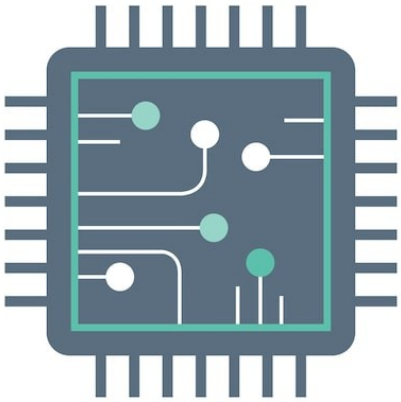
UNIVERSITÉ DU LUXEMBOURG

Circuits Numériques Project

Groupwork by Charles Biren, Hamza Bouziane and Mohamed Lemouissi



Name: BIREN CHARLES & BOUZIANE HAMZA & MOHAMED LEMOUISSI

Student ID: 0232644952, 0231668310, 0230593220

Subject: Circuits Numériques

Study Program: Bachelor Applied Information Technology

Professor: Bernard Steenis

Date of submission: January 16th 2026

Number of pages: 8

UNIVERSITÉ DU LUXEMBOURG

Declaration of authorship

We hereby declare that ***Charles Biren, Hamza Bouziane and Mohamed Lemouissi*** *are the sole authors of the work entitled "****MicroCPU_HCM_2026"****, submitted herewith.*

We confirm that the design and development of the MicroCPU architecture, the analysis of the instruction set, and the implementation of the CPU, memory, and interface modules were conducted by the authors. The understanding and interpretation of the provided laboratory specifications, the integration of the different hardware components, and the verification through simulation and FPGA synthesis were performed by the authors.

AI Assistance Disclaimer
This project was developed by the students and the provided files of Professor Bernard Steenis. An AI-based assistant was used as a support tool to clarify VHDL syntax, Vivado tool usage, and conceptual understanding of processor architecture. All design decisions, code integration, debugging, testing, and final validation were performed and verified by the students. The AI tool did not generate a complete solution autonomously.

All sources and aids used have been properly acknowledged. We confirm that this work has not been submitted previously, either in whole or in part, for any other course or assessment.

Date	Signature	Signature	Signature
18.01.2026			
	Charles BIREN	Hamza BOUZIANE	Mohamed LEMOUISSI
	First Name and Surname	First Name and Surname	First Name and Surname

Micro-CPU Project

Final Project Report

1. Introduction

This report presents the design, implementation, verification, and synthesis of a simple 8-bit MicroCPU based on the Von Neumann architecture. The processor was implemented in VHDL and deployed on a Basys 3 FPGA board (Artix-7).
The objective of this project was not only to produce a functioning microprocessor, but also to develop a rigorous understanding of:

- Control unit design using a finite state machine (FSM),
- Instruction fetch, decode, and execute sequencing,
- Data path design including registers and an Arithmetic Logic Unit (ALU),
- Memory organization and memory-mapped I/O,
- Verification through cycle-accurate simulation, and
- Hardware synthesis with resource utilization analysis.

The project was structured according to the seven laboratory tasks defined in the course (Labs 1–6), which guided a progressive and modular development of the system. This report emphasizes analytical reasoning, justification of design choices, and interpretation of simulation and synthesis results rather than merely describing implementation steps.

1.2Tools and Environment

The design and verification were conducted using the following environment:

- **Software:** AMD Vivado ML 2025.2
- **Hardware target:** Basys 3 FPGA (Artix-7 xc7a35tcpg236-1)
- **Hardware description language:** VHDL

Vivado was used for simulation, synthesis, implementation, and bitstream generation.

1.3 Reproducibility note

To reproduce this project, the following procedure should be followed:

1. Create a new Vivado project targeting the Basys 3 board (xc7a35tcpg236-1).
2. Add the provided VHDL source files (cpu.vhd, memory.vhd, interface.vhd, top_synth.vhd).
3. Add the testbench file (testbench.vhd) to Simulation Sources.
4. Set top_synth.vhd as the top module for synthesis.
5. Run behavioral simulation, followed by synthesis, implementation, and bitstream generation.

1.4 Design Trade-offs and Architectural Rationale (Professor-Level Section)

Control Unit: FSM vs. Microcode

A central design decision concerned the implementation of the control unit. Two classical approaches were considered:

1. **Finite State Machine (FSM)**
2. **Microcoded control unit (ROM-based sequencer)**

We selected an FSM-based control unit for the following reasons:

Advantages of FSM approach:

- Direct mapping to VHDL case statements, improving readability and maintainability.
- Easier debugging in simulation, as states explicitly correspond to architectural phases (fetch, decode, execute).
- Lower FPGA resource usage compared to a microcode ROM.

Disadvantages:

- Less flexibility than a microcoded approach.
- Harder to extend with new instructions without modifying the FSM structure.

Given the educational scope of this project, clarity and verifiability were prioritized over extensibility, making the FSM the most appropriate choice.

Instruction Register (IR) Width

The Instruction Register (IR) was defined as **24 bits**, capable of storing up to three instruction bytes. This decision was justified by the instruction set specification, which allows instructions of 1, 2, or 3 bytes.

Alternative approaches could have included:

- Fetching and decoding bytes directly from memory without storing them in a full IR.

However, this would have complicated control logic and introduced additional memory access cycles. Storing the full instruction in IR simplifies decoding and ensures deterministic execution behavior.

Segmented Program Counter (PC) Increment

The Program Counter (PC) was implemented such that only its **least significant byte (LSB)** is incremented during normal execution. This reflects a segmented memory model in which programs execute within a fixed memory page.

This design choice:

- Simplifies address computation,
- Aligns with the structure of the BIOS stored in page 0,
- Reduces hardware complexity.

However, it limits the size of continuous programs unless explicit instructions modify the page (MSB of PC).

2. Summary of Preliminary Labs (Labs 1–4)

Labs 1 to 4 established the functional foundation of the MicroCPU.

Lab 1 – Memory and BIOS
A dual-purpose memory module was implemented containing:

- BIOS program stored in page 0,
- General-purpose RAM,
- Support for instruction fetch,
- Memory-mapped I/O interface.

This memory model follows the Von Neumann paradigm, where instructions, data, and peripherals share a unified address space.

Lab 2/3 – CPU Core and Register File

The CPU core was built around a five-state FSM controlling instruction execution:

State	Meaning
0	Reset / Initialization
1	Fetch first byte of instruction
2	Fetch second byte (if needed)
3	Fetch third byte (if needed)
4	Execute instruction

A register file of **8 general-purpose 8-bit registers** was added, enabling:

- Register-to-register transfers,
- Memory load/store operations,
- ALU computations.

Lab 4 – Control Flow Instructions

Conditional and unconditional jump instructions were implemented using:

- A FLAGS register (R1),
- Program Counter (PC) updates,
- Three addressing modes:
    - Indirect,
    - Mixed,
    - Direct.

These labs were validated using behavioural simulations and provided the necessary execution flow required for the final project.

3. Lab 5 – Arithmetic Logic Unit (ALU)

3.1 ALU Architecture

The ALU was designed as a combinational unit operating on:

- **Operand 1:** Accumulator register R0
- **Operand 2:** Either another register or an immediate constant

Supported operations include:

- Arithmetic: ADD, SUB, CMP
- Logical: AND, OR, XOR
- Unary: NEG, NOT, INC, DEC
- Shifts and Rotations: SHL, SHR, ROTL, ROTR

Each operation produces:

- An 8-bit result,
- Four status flags stored in R1:
  - **Z** (Zero)
  - **S** (Sign)
  - **C** (Carry)
  - **O** (Overflow)

3.2 Instruction Decoding

Three ALU instruction formats were implemented:

- **Instruction 9**: ALU with immediate constant (2 bytes),
- **Instruction 10**: ALU with register operand (1 byte),
- **Instruction 11**: ALU unary operations (1 byte).

Instruction decoding occurs during the fetch phase, with control signals forwarded to a combinational ALU process.

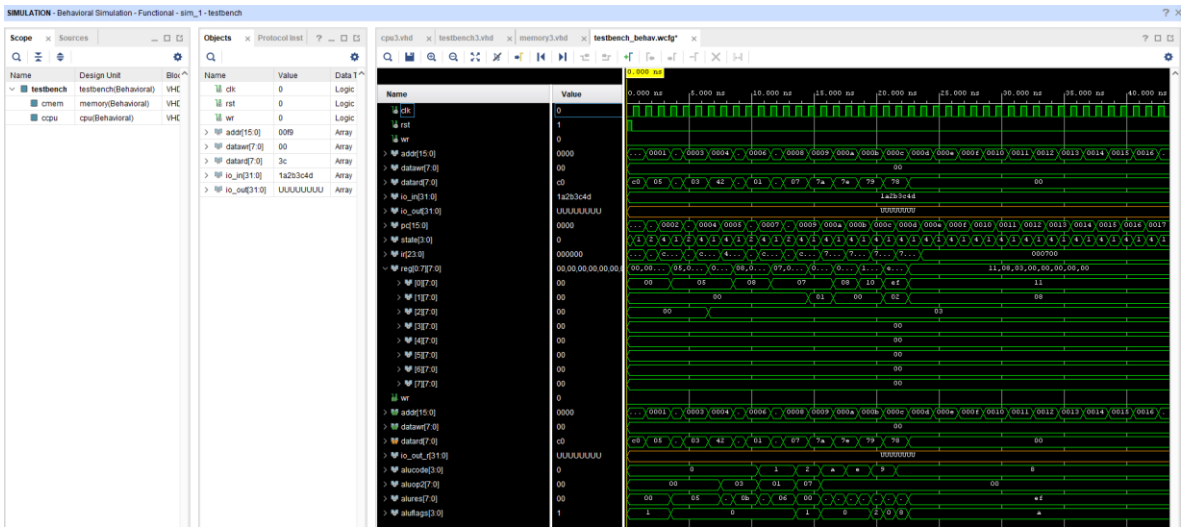
3.3 ALU Combinational Process

Behavioral simulation validated the correctness of the ALU and control unit. A typical two-byte instruction follows this sequence:

1. **State 1 (Fetch MSB):**
   - IR(23:16) ← DataRd
   - PC incremented (LSB only)
2. **State 2 (Fetch LSB):**
   - IR(15:8) ← DataRd
   - PC incremented
3. **State 4 (Execute):**
   - ALU computes result
   - Register or flags updated

This confirmed that the FSM correctly sequences multi-cycle instructions.

Screenshot of the Behavioural simulation from Lab 5



4. Lab 6 – Final Integration and FPGA Implementation

4.1 BIOS Integration and Analysis

The BIOS stored in memory served as both an initialization routine and a self-test program.
A simplified instruction-level interpretation is as follows:

Address	Instruction	Purpose
00	LOAD R6, #00	Initialize pointer MSB
01	LOAD R7, #FC	Initialize pointer LSB
02	LOAD R0, #48	Load test value
03	STORE [R6:R7], R0	Write to memory
04	JUMP 0204	Transfer control to test routine

The BIOS systematically:

- Initializes registers,
- Tests LOAD/STORE operations,
- Validates indirect and direct jumps,
- Interacts with memory-mapped I/O using buttons and switches.

Thus, the BIOS acts as a diagnostic and verification program rather than a simple bootloader.

4.2 Testbench Simulation

The testbench was extended to include:

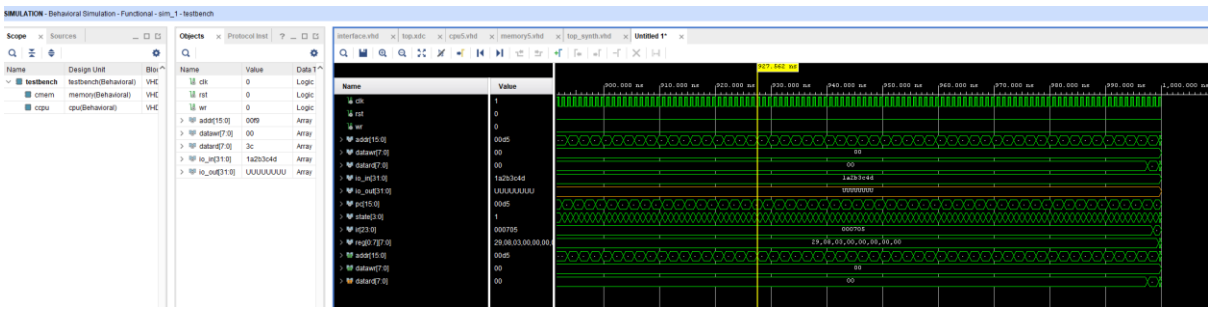
- Simulated button and switch inputs,
- Observation of CPU and memory signals,
- Monitoring of internal state, PC, and IR.

Simulation was executed over thousands of cycles to ensure coverage of:

- BIOS memory browsing,
- Execution of a demonstration program,
- Various input scenarios.

Results demonstrated correct:

- Instruction execution,
- Memory access behavior,
- Register and flag updates,
- I/O interaction.



4.3 Interface Integration

An interface module was integrated to map:

- switches and buttons to io_in,
- LEDs and 7-segment displays to io_out.

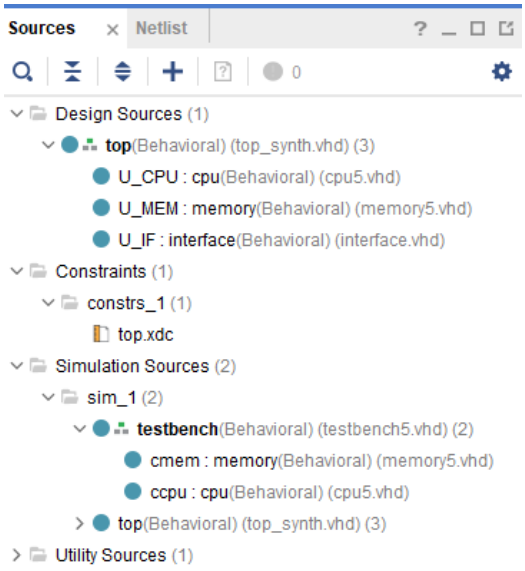
This enables interaction with the CPU through the Basys3 FPGA board.

4.4 Top-Level Module

A top-level VHDL module was written to instantiate:

- CPU,
- memory,
- interface.

All interconnections follow the project specification.
A constraint file (.xdc) was added to map logical signals to FPGA pins.



4.5 Synthesis, Implementation, and Bitstream Generation

The design was successfully:

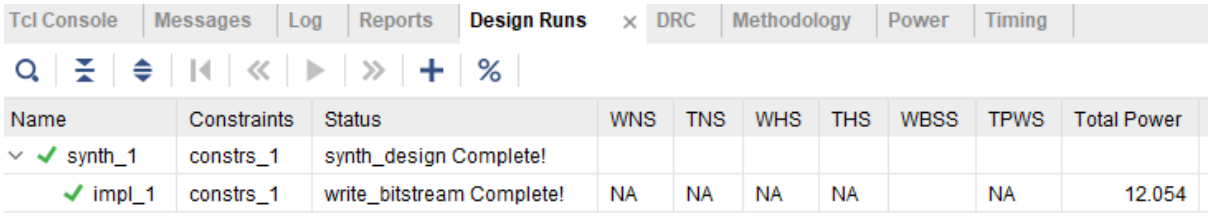
- synthesized,
- implemented,
- routed,
- and a bitstream was generated.

Resource Utilization (Vivado – Synth Design Report):

- **32 Flip-Flops:** Corresponding to peripheral output registers.
- **2 Block RAM (BRAM) tiles:** Each 36 kbit, implementing a total of 64 kbit of memory.

The explicit presence of BRAM in the synthesis report confirms correct hardware inference of RAM rather than distributed logic implementation.
Although Vivado reported warnings related to missing timing constraints, these do not impact functional correctness in the context of this educational project.

Minor Vivado methodology warnings were reported due to missing timing constraints; however, they do not affect functional correctness for this project.



Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power
synth_1	constrs_1	synth_design Complete!							
impl_1	constrs_1	write_bitstream Complete!	NA	NA	NA	NA		NA	12.054

5. Source Files

The following source files were used in the final design:

- cpu.vhd
- memory.vhd
- interface.vhd
- top_synth.vhd
- testbench.vhd
- top.xdc
- *The generated bitstream file (top.bit) is included with the submission.*

6. Final Conclusion

This project resulted in a fully functional MicroCPU implemented in VHDL and synthesized for FPGA deployment.
Through progressive development, the CPU evolved from basic instruction execution to a complete system with ALU, control flow, memory, and I/O integration.

Behavioural simulations and FPGA synthesis confirmed the correctness of the design.
This project provided valuable insight into processor architecture, hardware description languages, and FPGA design workflows.