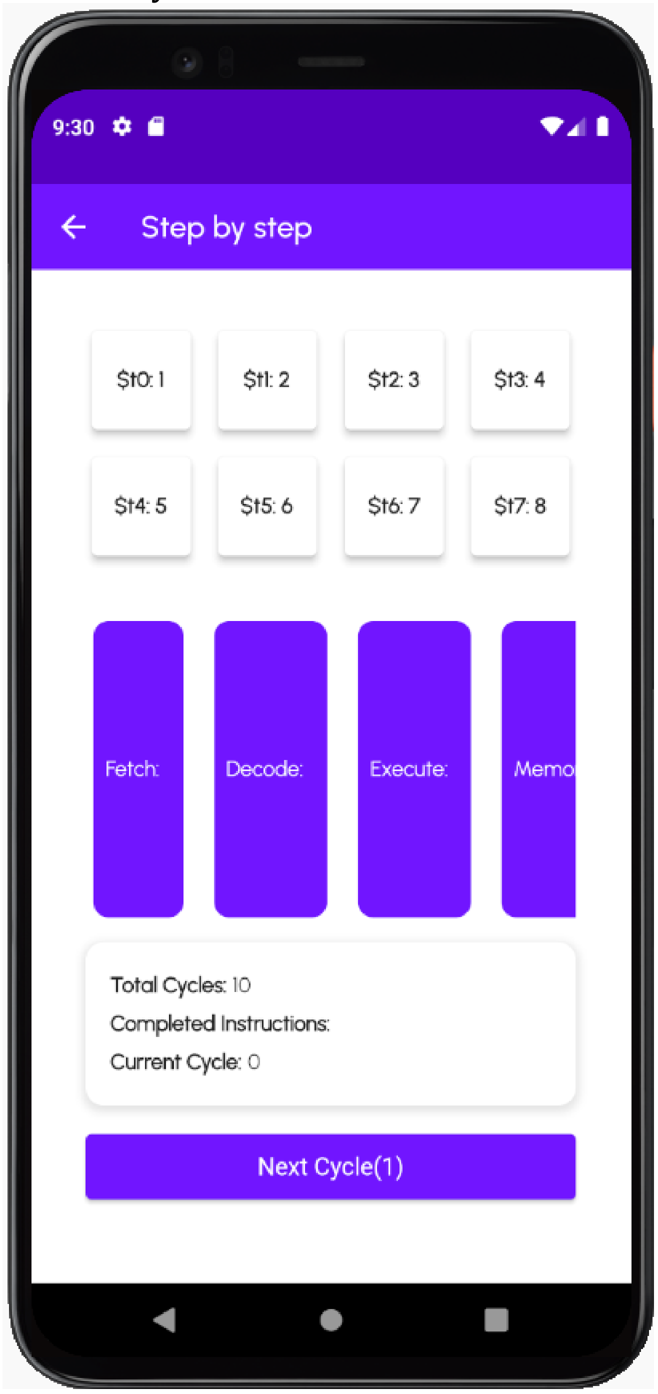
If the user chooses the final result run option the output will appear like the screen below.
This feature shows the final values of registers after the 5 stages of processing, and the total number of cycles needed to run the code.



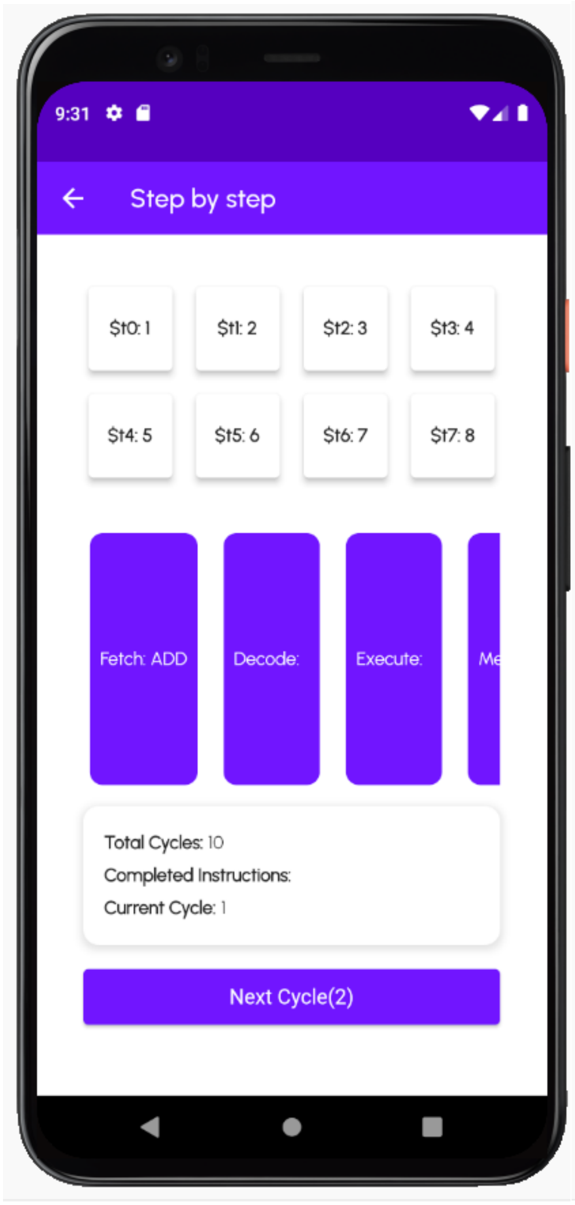
$t0: 1
$t1: 0
$t2: 3
$t3: 4
$t4: 1
$t5: 6
$t6: 7
$t7: 12

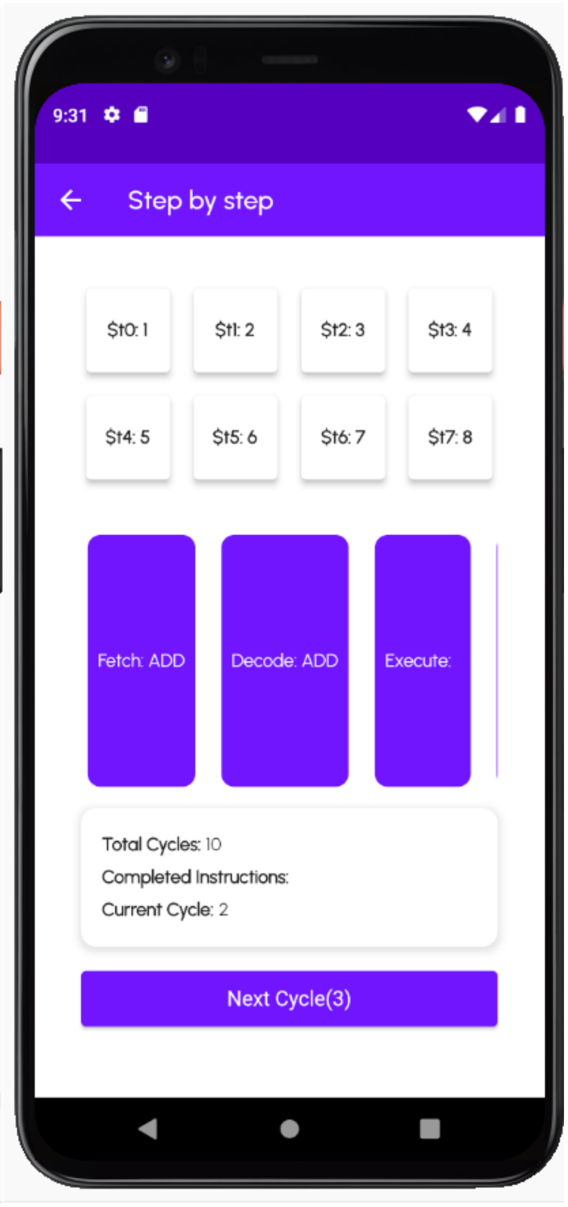
Total Cycles: 10

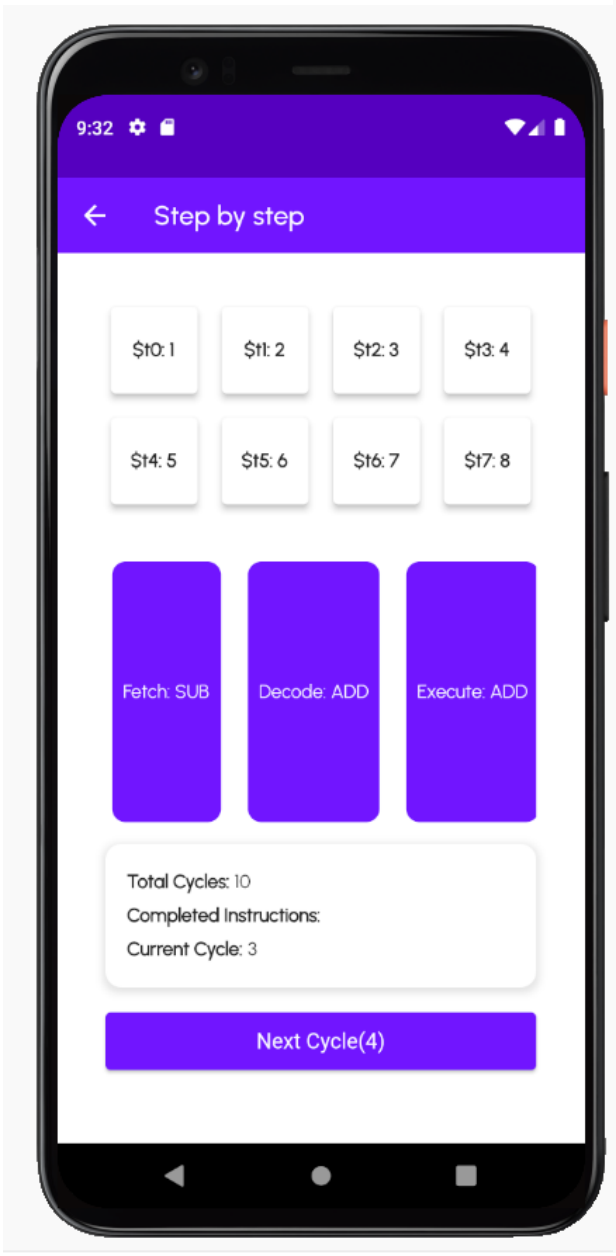
In case the user chooses the step-by-step option the output will show the changes of registers values step by step according to the current stage , and the order of instructions while entering the 5 stages .
the screen has 3 values :
Total cycles: represent the number of cycles needed to run all the code
Completed instructions: represent what instructions had done until the step user reached it.
Current cycle: counter count the number of cycles until this step.



Total Cycles: 10
Completed Instructions:
Current Cycle: 0


9:31
Step by step
$t0: 1
$t1: 2
$t2: 3
$t3: 4
$t4: 5
$t5: 6
$t6: 7
$t7: 8
Fetch: ADD
Decode:
Execute:
Me
Total Cycles: 10
Completed Instructions:
Current Cycle: 1
Next Cycle(2)


9:31
Step by step
$t0: 1
$t1: 2
$t2: 3
$t3: 4
$t4: 5
$t5: 6
$t6: 7
$t7: 8
Fetch: ADD
Decode: ADD
Execute:
Total Cycles: 10
Completed Instructions:
Current Cycle: 2
Next Cycle(3)



9:32

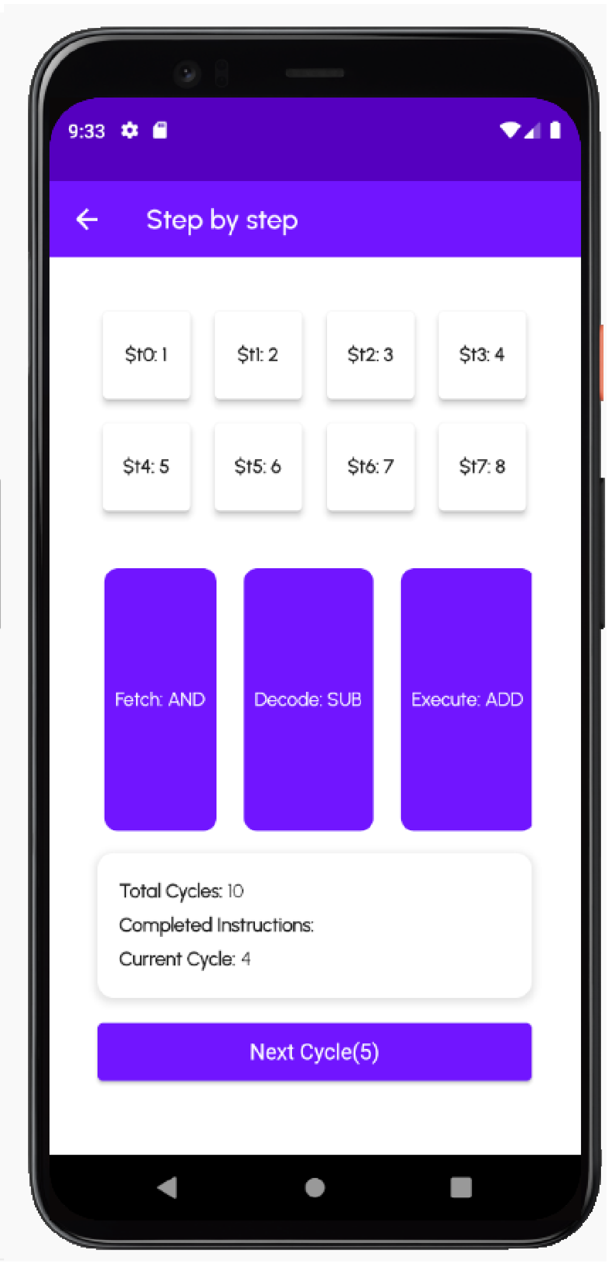
Step by step

$t0: 1 $t1: 2 $t2: 3 $t3: 4
$t4: 5 $t5: 6 $t6: 7 $t7: 8

Fetch: SUB
Decode: ADD
Execute: ADD

Total Cycles: 10
Completed Instructions:
Current Cycle: 3

Next Cycle(4)



9:33

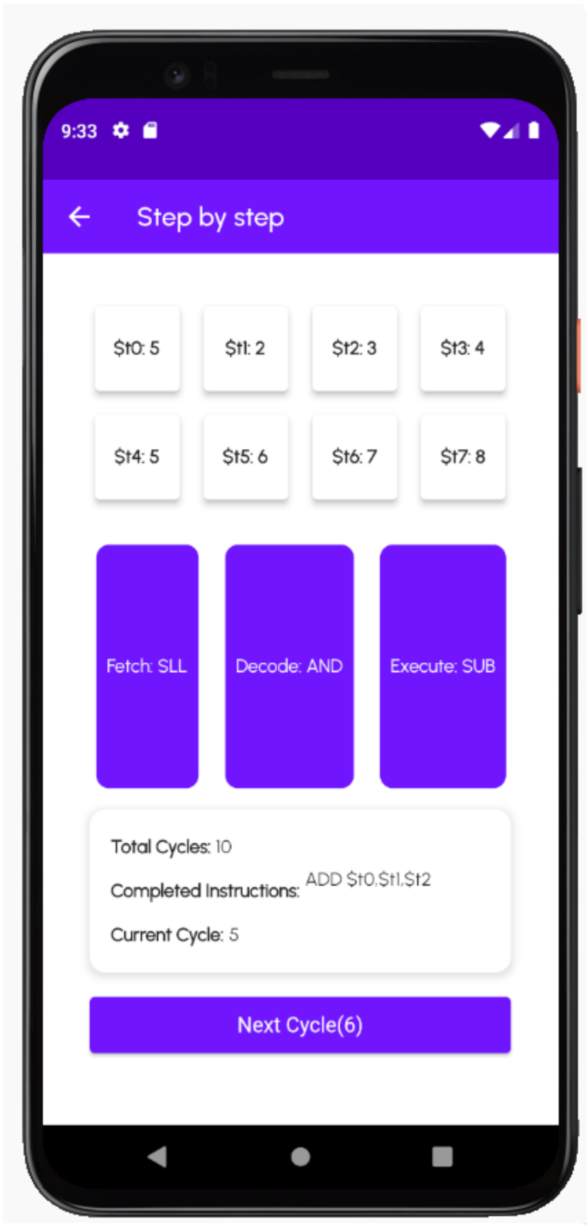
Step by step

$t0: 1 $t1: 2 $t2: 3 $t3: 4
$t4: 5 $t5: 6 $t6: 7 $t7: 8

Fetch: AND
Decode: SUB
Execute: ADD

Total Cycles: 10
Completed Instructions:
Current Cycle: 4

Next Cycle(5)



$t0: 5 $t1: 2 $t2: 3 $t3: 4

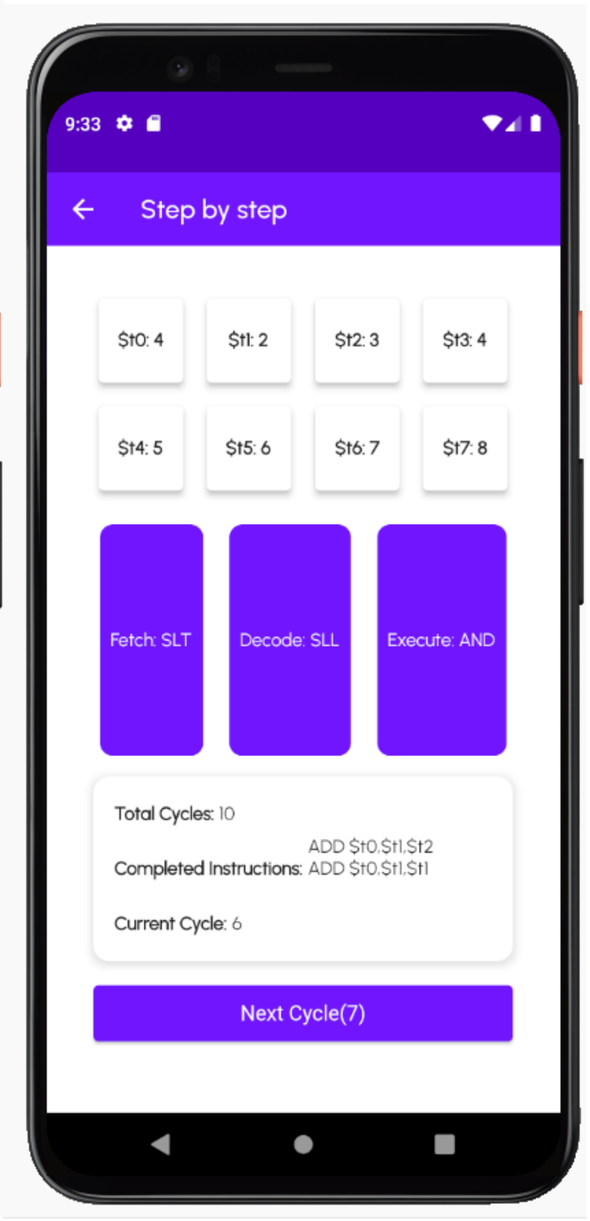
$t4: 5 $t5: 6 $t6: 7 $t7: 8

Fetch: SLL
Decode: AND
Execute: SUB

Total Cycles: 10

Completed Instructions: ADD $t0,$t1,$t2

Current Cycle: 5



$t0: 4 $t1: 2 $t2: 3 $t3: 4

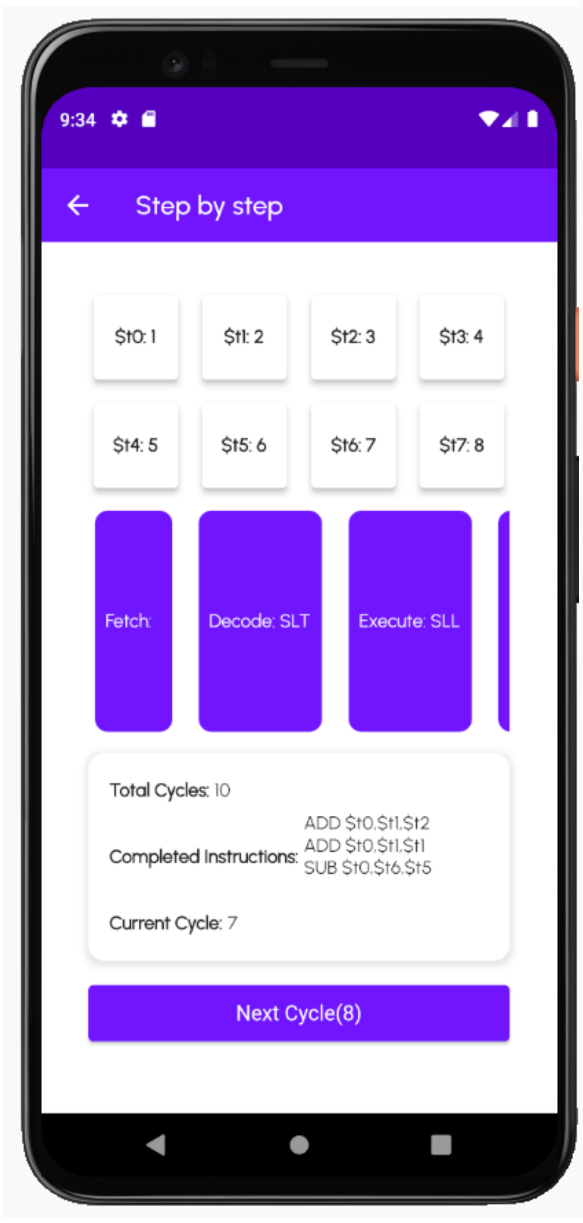
$t4: 5 $t5: 6 $t6: 7 $t7: 8

Fetch: SLT
Decode: SLL
Execute: AND

Total Cycles: 10

Completed Instructions: ADD $t0,$t1,$t2
ADD $t0,$t1,$t1

Current Cycle: 6



9:34

Step by step

$t0: 1 $t1: 2 $t2: 3 $t3: 4

$t4: 5 $t5: 6 $t6: 7 $t7: 8

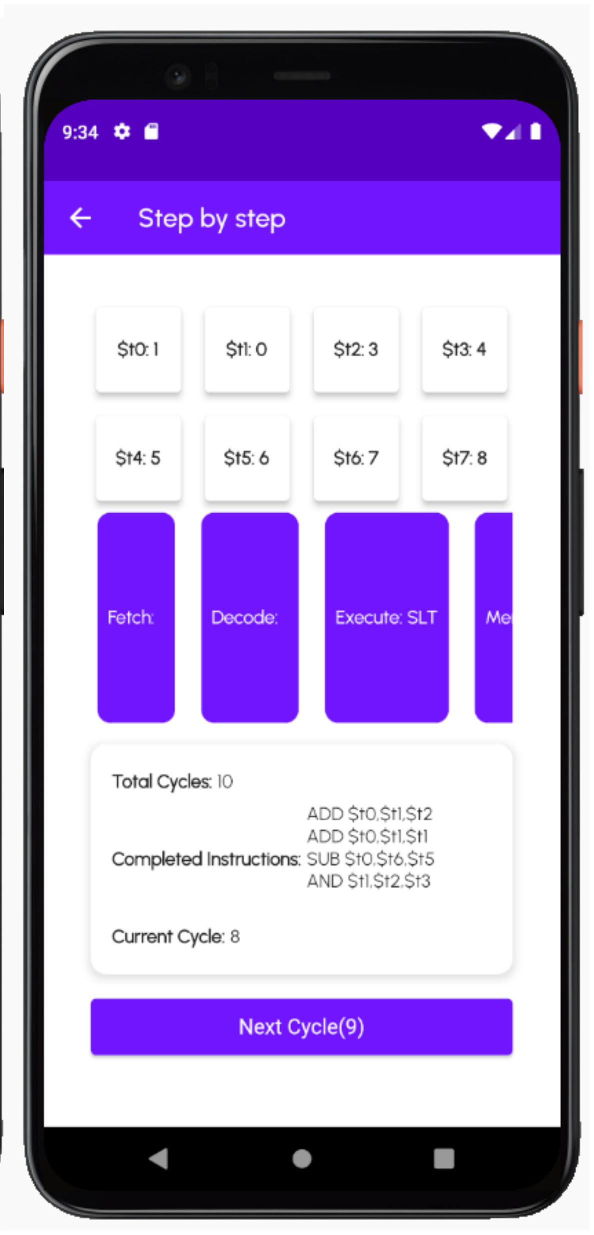
Fetch: | Decode: SLT | Execute: SLL

Total Cycles: 10

Completed Instructions: ADD $t0,$t1,$t2
ADD $t0,$t1,$t1
SUB $t0,$t6,$t5

Current Cycle: 7

Next Cycle(8)



9:34

Step by step

$t0: 1 $t1: 0 $t2: 3 $t3: 4

$t4: 5 $t5: 6 $t6: 7 $t7: 8

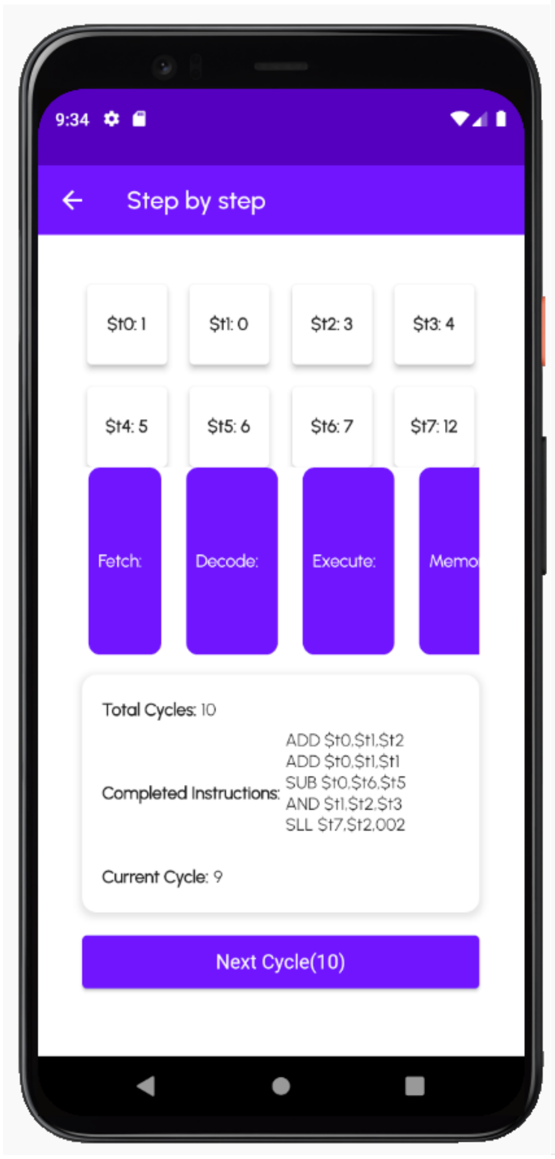
Fetch: | Decode: | Execute: SLT | Me

Total Cycles: 10

Completed Instructions: ADD $t0,$t1,$t2
ADD $t0,$t1,$t1
SUB $t0,$t6,$t5
AND $t1,$t2,$t3

Current Cycle: 8

Next Cycle(9)


9:34

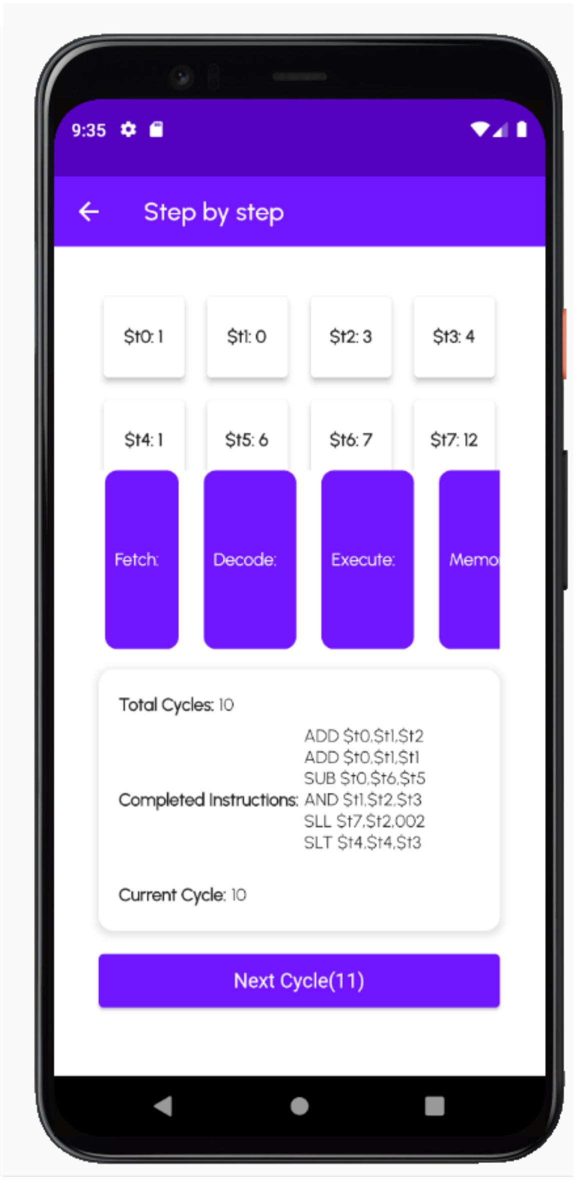
Step by step

$t0: 1	$t1: 0	$t2: 3	$t3: 4
$t4: 5	$t5: 6	$t6: 7	$t7: 12

Total Cycles: 10

Completed Instructions: ADD $t0,$t1,$t2
ADD $t0,$t1,$t1
SUB $t0,$t6,$t5
AND $t1,$t2,$t3
SLL $t7,$t2,002

Current Cycle: 9


9:35

Step by step

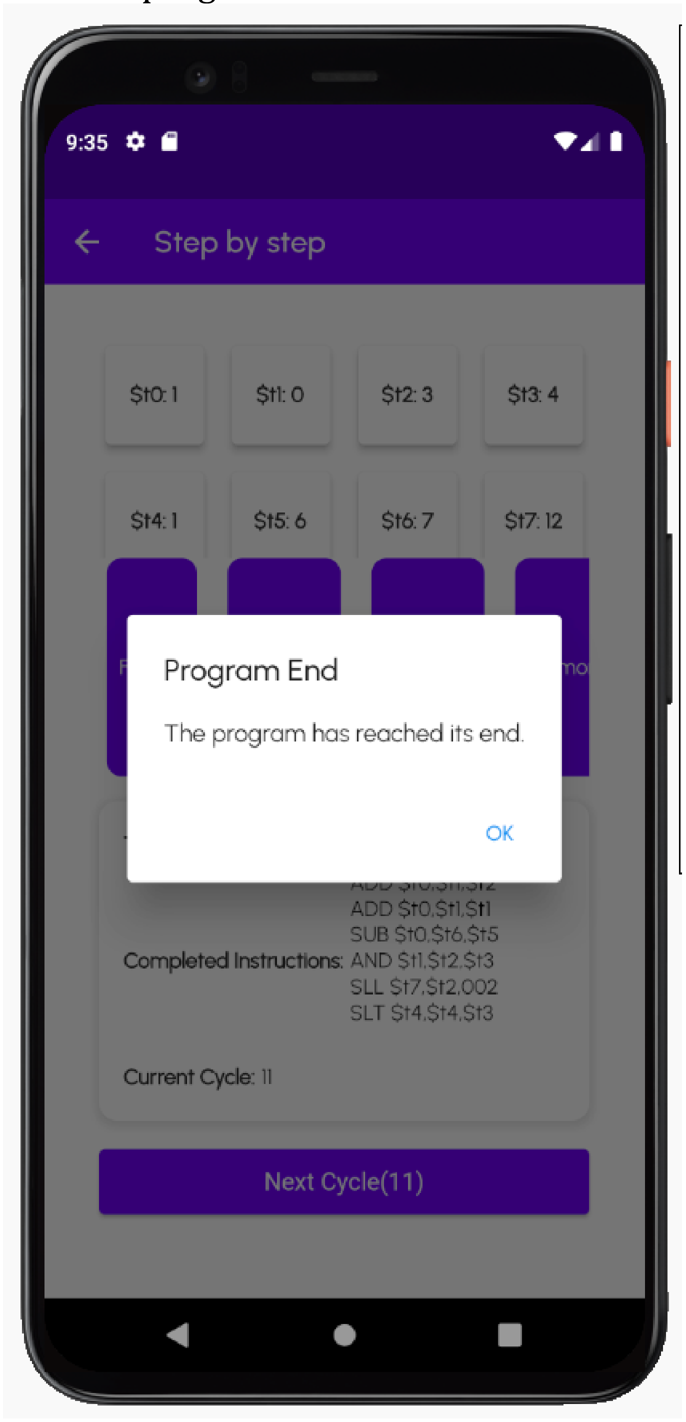
$t0: 1	$t1: 0	$t2: 3	$t3: 4
$t4: 1	$t5: 6	$t6: 7	$t7: 12

Total Cycles: 10

Completed Instructions: ADD $t0,$t1,$t2
ADD $t0,$t1,$t1
SUB $t0,$t6,$t5
AND $t1,$t2,$t3
SLL $t7,$t2,002
SLT $t4,$t4,$t3

Current Cycle: 10

After finishing the run process the application will appear a message telling the user the program ended.



$t0: 1 $t1: 0 $t2: 3 $t3: 4

$t4: 1 $t5: 6 $t6: 7 $t7: 12

Program End

The program has reached its end.

Completed Instructions:
```
ADD $t0,$t1,$t1
SUB $t0,$t6,$t5
AND $t1,$t2,$t3
SLL $t7,$t2,002
SLT $t4,$t4,$t3
```

Current Cycle: 11