INCREASING SYSTEM SAFETY OF MICROCONTROLLER AND PERIPHERALS WITH SOFTWARE LIBRARY

Software Qualification Test Document

Dissertation by

SUBHRA SAR

2023HT65019

Dissertation work carried out at

Bosch Global Software and Technologies, Bangalore

Master of Technology in Automotive Electronics Degree Program

Under the Supervision of

M SATISH KUMAR

Bosch Global Software and Technologies, Bangalore

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE

PILANI (RAJASTHAN)

March, 2025

Table of Contents

Introduction

This document serves as a reference for conducting the qualification test of the Proof-of-Concept (PoC) code developed to validate various instruction sets of the Cortex-M processor series. It provides a concise overview of the instruction sets evaluated through the implementation of supplementary logic within the code. Additionally, it outlines the expected test outcomes for both positive and negative scenarios, with a focus on the verification of register values.

The results obtained from simulations performed using Keil µVision 5 are included as supporting evidence, demonstrating the validity and effectiveness of the implemented code under positive circumstances.

The structure of this document is organized according to the names of the assembly files, each corresponding to a specific group of instruction sets.

Note

The Software test code has been updated in the Git Hub Link:
SARSUBHRA/Cortex_M4: Verification of assembly code for Cortex_M4

Add.s

Description

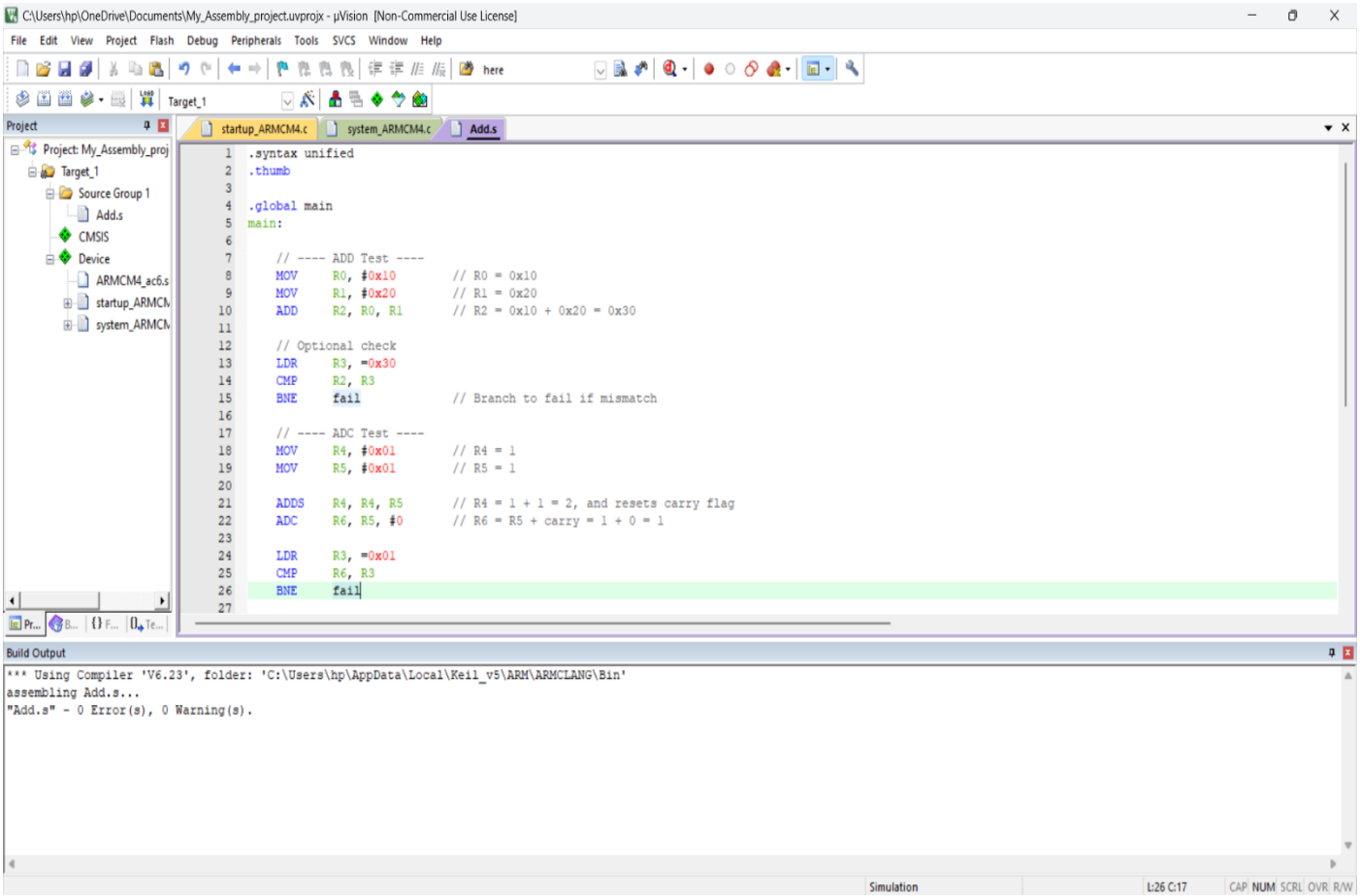
ADD: Adds two registers or a register and immediate.
ADC: Adds with carry — useful for multi-word addition.
ADR: Loads the current or nearby program counter-relative address into a register.

Expected Test Result

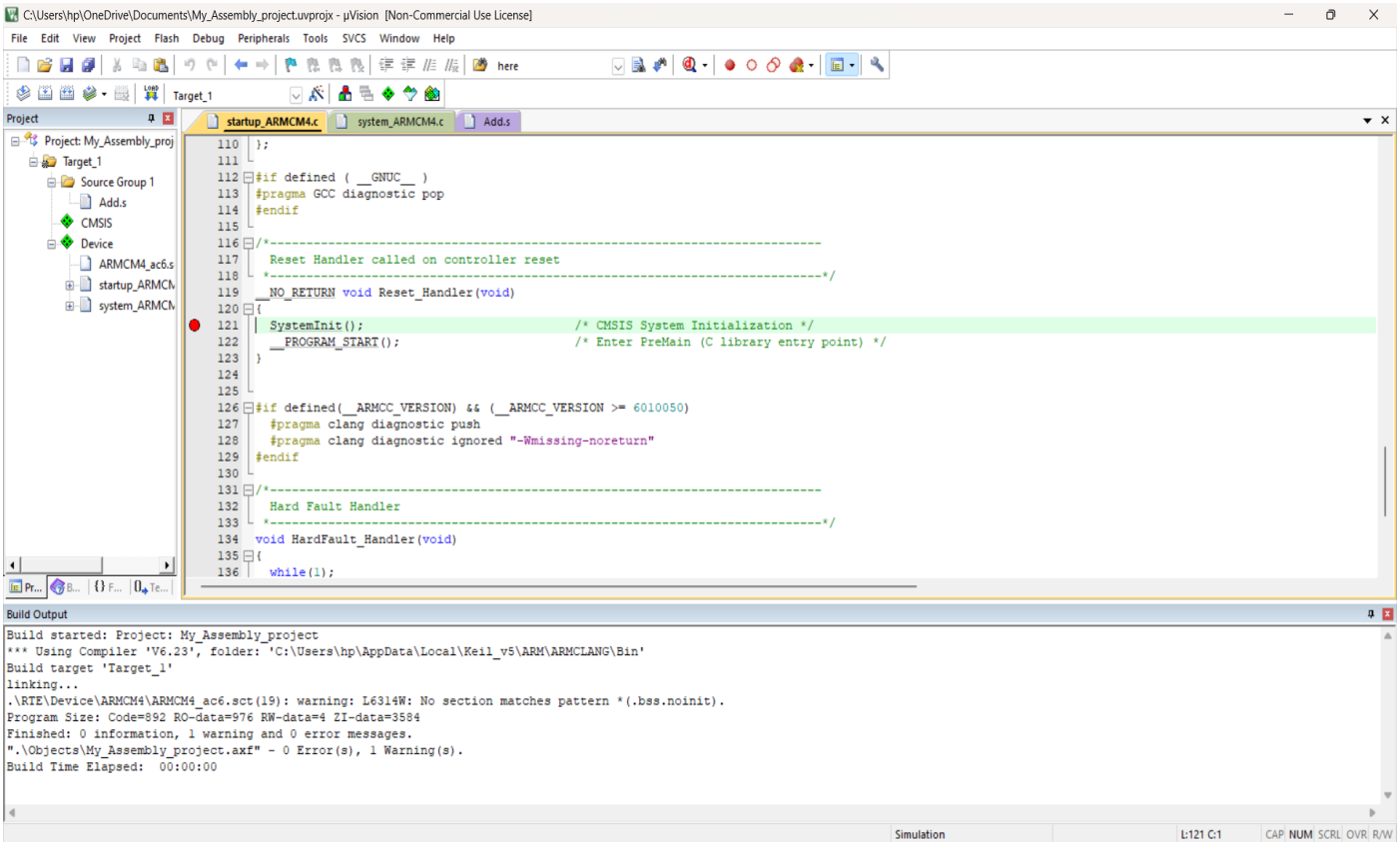
Check the value of R3:
0x01: success
0xFF: fail

Obtained Test Result

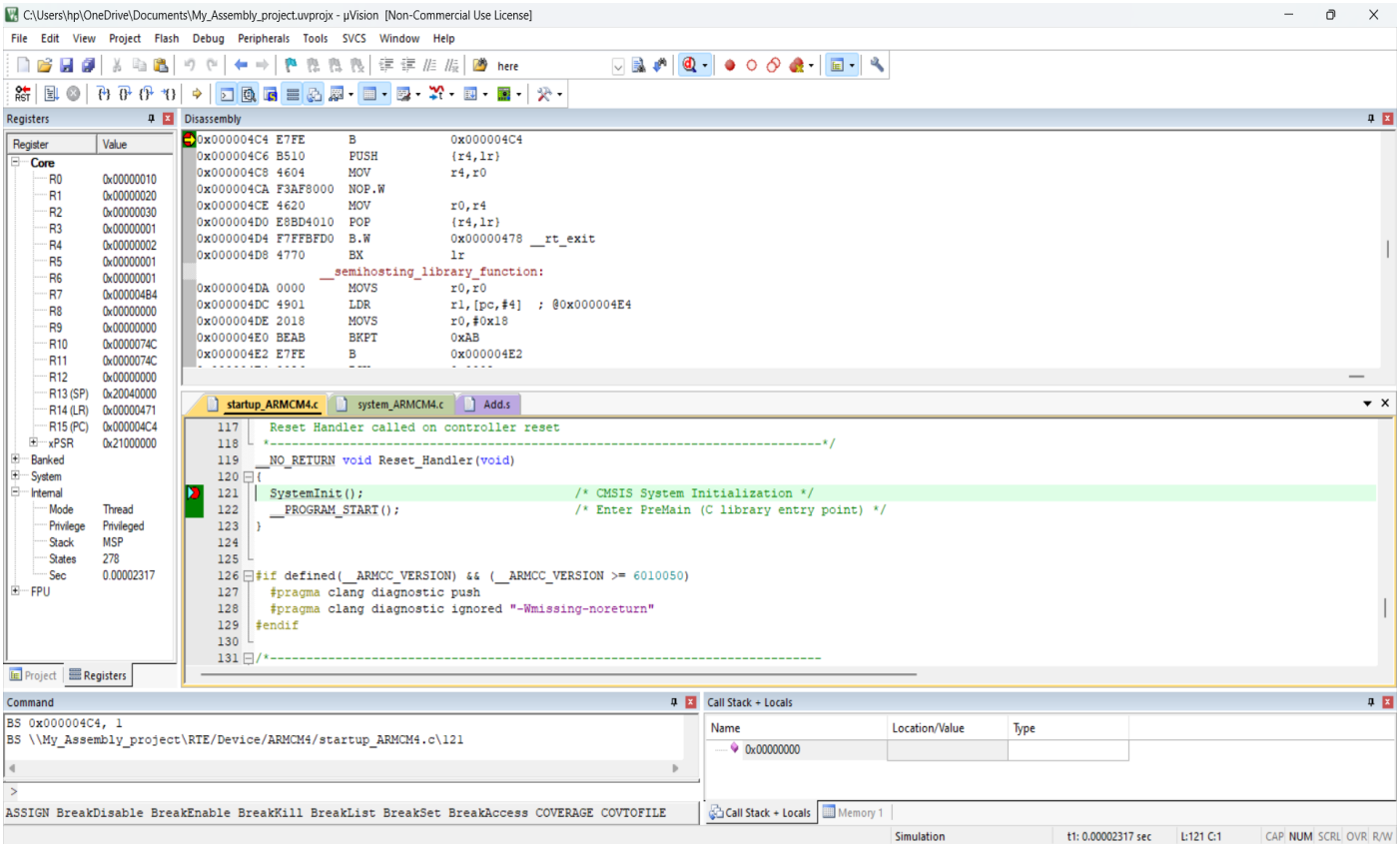
Compilation:



Build:



Execution:



Substract_Multiply.s

Description

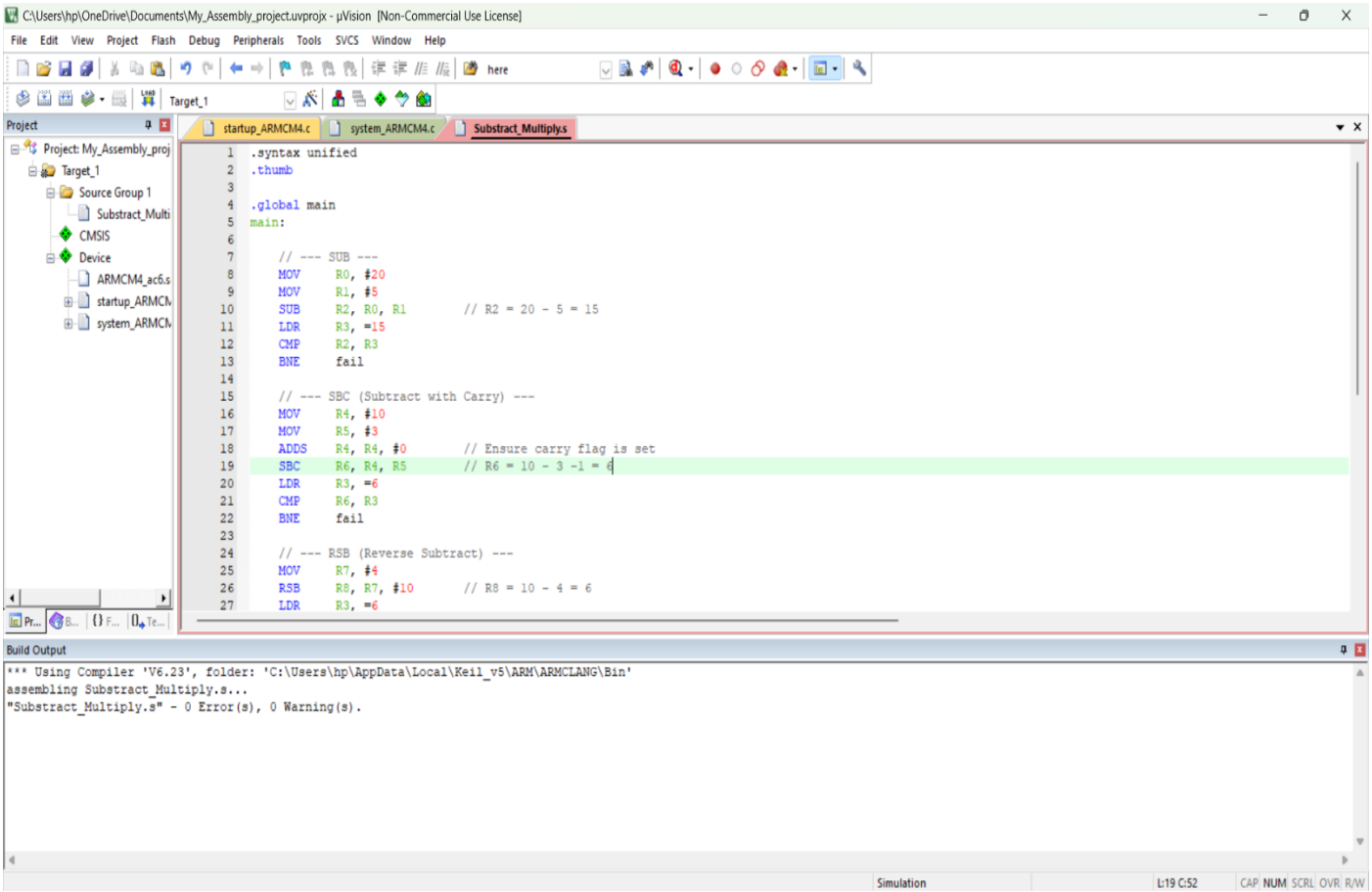
SUB: Subtract
SBC: Subtract with carry
RSB: Reverse subtracts
MUL: Multiply

Expected Test Result

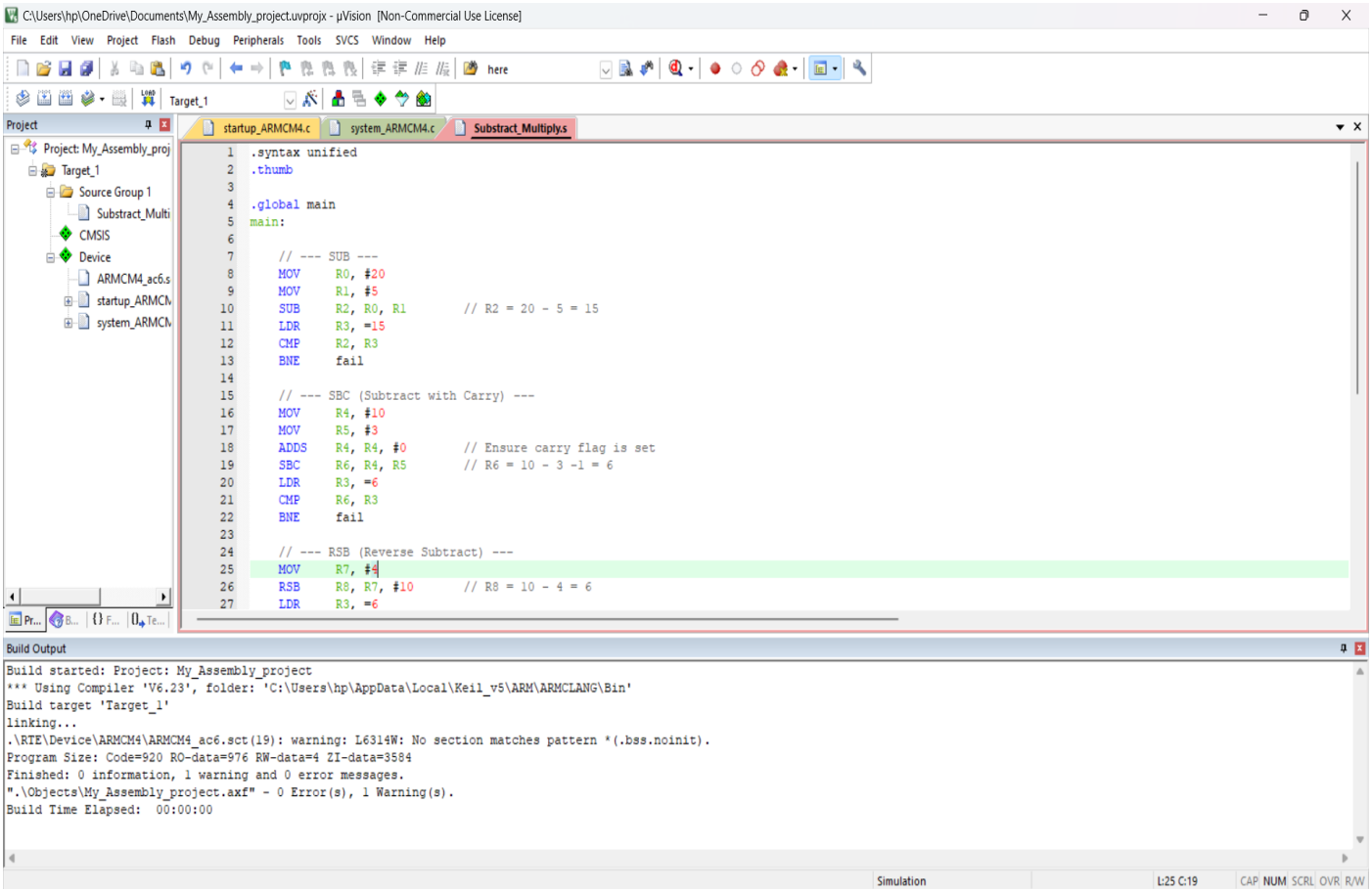
Check the value of R0:
0x01: success
0xFF: fail

Obtained Test Result

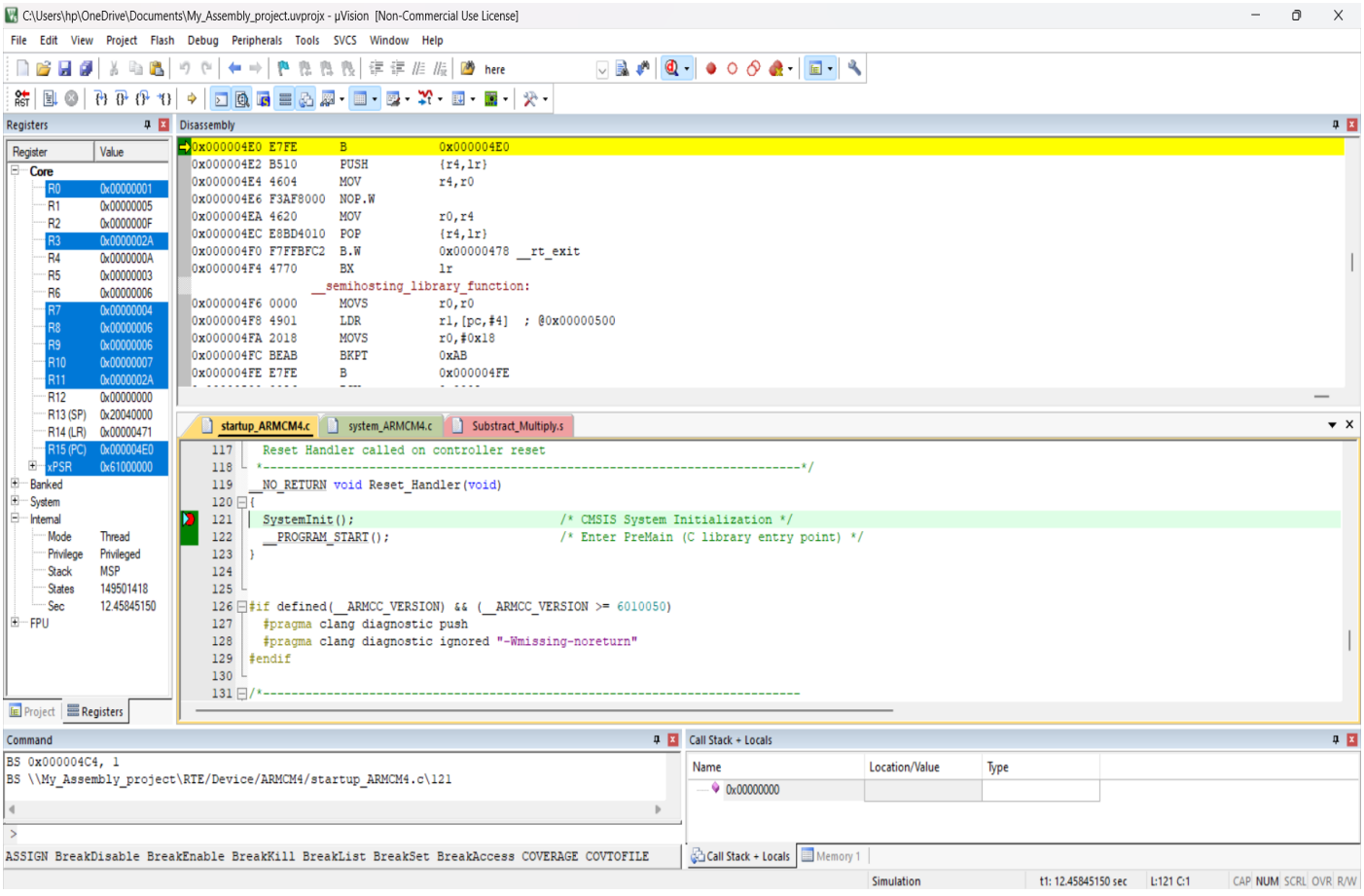
Compilation:



Build:



Execution:



Move.s

Description

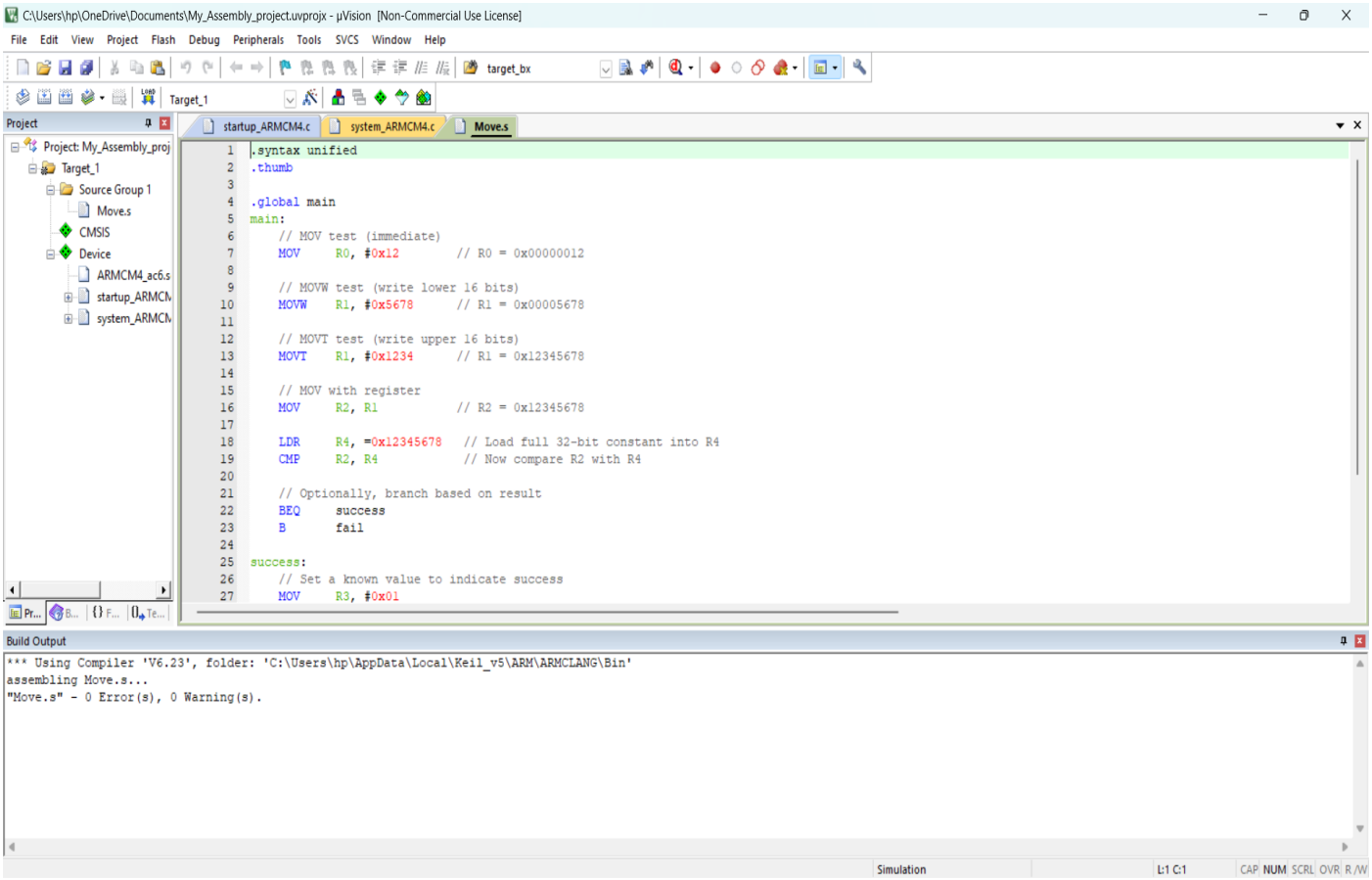
MOV: Move immediate value to register
MOVW: Move 16-bit constant (lower half)
MOVT: Move 16-bit constant (upper half)

Expected Test Result

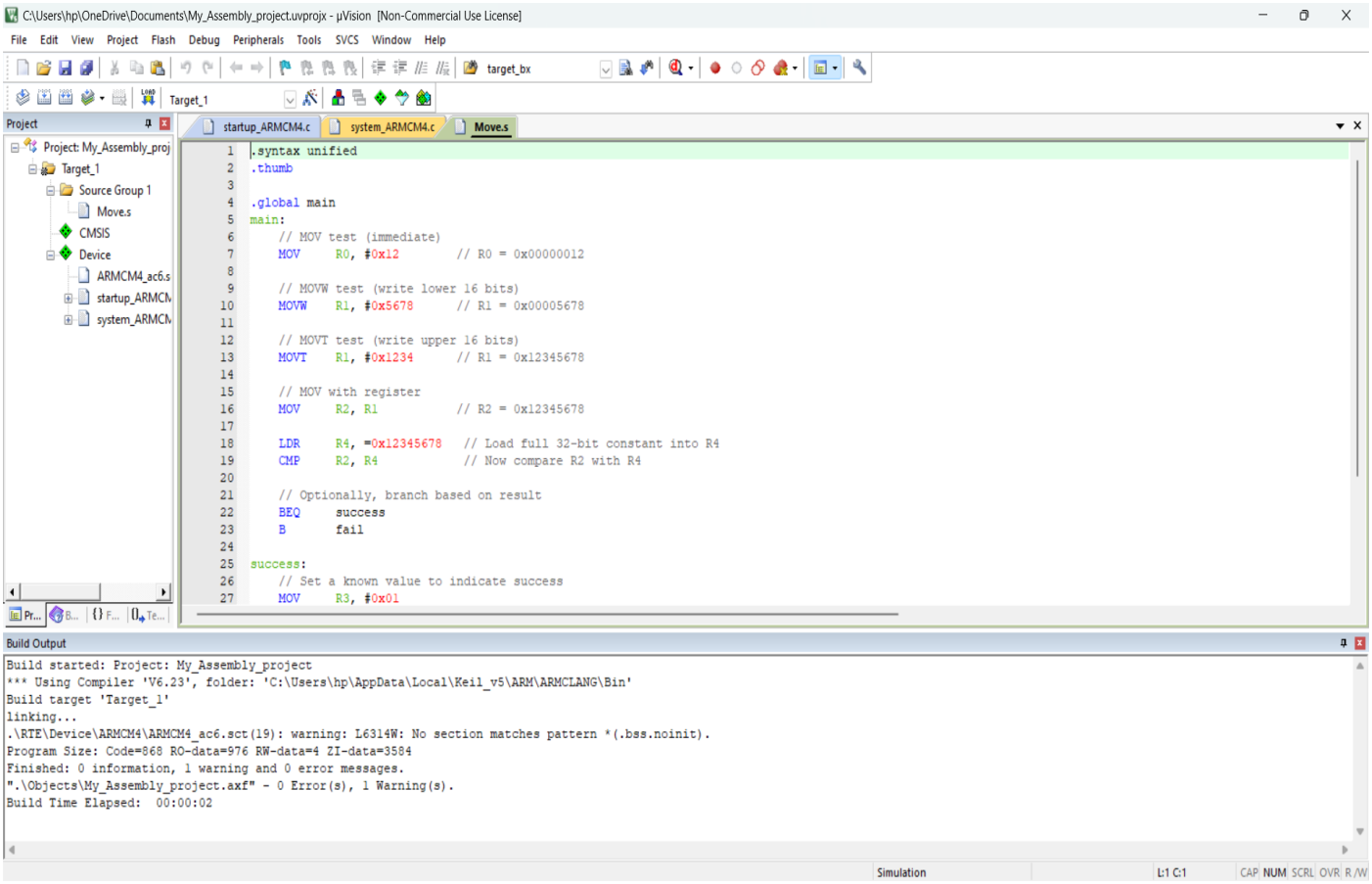
Check the value of R3:
0x01: success
0xFF: fail

Obtained Test Result

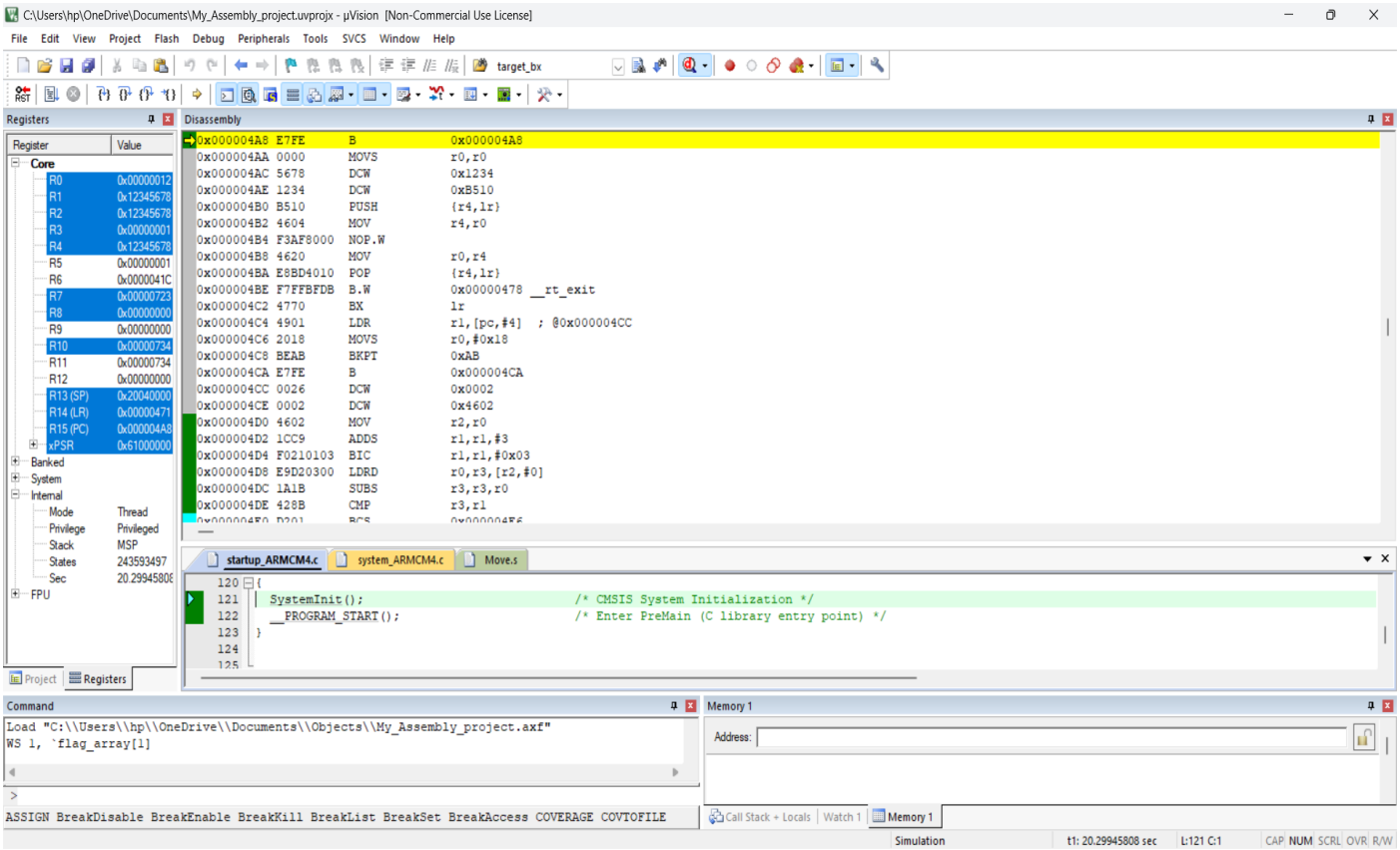
Compilation:



Build:



Execution:



Store.s

Description

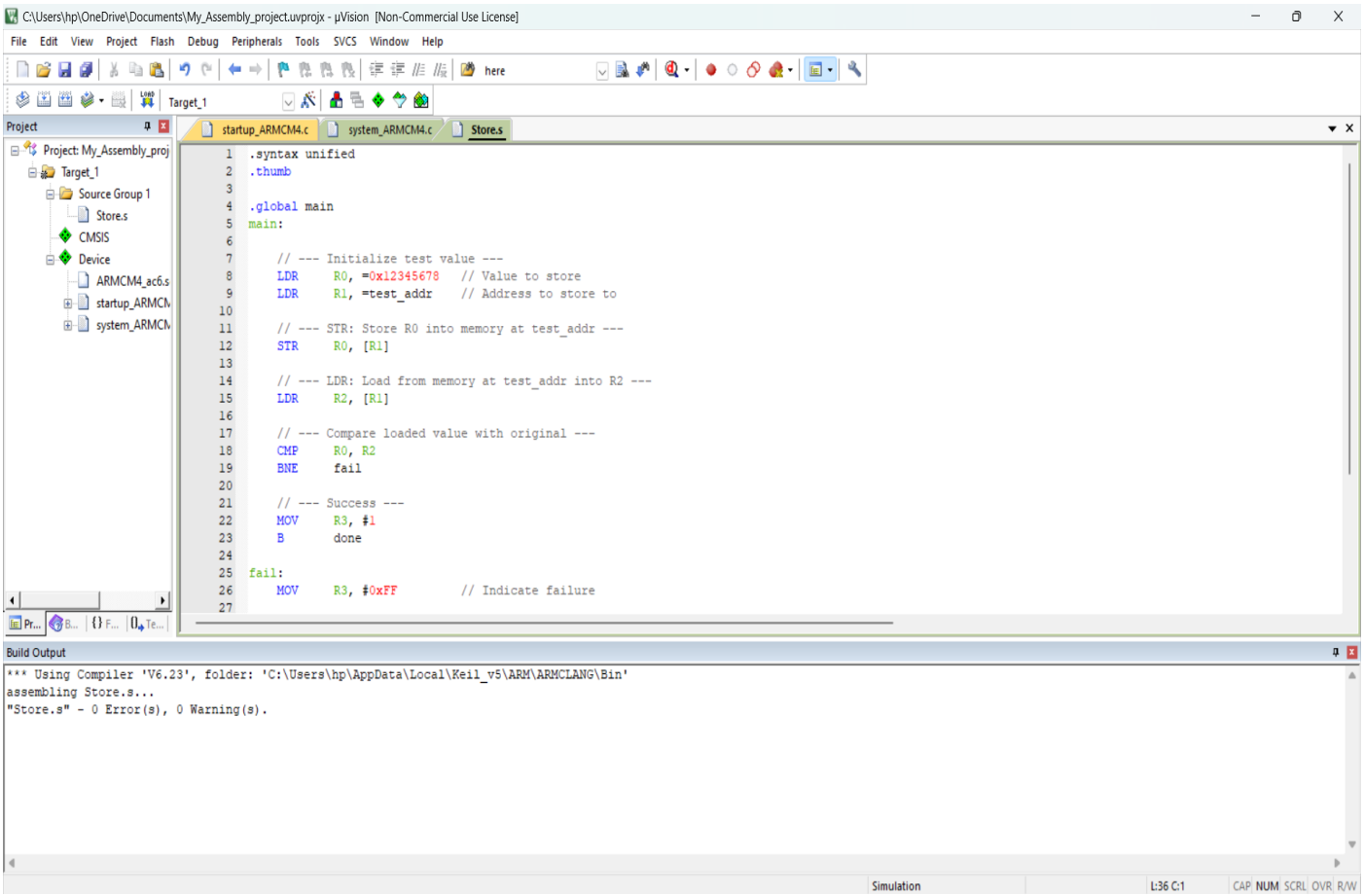
STR: Store Register to Memory
LDR: Load Register from Memory

Expected Test Result

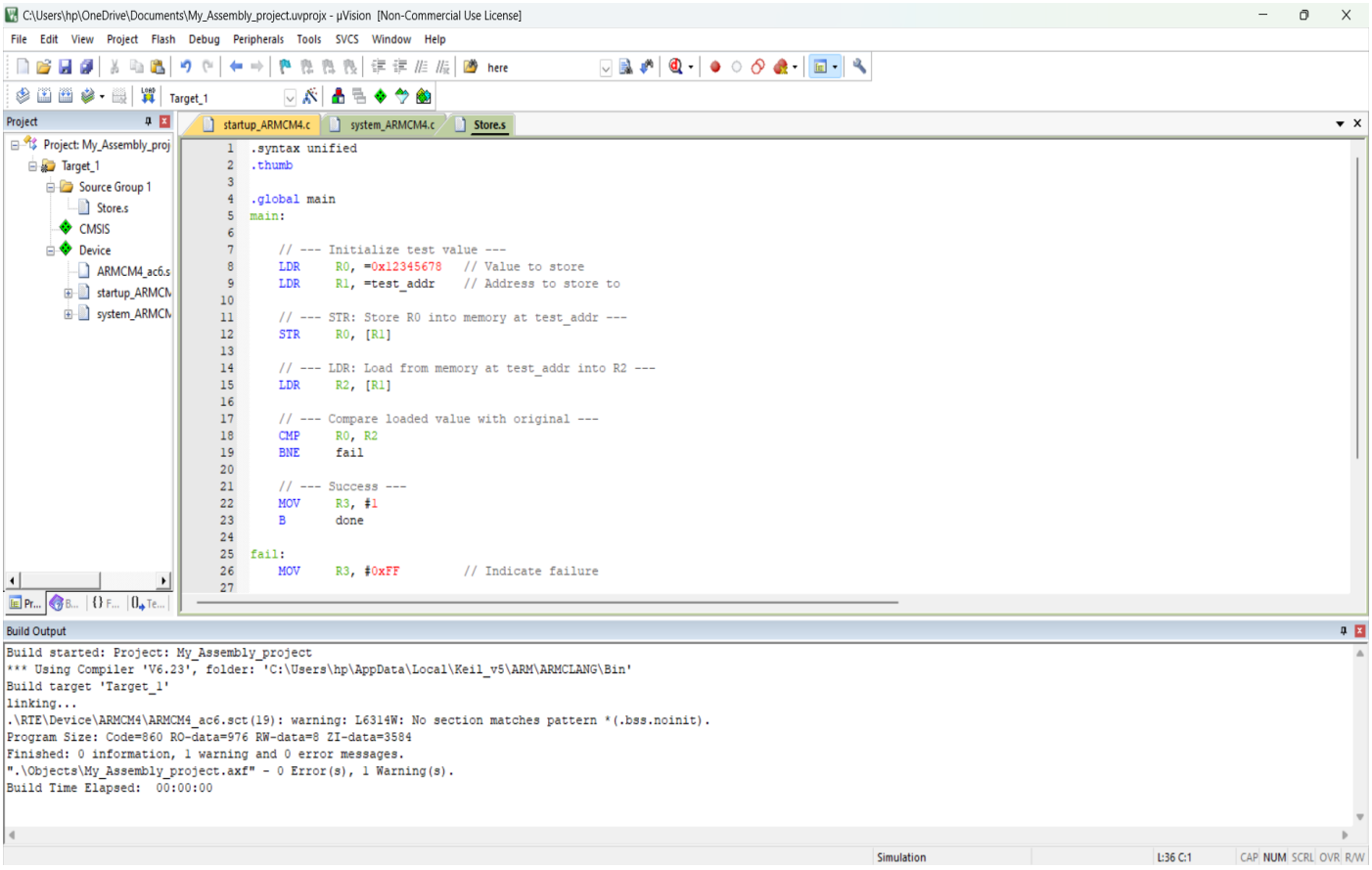
Check the value of R3:
0x01: success
0xFF: fail

Obtained Test Result

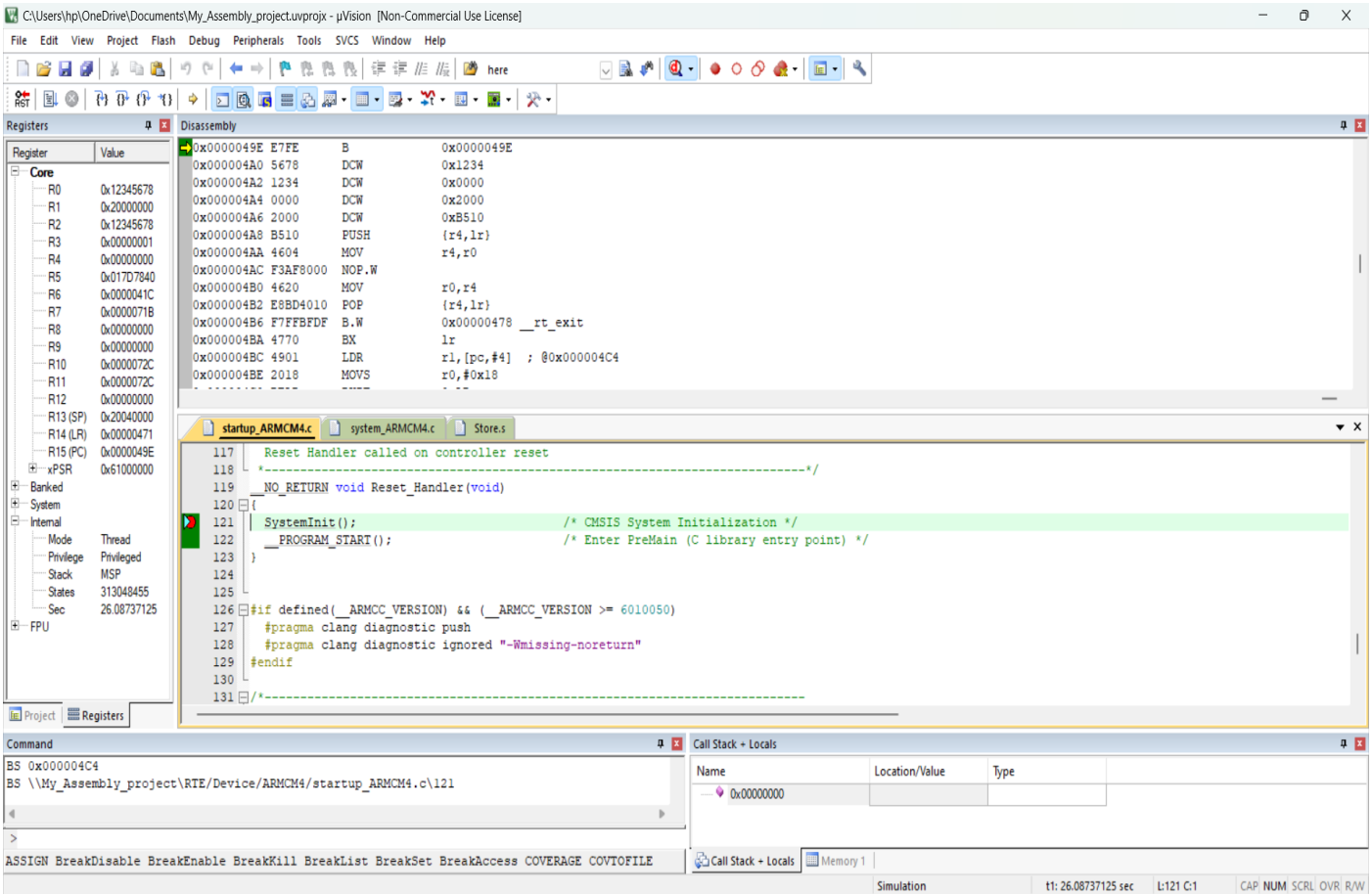
Compilation:



Build:



Execution:



Bitwise.s

Description

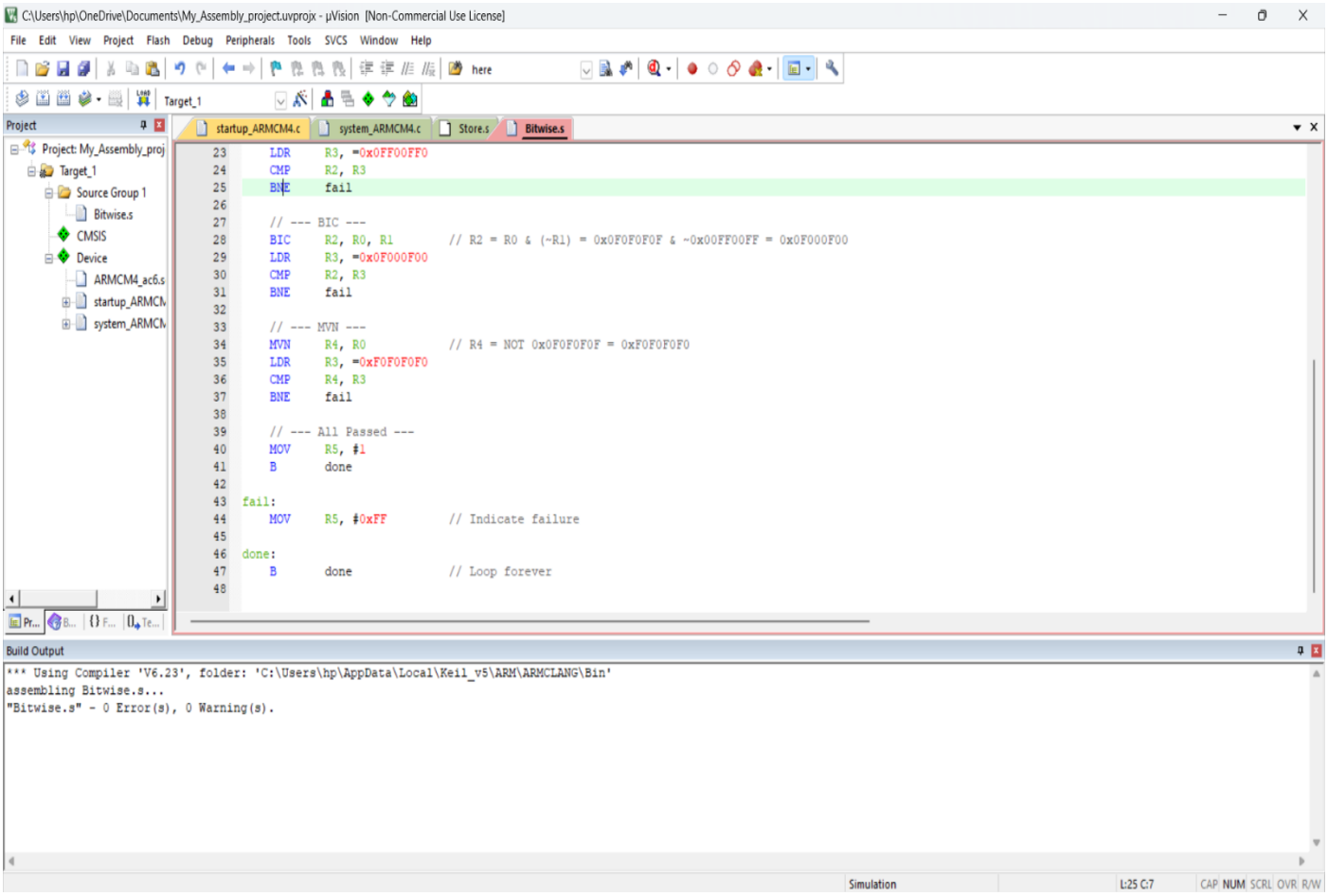
AND: Bitwise AND
ORR: Bitwise OR
EOR: Bitwise Exclusive OR (XOR)
BIC: Bit Clear (AND with NOT)
MVN: Move NOT

Expected Test Result

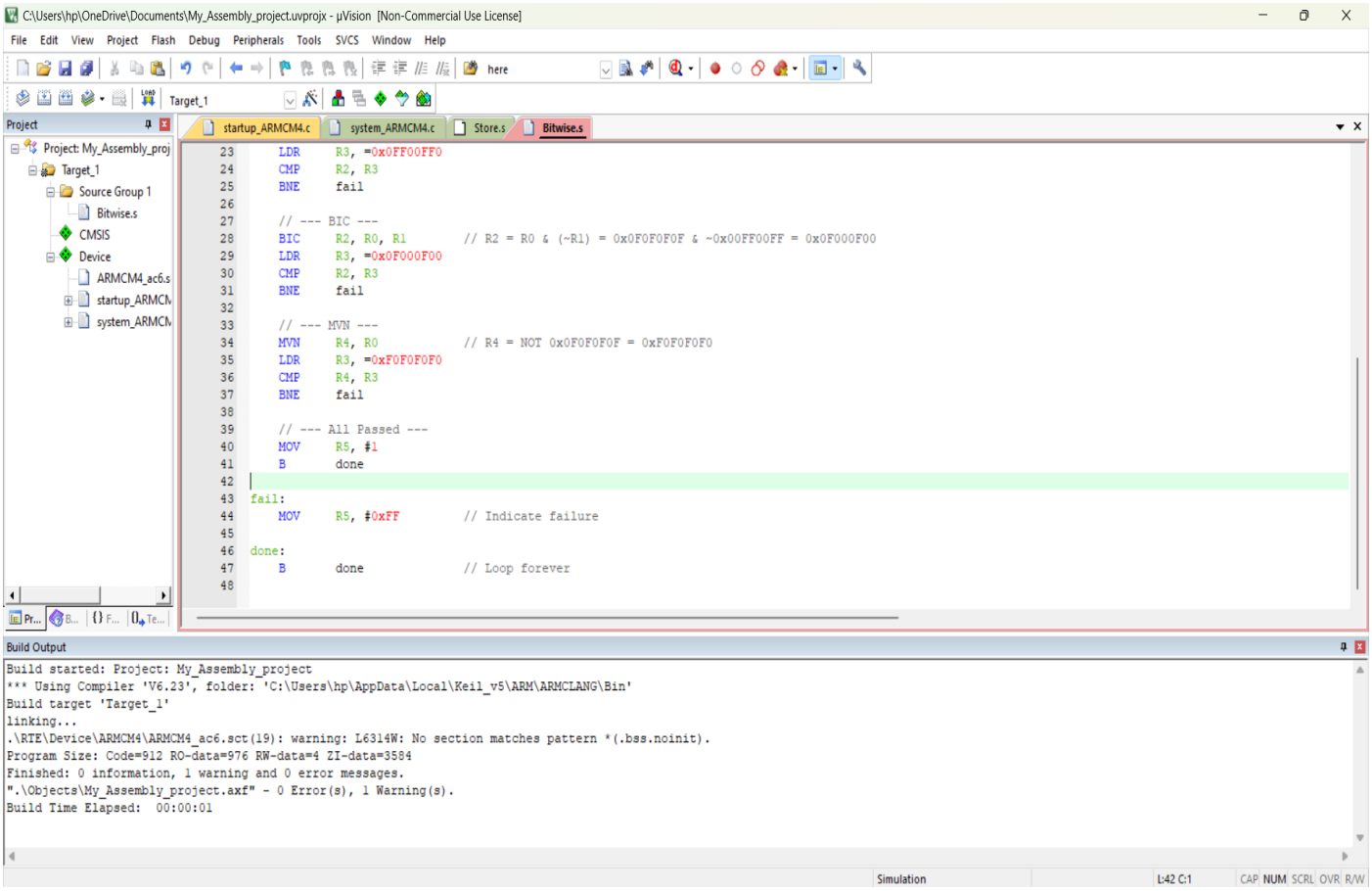
Check the value of R5:
0x01: success
0xFF: fail

Obtained Test Result

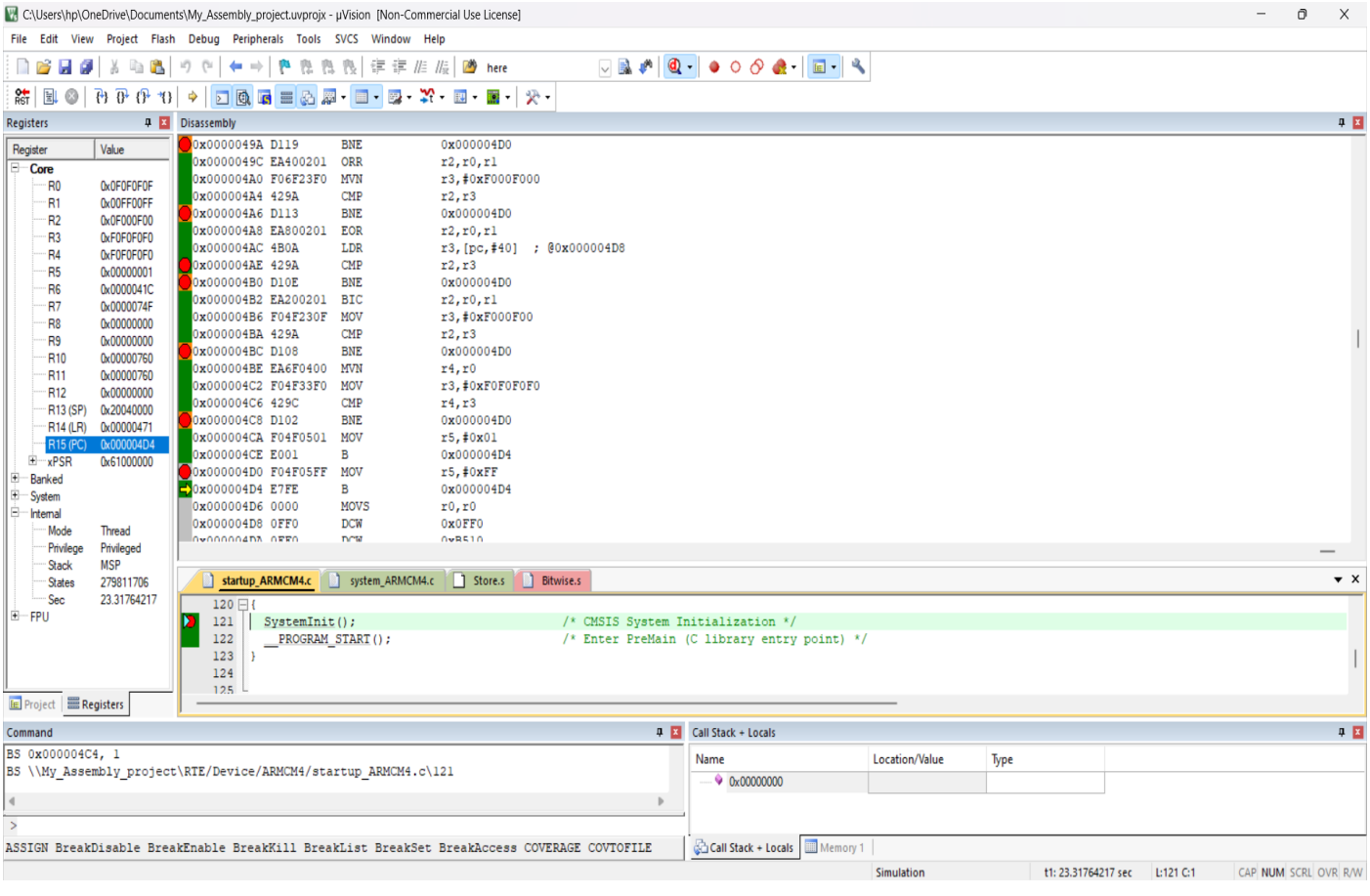
Compilation:



Build:



Execution:



Shift_Rotate.s

Description

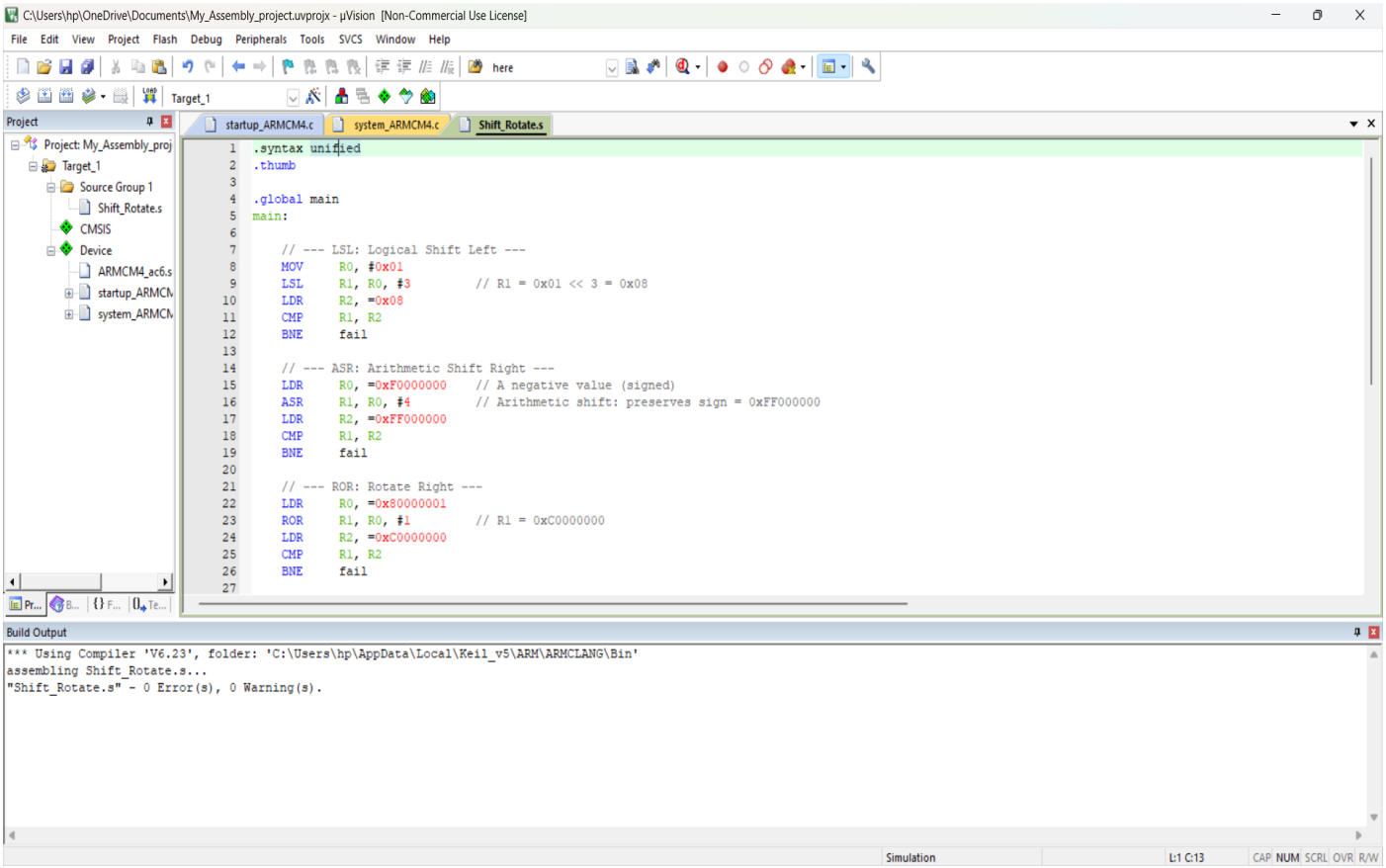
LSL: Logical Shift Left
ASR: Arithmetic Shift Right
ROR: Rotate Right
RRX: Rotate Right with Extend (through carry)

Expected Test Result

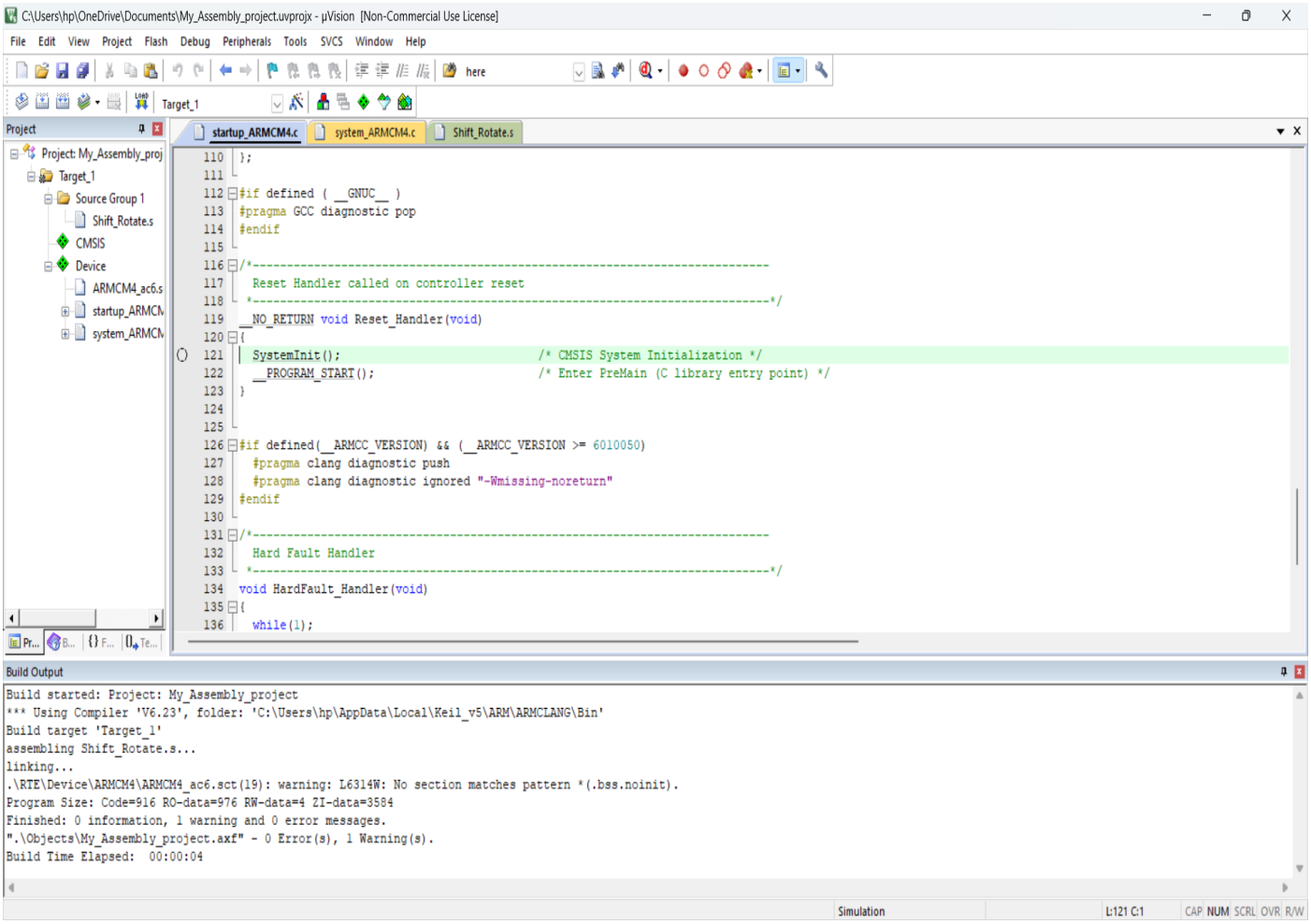
Check the value of R5:
0x01: success
0xFF: fail

Obtained Test Result

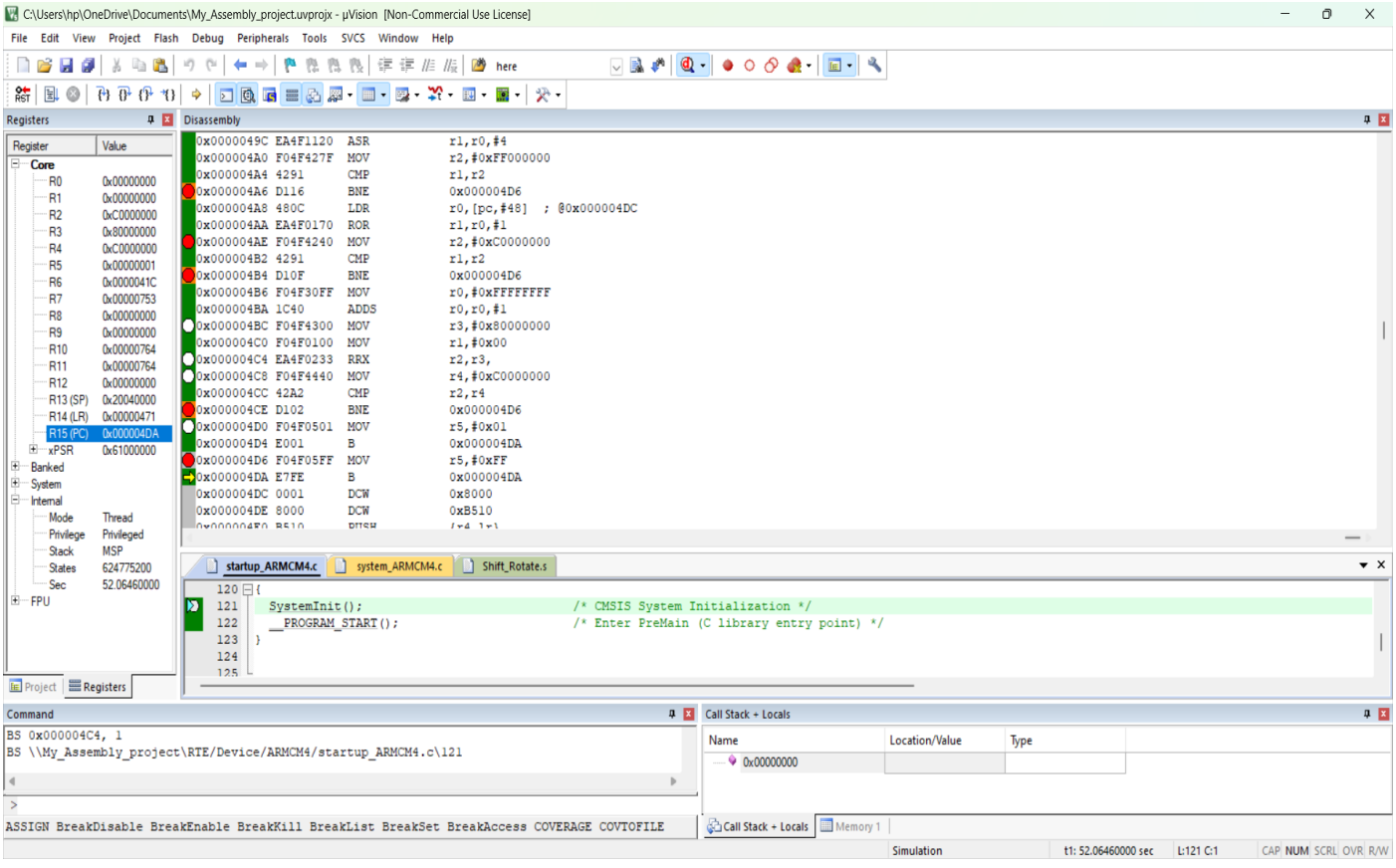
Compilation:



Build:



Execution:



Compare.s

Description

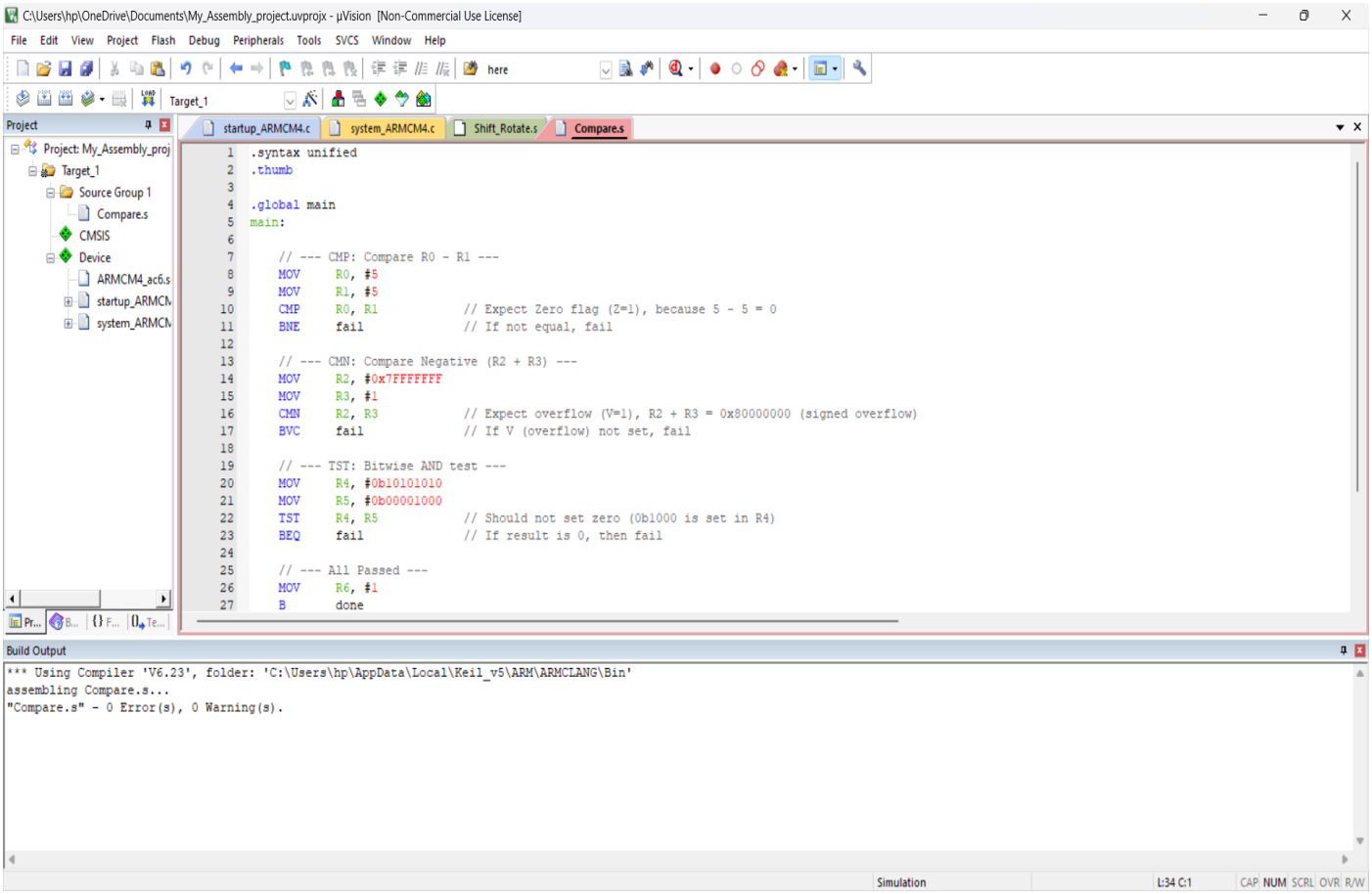
CMP: Compare (subtract, sets flags)
CMN: Compare Negative (add, sets flags)
TST: Bitwise AND (sets flags)

Expected Test Result

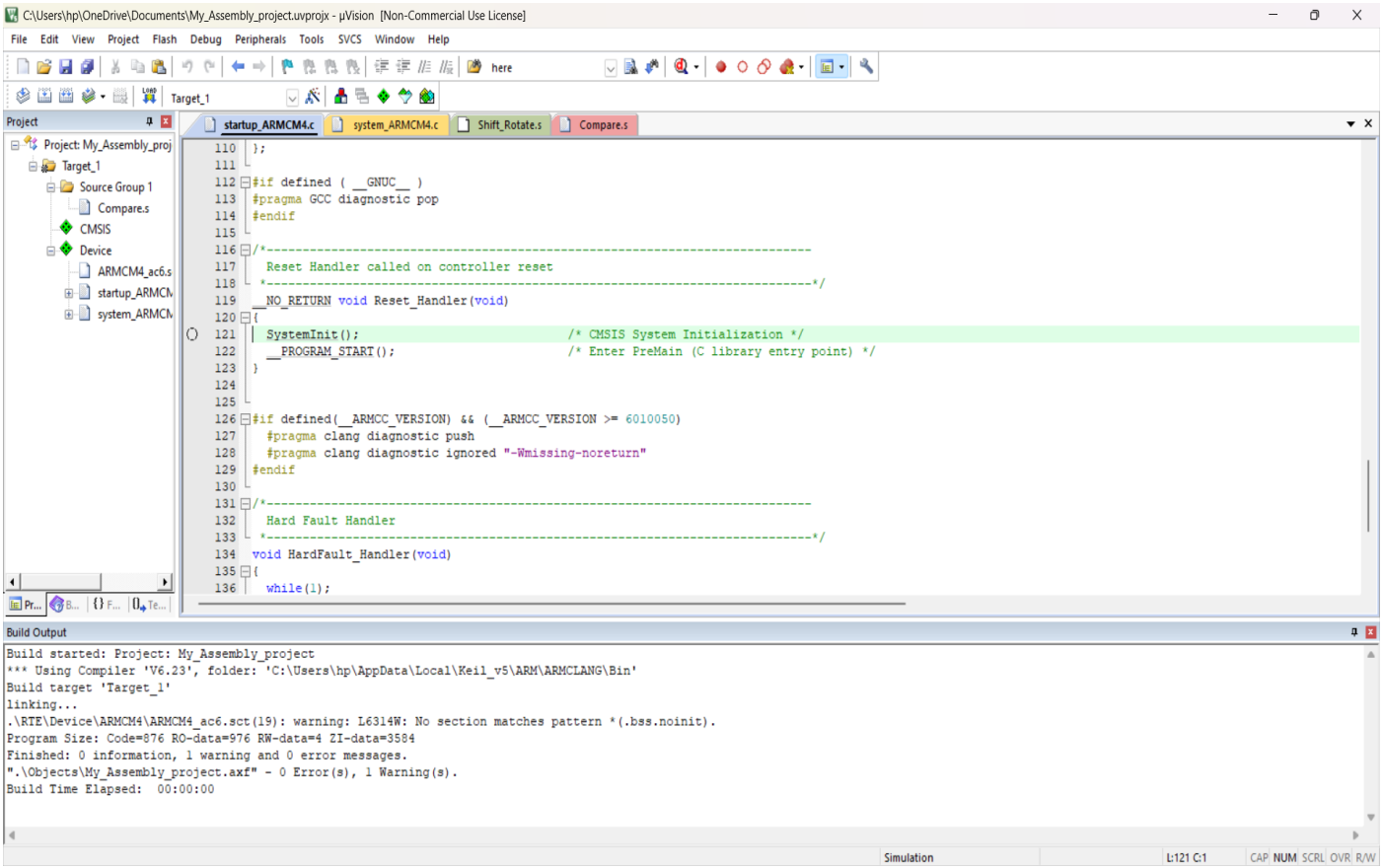
Check the value of R6:
0x01: success
0xFF: fail

Obtained Test Result

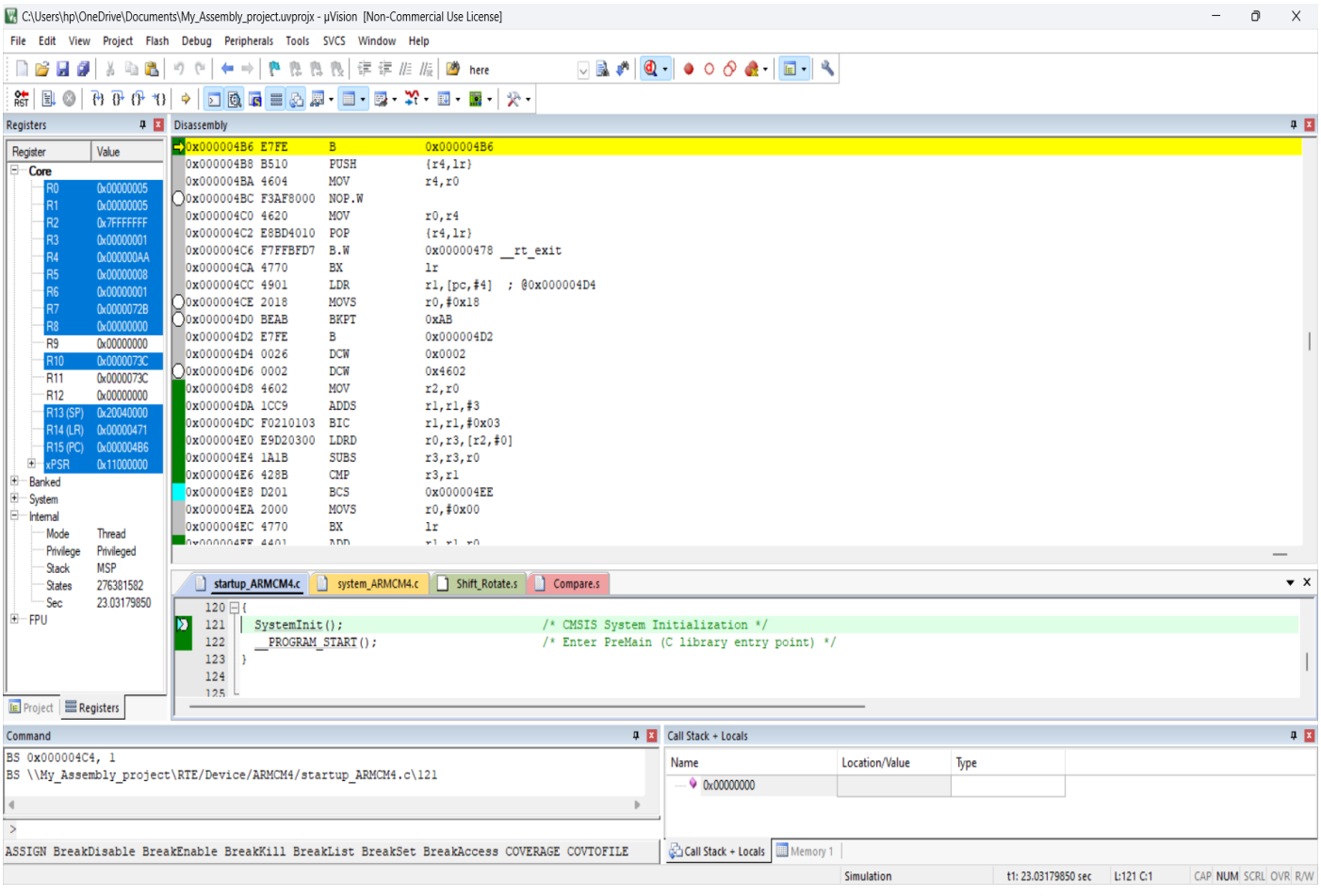
Compilation:



Build:



Execution:



Stack.s

Description

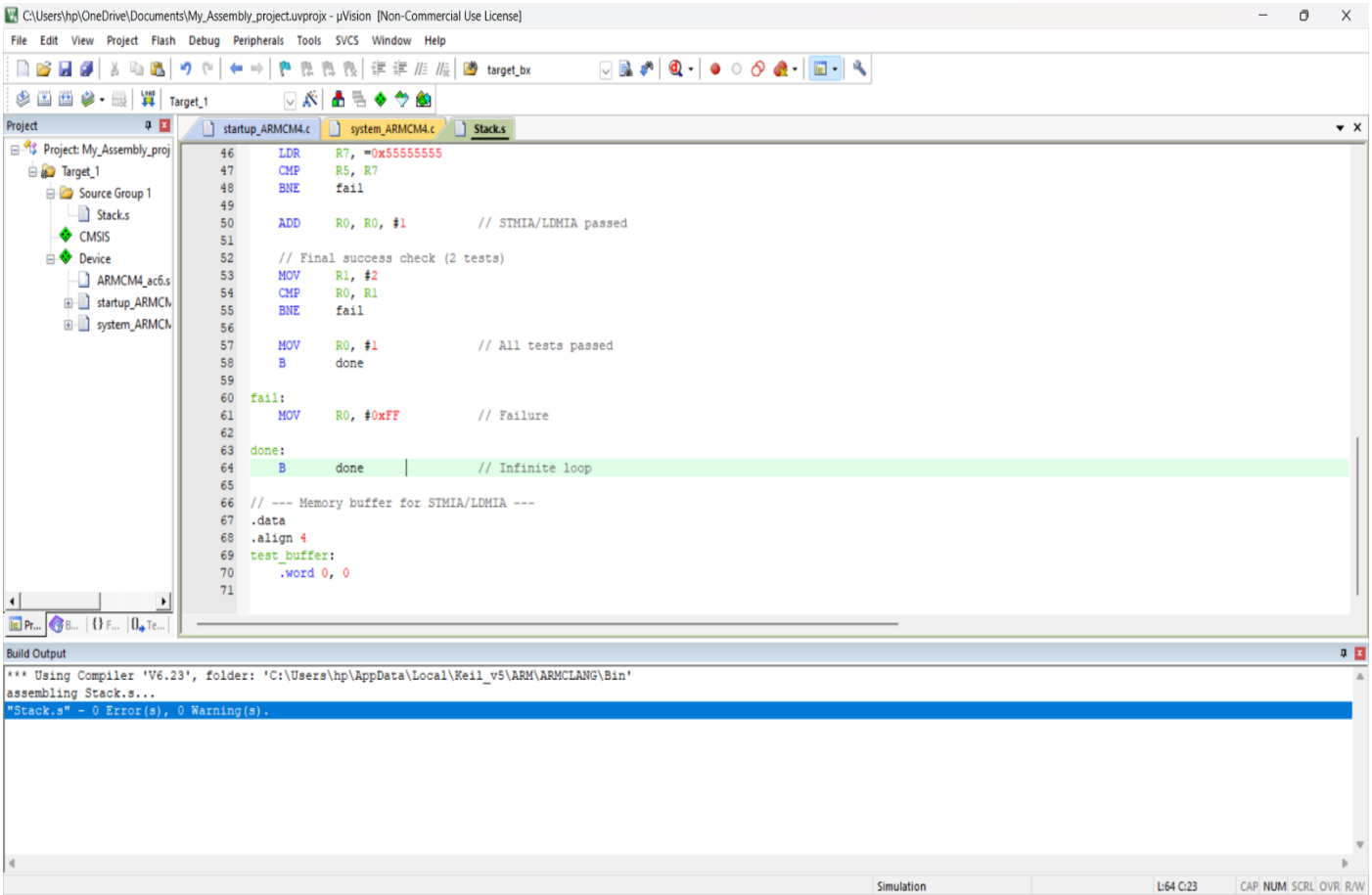
PUSH: Store multiple registers to the stack
POP: Restore multiple registers from the stack
STMIA: Store Multiple Increment After (to memory)
LDMIA: Load Multiple Increment After (from memory)

Expected Test Result

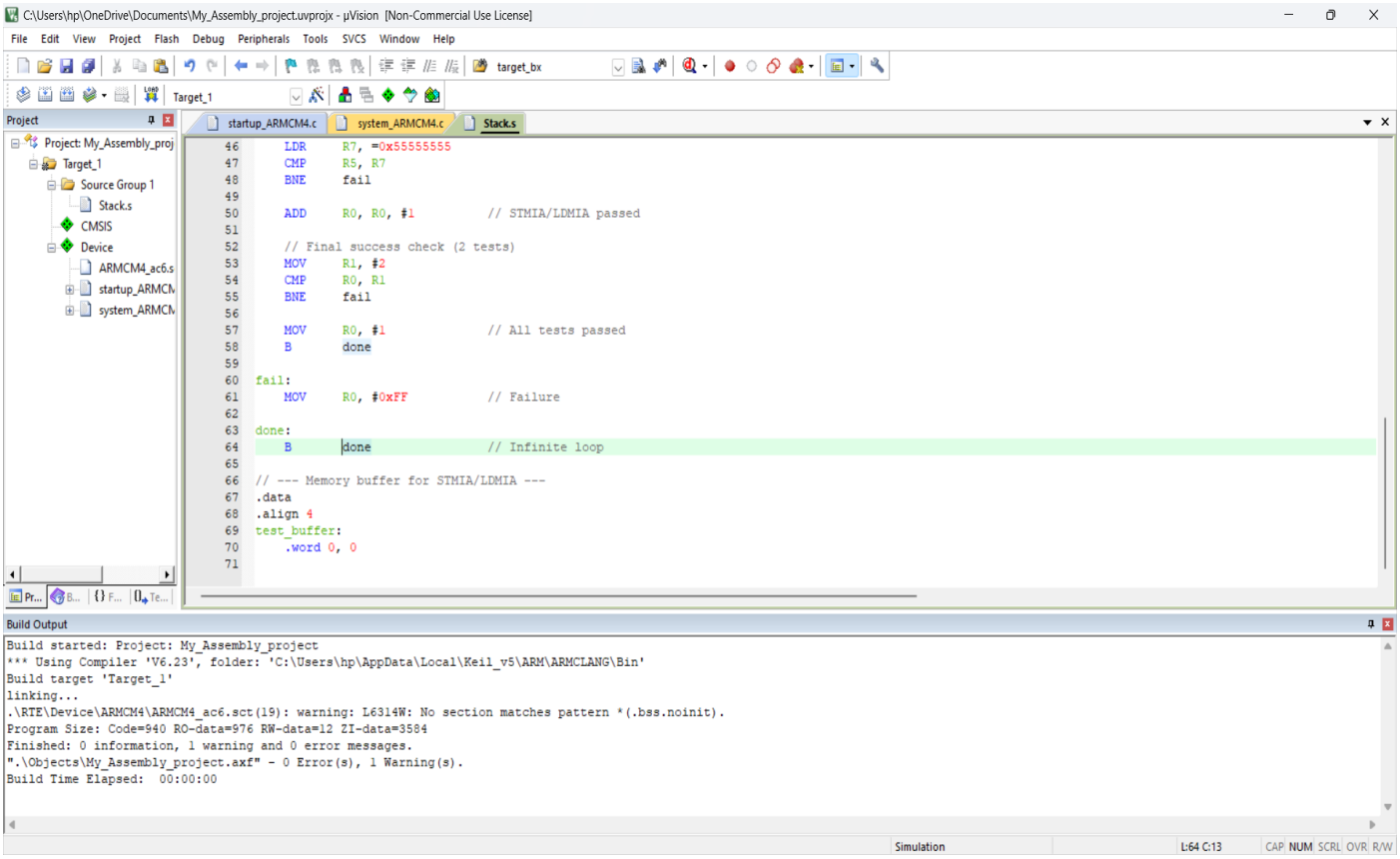
Check the value of R0:
0x01: success
0xFF: fail

Obtained Test Result

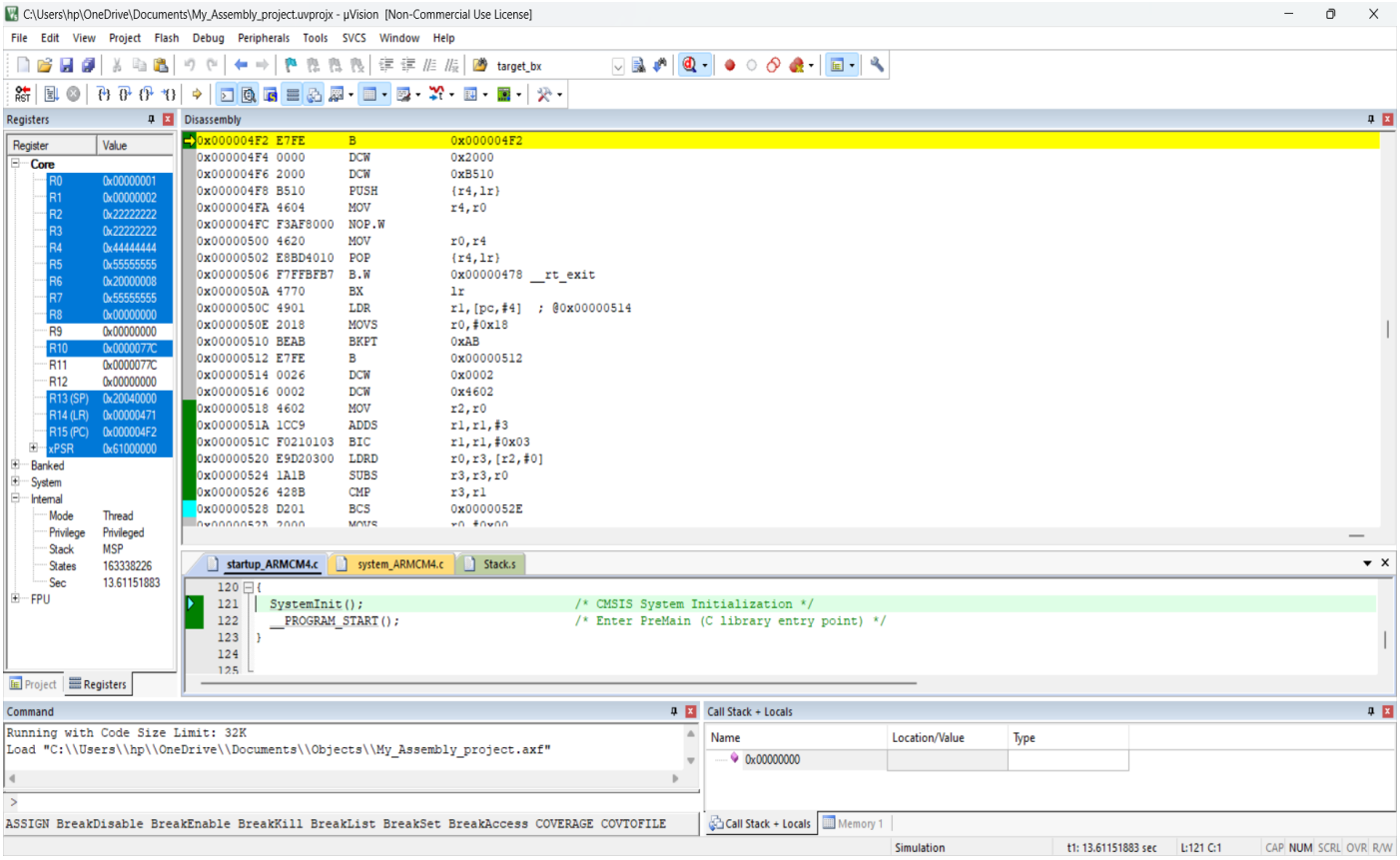
Compilation:



Build:



Execution:



Floating.s

Description

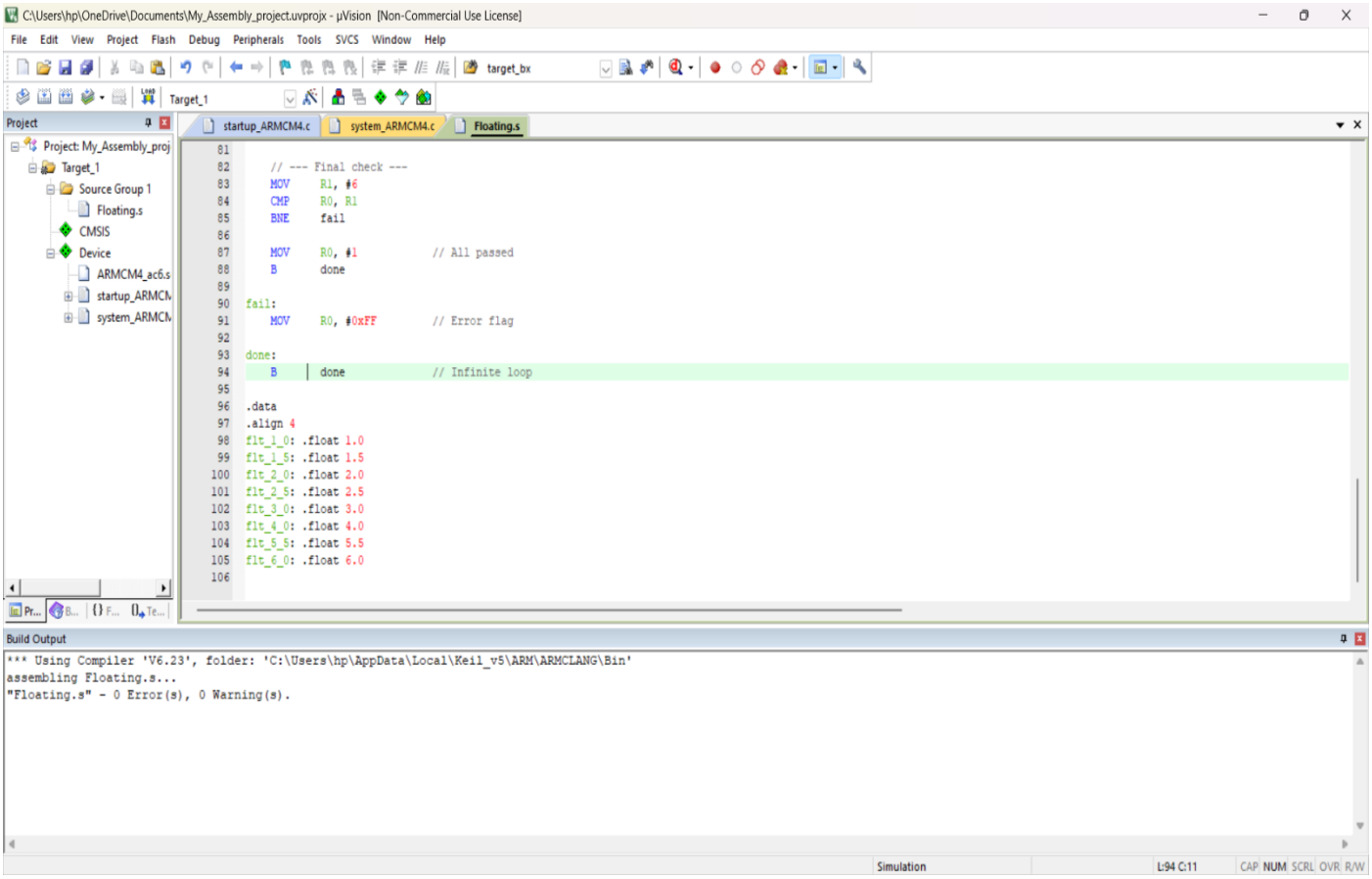
VMOV: Move values between registers or between core and FPU
VADD.F32: Floating-point addition
VSUB.F32: Floating-point subtraction
VMUL.F32: Floating-point multiplication
VDIV.F32: Floating-point division
VCMP.F32: Floating-point compare

Expected Test Result

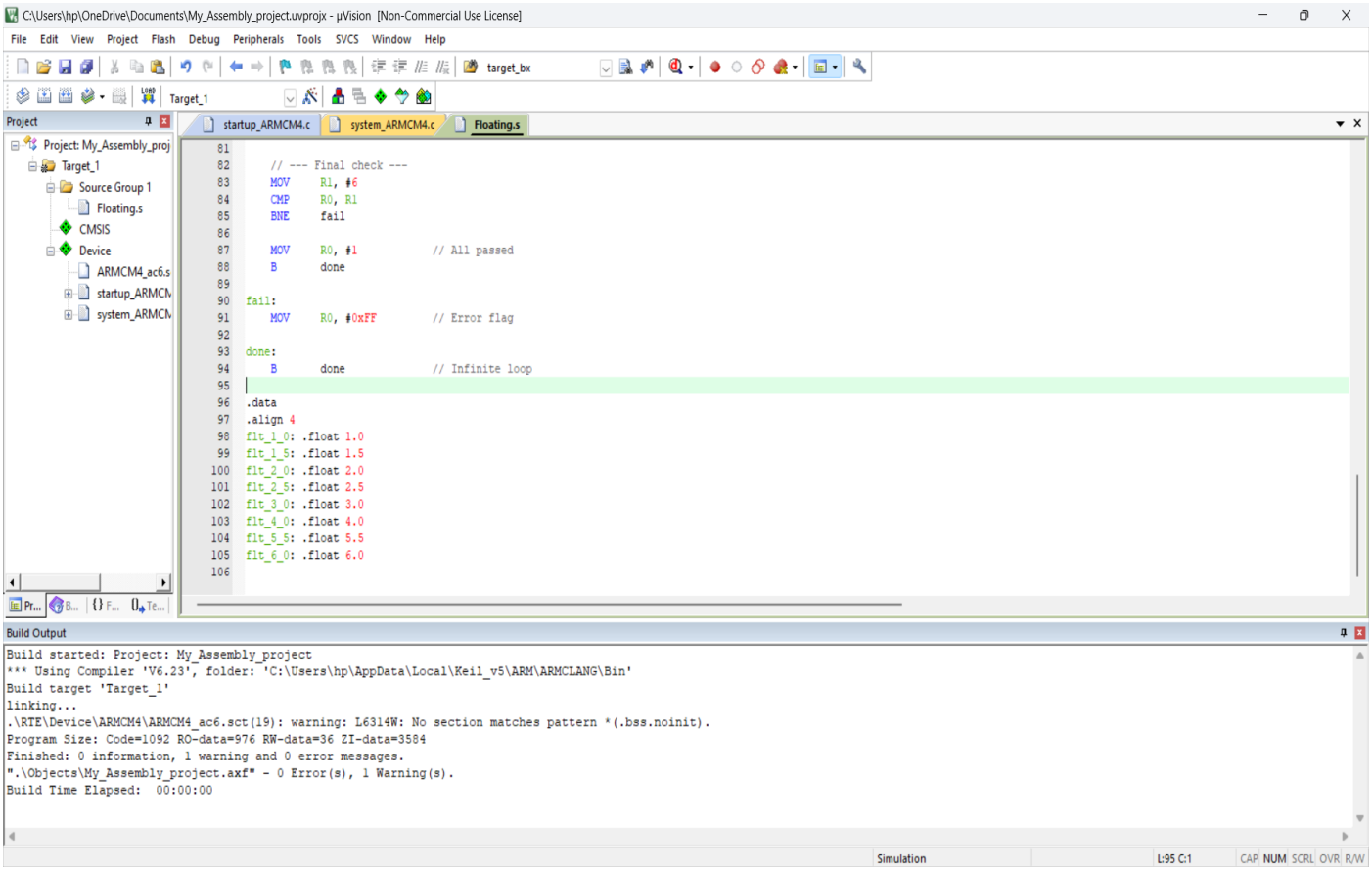
Check the value of R0:
0x01: success
0xFF: fail

Obtained Test Result

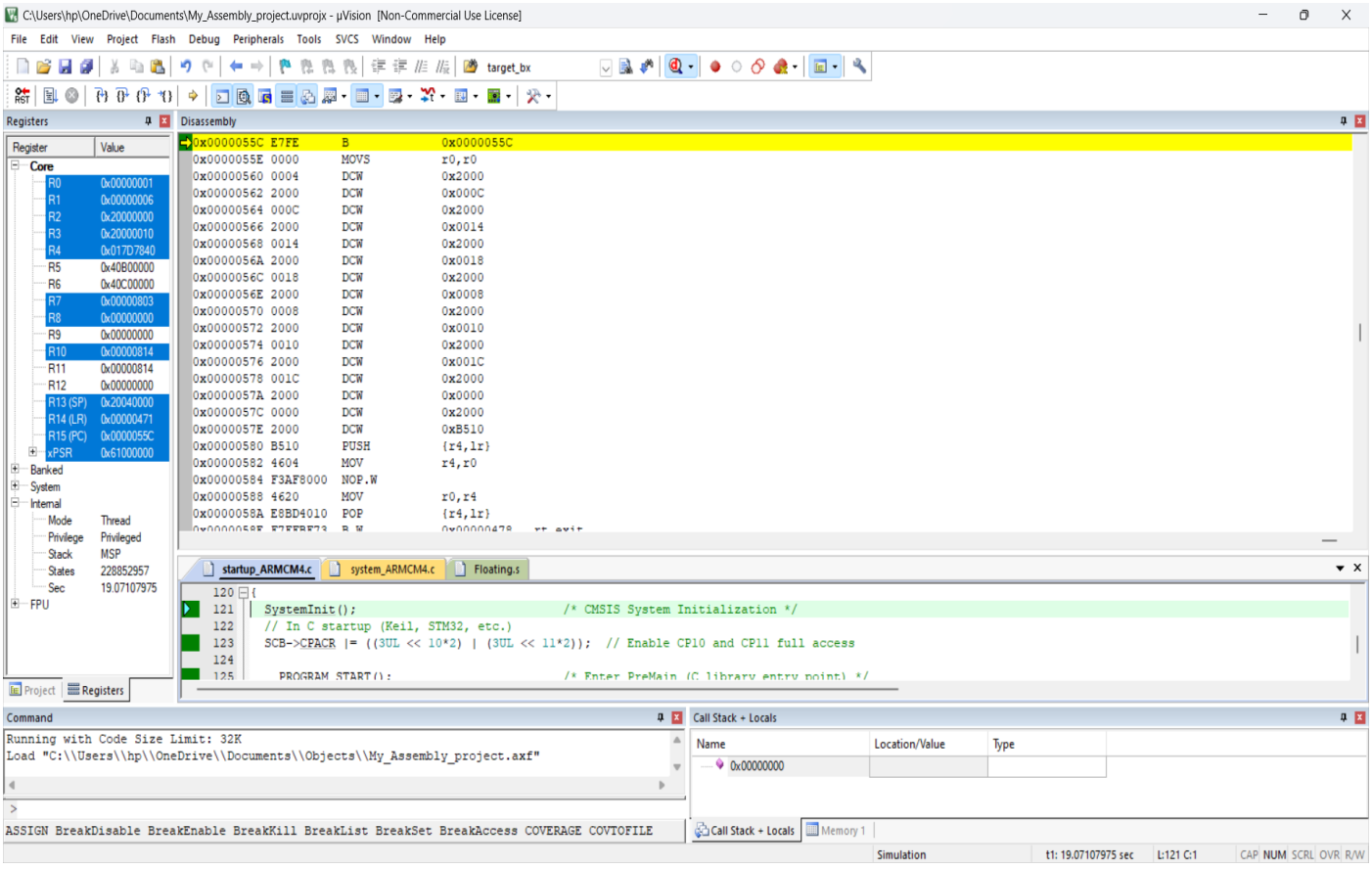
Compilation:



Build:



Execution:



Control_Flow.s

Description

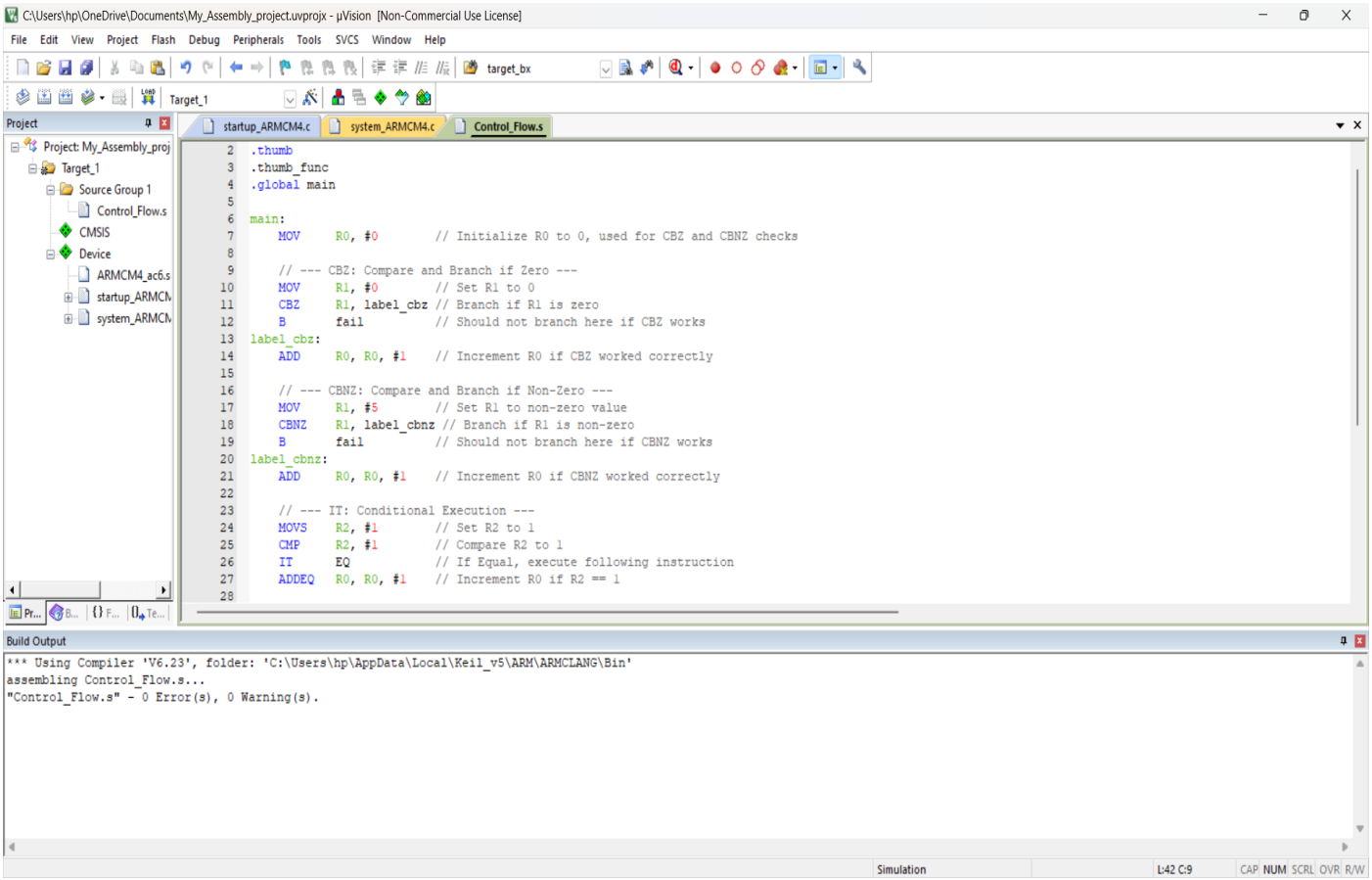
CBZ: If the register value is zero, branch to the specified label.
CBNZ: If the register value is non-zero, branch to the specified label.
IT: Conditional execution of the following instructions based on a condition.

Expected Test Result

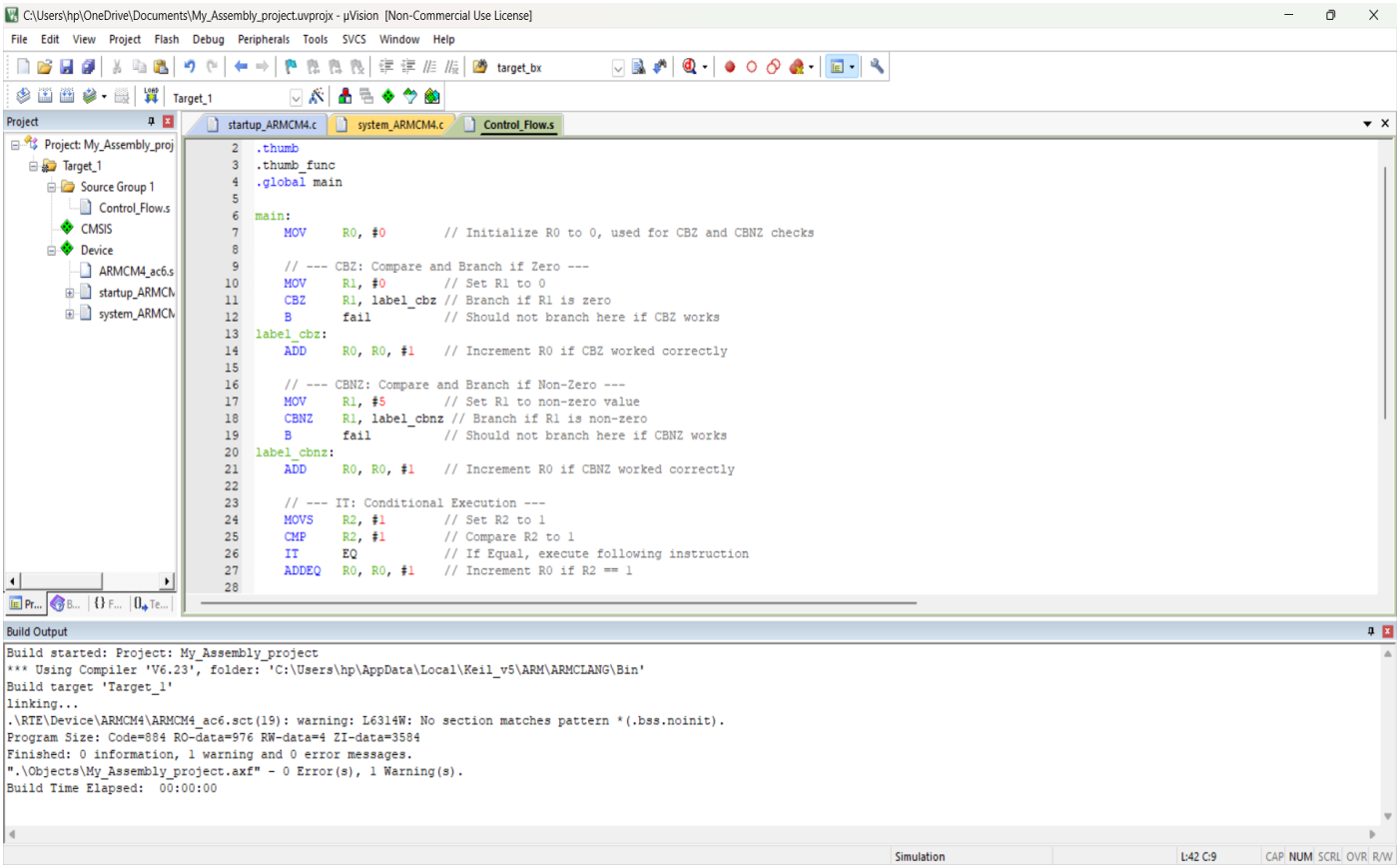
Check the value of R0:
0x01: success
0xFF: fail

Obtained Test Result

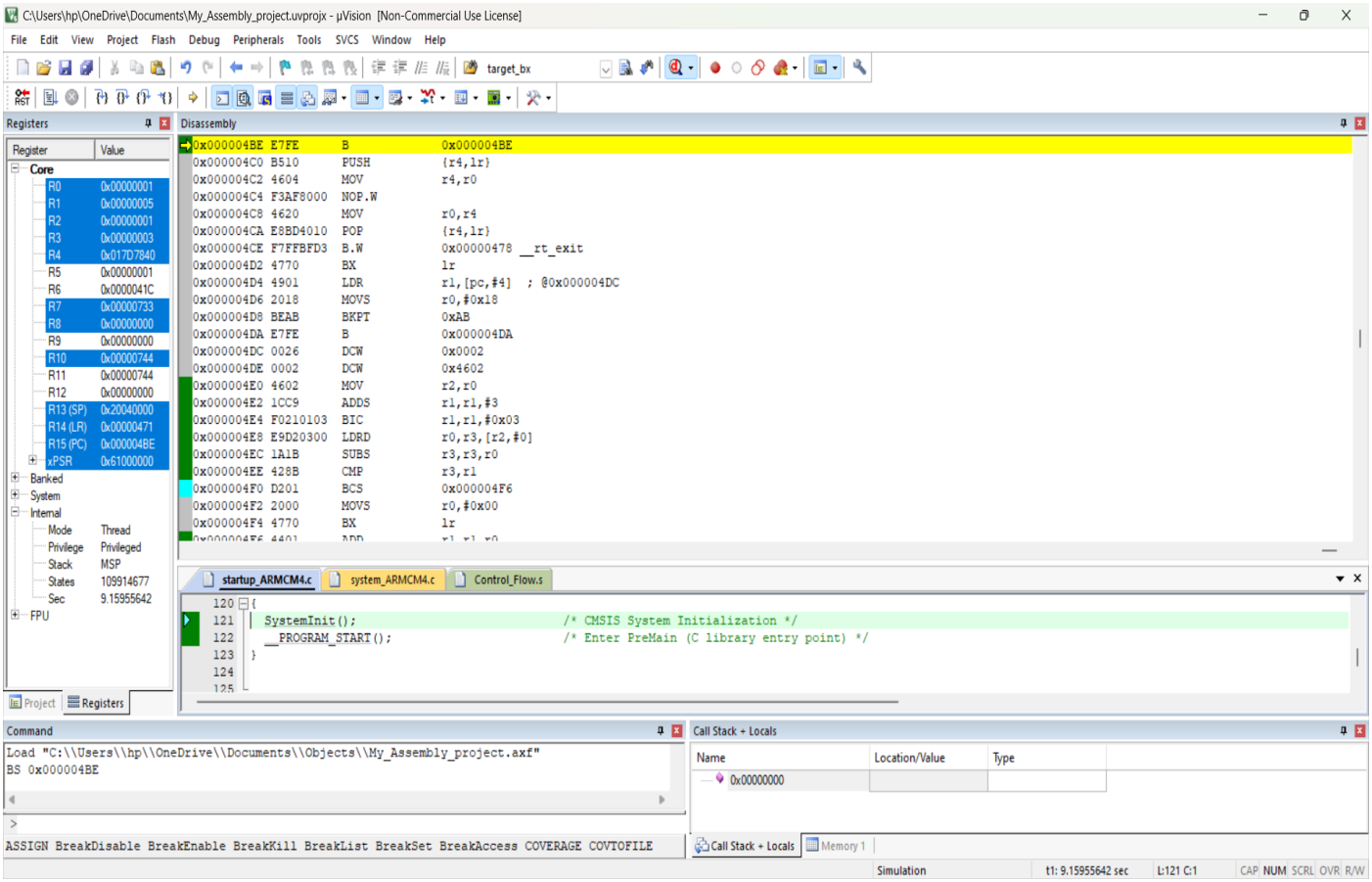
Compilation:



Build:



Execution:



Branching.s

Description

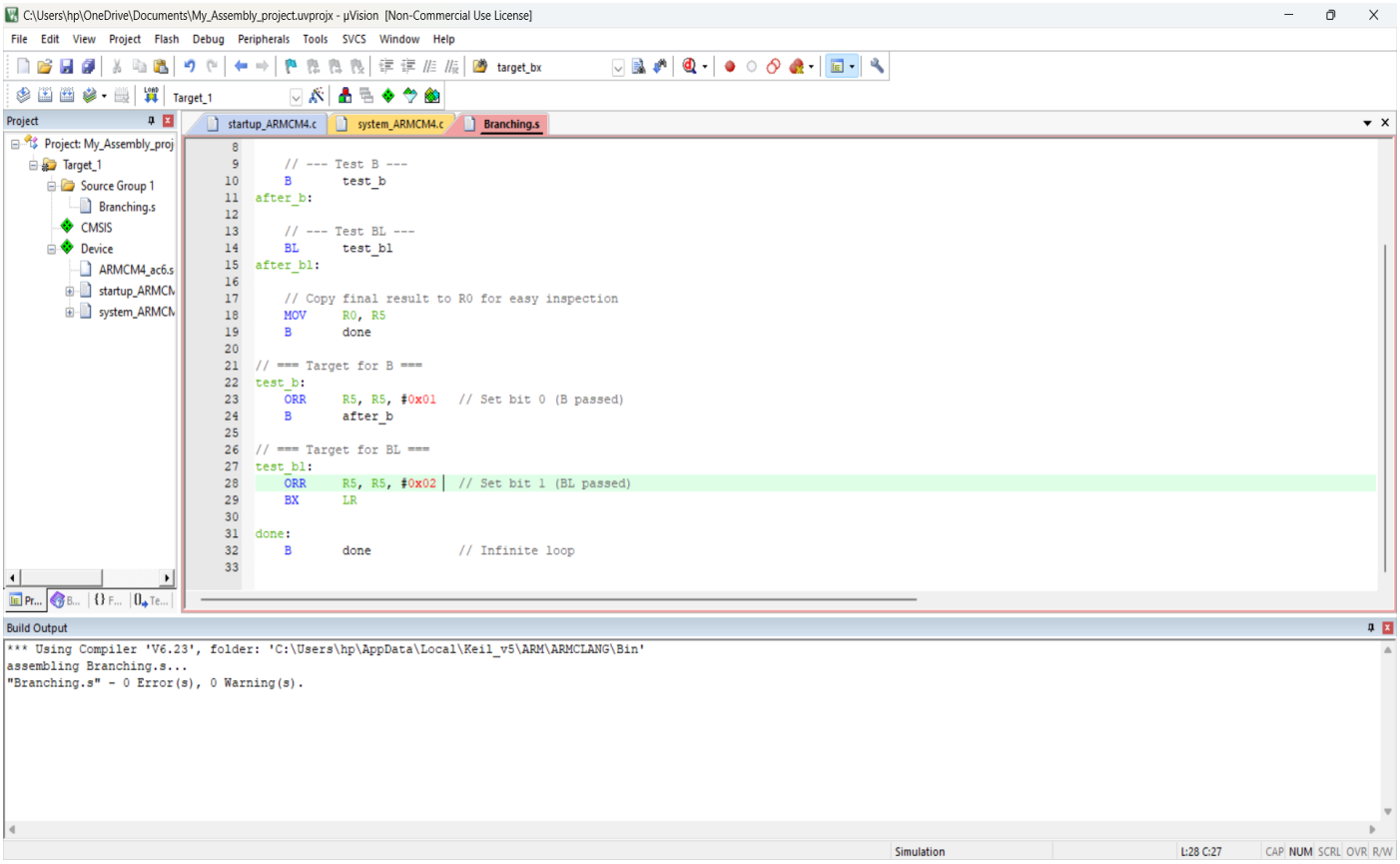
B: Unconditional branch
BL: Branch with link (calls a subroutine)

Expected Test Result

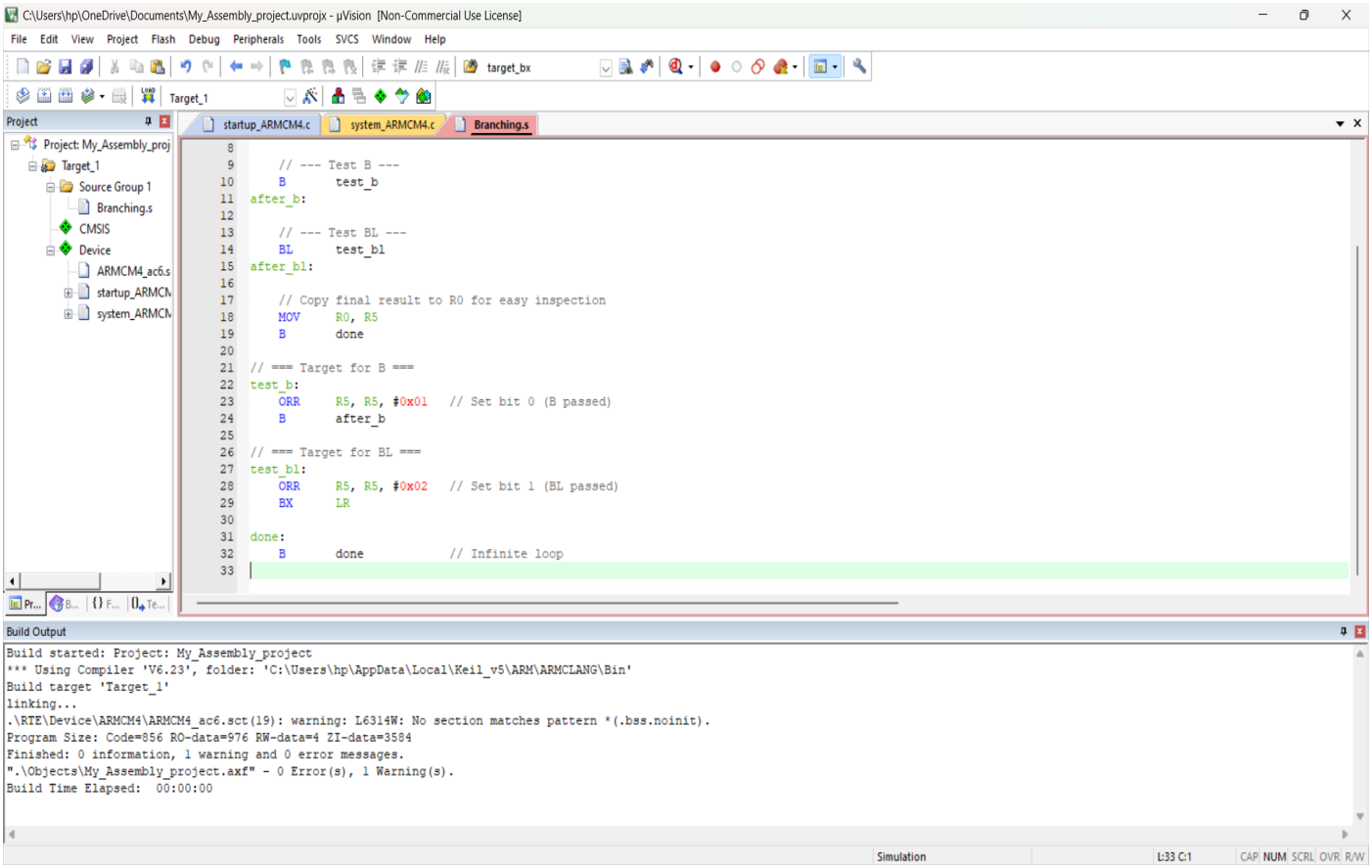
Check the value of R5 or R0:
0x01: only B passed
0x02: only BL passed
0x03: both B and BL executed correctly

Obtained Test Result

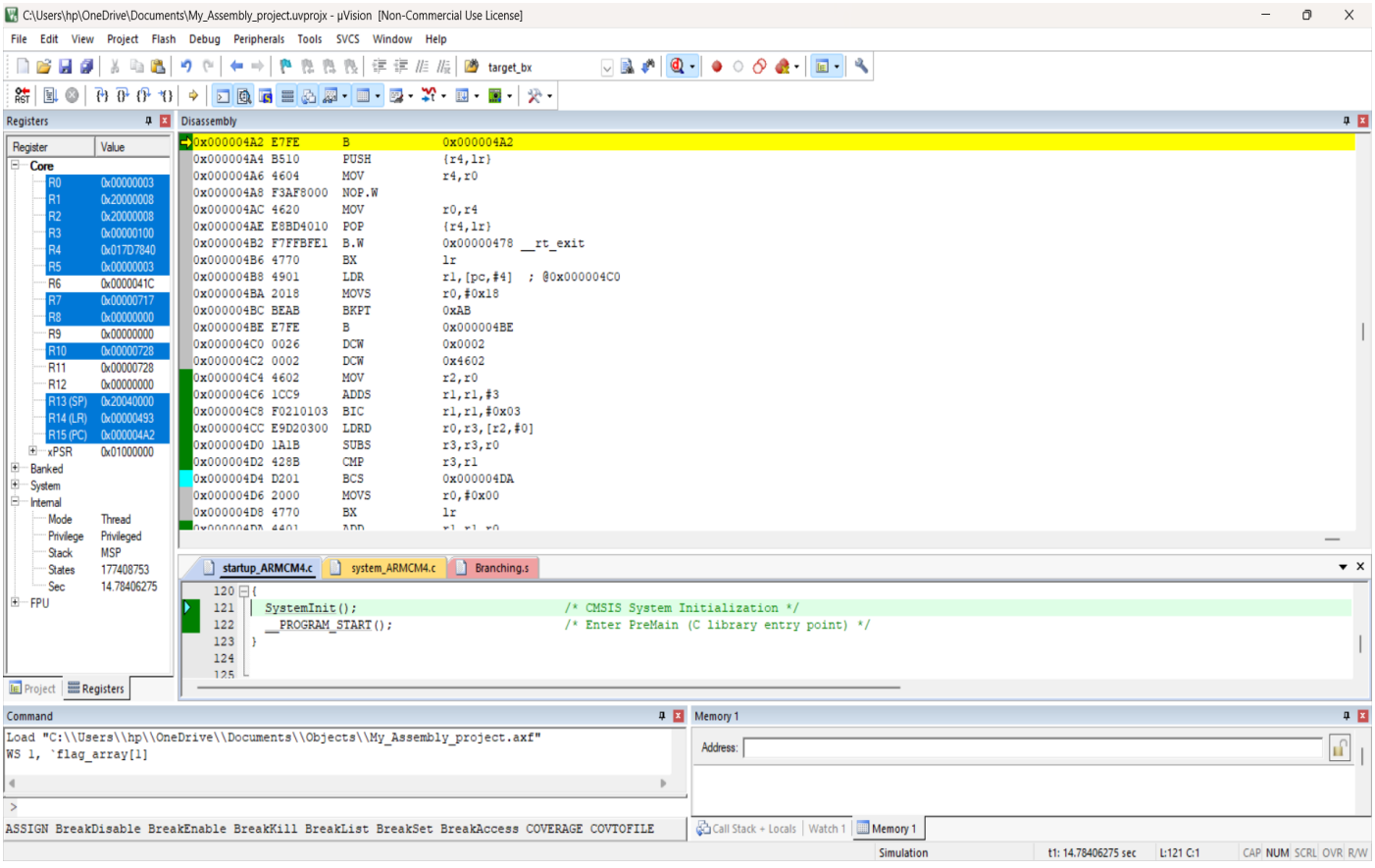
Compilation:



Build:



Execution:



Conclusion

By conducting this testing, it is ensured that the Proof-of-Concept (PoC) code functions as intended for the majority of instruction sets across various types within the Cortex-M processor series. This approach can be further extended to cover the remaining instruction sets as well.

Looking ahead, the same underlying logic can be adapted for other processor architectures, enabling validation of enhanced assembly code. This would contribute to improved safety and increased reliability in microcontroller operations.