Indian Institute of Technology Kharagpur

AUTUMN Semester, 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Computer Organization and Architecture Laboratory

Final Program Demonstration: Algorithmic Implementation

November 11, 2025

Instructions: *This document outlines the final program demonstration. The objective is to implement and execute* ***two (2)*** *mandatory algorithms as separate programs. This will serve as a capstone test of your processor's instruction set, control unit, and memory subsystems.*

Context and Objective

Having demonstrated autonomous program execution, the final step is to prove the robustness and completeness of your processor. You will create two separate programs, one for each algorithm below. Each program will be loaded from an instruction ROM and operate on initial data from a read/write data RAM. Each program will execute, store its final 32-bit result, and then halt. The results will be displayed on the LEDs for verification.

FPGA Task: Implement Both Algorithms

You will create two separate projects and generate two separate '.bit' files.

- **Project 1 (Booth):** A project whose instruction ROM ('.coe' file) contains the Booth's algorithm and stores its result in 'R2'.
- **Project 2 (Hamming):** A project whose instruction ROM ('.coe' file) contains the Total Hamming Weight algorithm and stores its result in 'R3'.
- **Data Memory (RAM):** For each project, you must use a BRAM IP configured as read/write RAM. This RAM will be initialized from a '.coe' file. **Note:** The TAs will provide their own '.coe' files with different input values to test the generality of your program.
- **Instruction Memory (ROM):** Each project will have its own BRAM IP for instruction ROM, initialized with its specific program's machine code.

Problem 1: Booth's Multiplication

Implement Booth's multiplication algorithm to multiply two 16-bit signed integers, producing a 32-bit result.

- **Input:** Load the multiplicand M and multiplier R from designated addresses in the data memory.
- **Task:** Implement Booth's algorithm.
- **Output:** The final 32-bit product must be stored in register 'R2'.

Pseudocode for Booth's Algorithm (16-bit):

```
M = Multiplicand (from data memory)
R = Multiplier (from data memory)
A = 0
Q_minus_1 = 0
Count = 16

while Count > 0:
    Q_0 = LSB of R
    if Q_0 == 1 and Q_minus_1 == 0:
        A = A - M
    if Q_0 == 0 and Q_minus_1 == 1:
        A = A + M

    # Arithmetic shift right (A, R, Q_minus_1)
    Q_minus_1 = LSB of R
    R = R >> 1 (arithmetic shift)
    LSB of A -> MSB of R
    A = A >> 1 (arithmetic shift, preserve sign bit)

    Count = Count - 1

# Final 32-bit result is in {A, R}
# Combine {A, R} into R2 for the result
```

Problem 2: Total Hamming Weight

Compute the total Hamming weight of a 5-word array in memory.

- **Input:** A 5-word (32-bit) array of numbers loaded from the data memory.
- **Task:** Implement an iterative algorithm that loads each number, calculates its Hamming weight, and sums the weights. **You must use the 'HAM' instruction** from the ISA specification.
- **Output:** The final 32-bit total sum of the Hamming weights must be stored in register 'R3'.

Pseudocode for Total Hamming Weight:

```
// R3 will store the total sum
// R4 = loop counter, R5 = current address
// R6 = temp for data, R7 = temp for ham_weight

ADDI R3, R0, 0     // R3 = TotalSum = 0
ADDI R4, R0, 5     // R4 = Counter = 5
ADDI R5, R0, 24    // R5 = CurrentAddress = 24

LOOP:
    LD R6, 0(R5)       // R6 = Mem[CurrentAddress]
    HAM R7, R6         // R7 = HammingWeight(R6)
    ADD R3, R3, R7     // TotalSum = TotalSum + R7

    ADDI R5, R5, 8     // CurrentAddress += 8
    SUBI R4, R4, 1     // Counter--
    BPL R4, LOOP       // Branch to LOOP if Counter > 0
```

```
# Final result is in R3
# (Processor proceeds to HALT)
```

Hardware Pin Assignments (Nexys4 DDR / Nexys A7)

This setup is for resetting the processor and viewing the 32-bit result.

Table 1: Required Pinout for the Nexys4 DDR / Nexys A7 Board

Signal in Verilog	FPGA Pin	Description
`clk`	`E3`	100MHz System Clock Input
`reset`	`BTNC`	CPU Reset Button (Active-High)
`led[15:0]`	`H17, K15, ...`	16 LEDs (Data Output)
`sw[0]`	`J15`	Display Select: (0=Lower 16 bits, 1=Upper 16 bits)

Suggested Demonstration Flow

1. **Demonstration for Problem 1 (Booth's):**
   - Briefly explain your assembly code and show the data '.coe' file (provided by the TA) for Problem 1.
   - Load the FPGA with the generated bit file for **Problem 1**.
   - Press the designated `reset` **button** ('BTNC') once to run the program.
   - Explain that the program has run to completion and is now in the 'HALT' state.
   - Show the TA the final 32-bit result (from register `R2`) using the `sw[0]` switch to select the upper/lower 16 bits.
2. **Demonstration for Problem 2 (Hamming):**
   - Briefly explain your assembly code and show the data '.coe' file (provided by the TA) for Problem 2.
   - Load the FPGA with the generated bit file for **Problem 2**.
   - Press the designated `reset` **button** ('BTNC') once to run the program.
   - Explain that the program has run to completion and is now in the 'HALT' state.
   - Show the TA the final 32-bit result (from register `R3`) using the `sw[0]` switch to select the upper/lower 16 bits.
3. The TA will verify that the results displayed for both problems match the expected output for their custom input values.