
The German University in Cario (GUC)

Faculty of Media Engineering Technology

Computer Science and Engineering

Computer System Architecture – CSEN 601

---

# Milestone 02

# Package Number: 3

# Team Number: 10

---

Team Members:

Mahmoud Magdy, 58-24197, T20

Mahmoud Mohamed, 58-20153, T20

Mohamed Ehab, 58-8804, T20

Ramez Youssef, 58-12387, T20

Moustafa Mohamed, 58-11800, T30

Omar Mohamed Zakaria, 58-5054, T30

Under Supervision of:

Dr.Eng. Catherine M. Elias

## 1. Introduction

This report explains the design and operation of a CPU simulator that follows a Harvard architecture and implements a three-stage instruction pipeline. The simulator mimics a simplified 8-bit CPU using C programming language. The main components include instruction fetch, decode, execute stages, a status register (SREG), and support for basic arithmetic, logic, memory, and control flow instructions.

---

## 2. Memory and Register Structure

The CPU architecture is Harvard-based, meaning it has separate instruction and data memory spaces:

**Instruction Memory**: 1024-word array holding 16-bit instructions.

**Data Memory**: 2048-byte array for runtime data storage.

**Registers**: 64 general-purpose 8-bit registers (R0 to R63).

**Program Counter (PC)**: Tracks the current instruction address.

**Status Register (SREG)**: An 8-bit register storing condition flags resulting from operations.

---

## 3. Instruction Format

Each instruction is 16 bits:

Bits 15–12: Opcode (identifies the operation)

Bits 11–6: Destination or source register (r1)

Bits 5–0: Second register or immediate value (r2 or imm)

The instructions are parsed from an external file and loaded into instruction memory during initialization.

---

### 4. SREG (Status Register) Description

The Status Register (SREG) stores five primary flags:

**Z (Zero)**: Set if the result is zero.

**S (Sign)**: Set if the result is negative (N XOR V).

**N (Negative)**: Set if the most significant bit of the result is 1.

**V (Overflow)**: Set if a signed overflow occurs.

**C (Carry)**: Set if an unsigned overflow occurs.

These flags are updated during arithmetic and logic operations and are used for conditional branching.

---

### 5. Instruction Execution Pipeline

The simulator uses a three-stage pipeline:

**Fetch Stage**

Retrieves the next instruction from instruction memory using the current PC.

If no valid instruction is found or PC exceeds memory size, fetch stage becomes inactive and program termination begins.

After fetching, the PC is incremented.

**Decode Stage**

Interprets the instruction fetched.

Extracts opcode, register numbers, and immediate values.

Identifies the instruction mnemonic based on the opcode.

Prepares operands for the execution stage.

Immediate values are sign-extended when necessary.

**Execute Stage**

Carries out the operation based on the decoded instruction.

Handles arithmetic, logic, memory access, or control flow.

Modifies registers or memory accordingly.

Updates SREG flags based on the result.

Supports control flow operations such as conditional branches and jumps.

---

**6. Pipelining and Control Hazards**

Pipelining allows simultaneous processing of multiple instructions in different stages. This increases execution efficiency by overlapping the stages.

However, control hazards arise with branch instructions (BEQZ and BR). If a branch is taken:

The pipeline is flushed by deactivating both decode and execute stages.

The PC is updated to the new branch target address.

Fetching resumes from the new PC.

This flushing ensures that instructions fetched after the branch but before confirming it were not executed erroneously.

---

## 7. Supported Instructions

The CPU supports the following instructions:

| Mnemonic | Description |
|---|---|
| ADD | Add two registers |
| SUB | Subtract one register from another |
| MUL | Multiply two registers |
| MOVI | Move immediate value into register |
| BEQZ | Branch if register value equals zero |
| ANDI | Bitwise AND between register and immediate |
| EOR | Bitwise XOR between two registers |
| BR | Unconditional jump |
| SAL | Shift left arithmetic |
| SAR | Shift right arithmetic |
| LDR | Load from memory into register |
| STR | Store register value into memory |

---

## 8. Program Lifecycle

The following steps occur during program execution:

**Initialization**:

All registers and memory are zeroed.

PC is set to 0.

Instructions are loaded from an external text file.

**Pipeline Execution**:

A loop repeatedly invokes the pipeline cycle.

Each cycle processes stages in order: Execute → Decode → Fetch.

If all stages are inactive and no further instructions are left, the program terminates.

**Final Output**:

After execution, the system prints:

Final PC value

SREG status

Register values

Data memory values (non-zero only)

Loaded instruction memory

---

## 9. Conclusion

This CPU simulator demonstrates how a simplified 3-stage pipeline can be used to implement concurrent instruction execution. It accurately simulates instruction decoding, flag updating, and memory/register management. Additionally, it models control hazards and the effect of pipeline flushing due to branching. The use of separate instruction and data memory spaces reflects the Harvard architecture design. The implementation is suitable for educational purposes in computer architecture and operating system courses.