CG2028 Assignment Report

Richard Willie (A0219710L)
Hu Jialun (A0211251B)

Group B01 7

1 Overview

This report describes the design and implementation of an assembly language solution to the given assignment. To summarize, the assembly program performs the k-nearest neighbours algorithm (k-NN) to solve a classification problem, with the following constraints:

- The distances between the sample data and each training point are unique.
- Find the nearest k points, where k is always 1.
- The squared of the distances is within 32 bits.
- All the machine code follows the encoding format given in Lecture 4.

2 Usage of Registers

When the assembly language function classification () is called, the C program passes 4 parameters (N, points, label, sample) to register R0, R1, R2, and R3. The return value of the function is passed back to the C program through the register R0. [1] The mapping of each register is as follows:

- **R0**: Stores the class of the nearest point.
- **R1**: Stores the memory address of points[0].
- **R2**: Stores the memory address of label[0].
- **R3**: Stores the memory address of sample[0].
- **R4**: Multiple duties during different stages of the program:
  - Stores the loop counter before passing its value to R6.
  - Stores the y-coordinate of the neighbouring data points, i.e., points[$2i+1$] where $i = N-1, \ldots, 0$.
  - Stores the y-difference of the data points to the sample point, i.e., $y_p - y_s$.
  - Stores the squared of the distances from the neighbouring points, i.e., $d^2 = (x_p - x_s)^2 - (y_p - y_s)^2$.
- **R5**: Multiple duties during different stages of the program:
  - Stores the y-coordinate of the sample point, i.e., sample[1].
  - Stores the x-coordinate of the neighbouring data points, i.e., points[$2i$] where $i = N-1, \ldots, 0$.
  - Stores the x-difference of the data points to the sample point, i.e., $x_p - x_s$.
  - Stores the squared of the x-differences, i.e., $(x_p - x_s)^2$.
- **R6**: Stores $i = N-1, \ldots, 0$. It is used as a counter for the loop.
- **R7**: Stores the x-coordinate of the sample point, i.e., sample[0].
- **R12 (IP)**: Stores the smallest distance from the neighbouring points.

3 Implementation Details

The flowchart below describes the high-level view of the program architecture. The intra-procedure call scratch register (IP, R12) is used since it is caller-saved as per ARM EABI [1], thus reducing the stack push/pop operation by two.

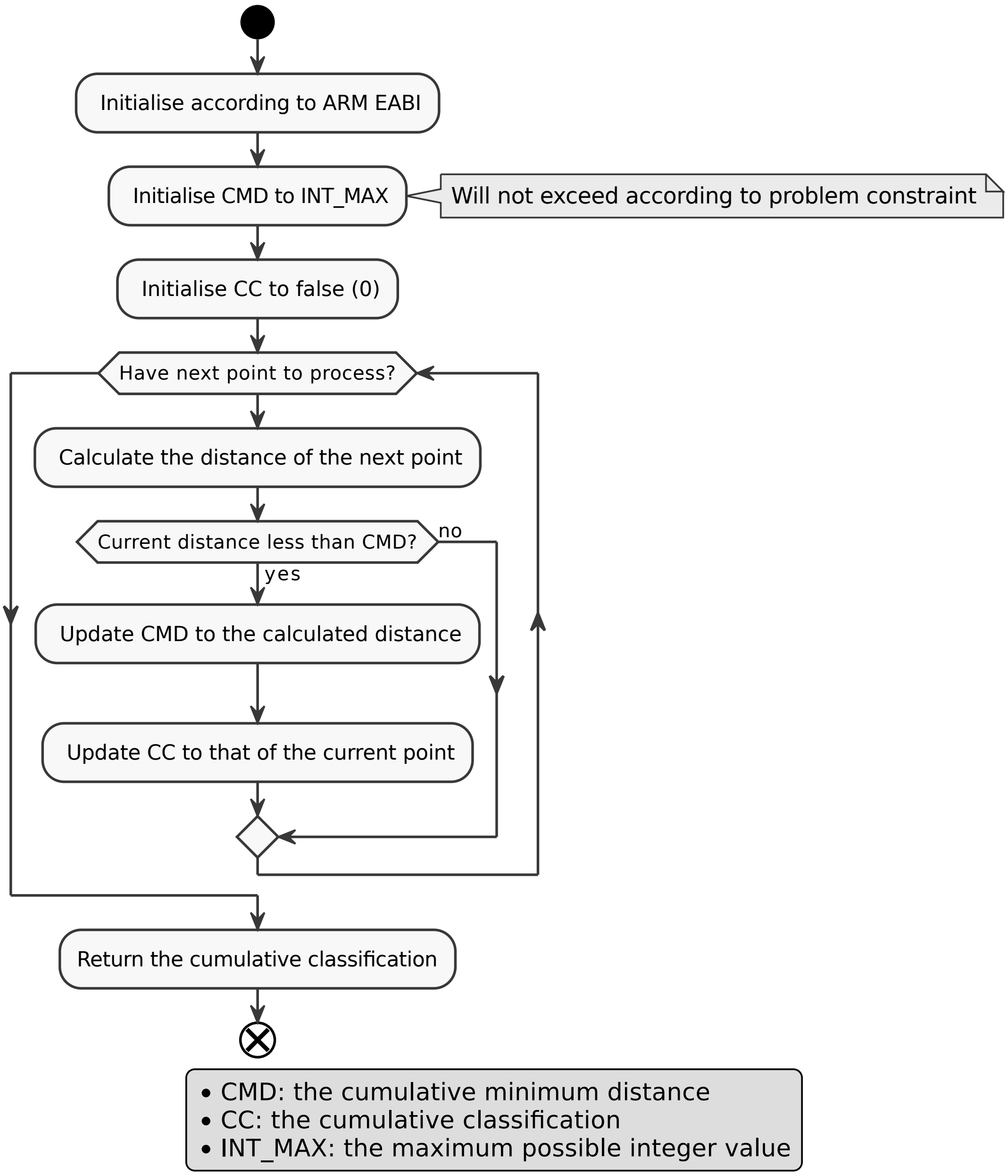

Figure 1: Flowchart

4 Microarchitecture Design

The diagrams below illustrate the microarchitecture design (modified from Lecture 4, page 28) that supports MUL and MLA instructions. Both MUL and MLA diagrams have the same hardware specifications. The modifications are drawn in red color. All other parts remain exactly the same as the microarchitecture diagram given in Lecture 4, page 28. The orange color highlighting indicated the datapath of the MUL and MLA operations.

The decoder takes one more input which is M (bit 4 of the instruction). M denotes whether the operation is of multiplication or not. In particular, the input M = 1 if multiplication instruction is performed and M = 0 if otherwise.

Conditions for multiplication to be performed:

- Data processing instruction: $op == 00$.
- No immediate value: $I == 0$.
- Multiplication instruction: $M == 1$.

The decoder will also send out an additional *MultControl* signal, which is a 4-bit signal that contains the *cmd* of MUL (0b0000) or MLA (0b0001) to a multiplexer to choose between MUL or MLA operation.

To be more precise:

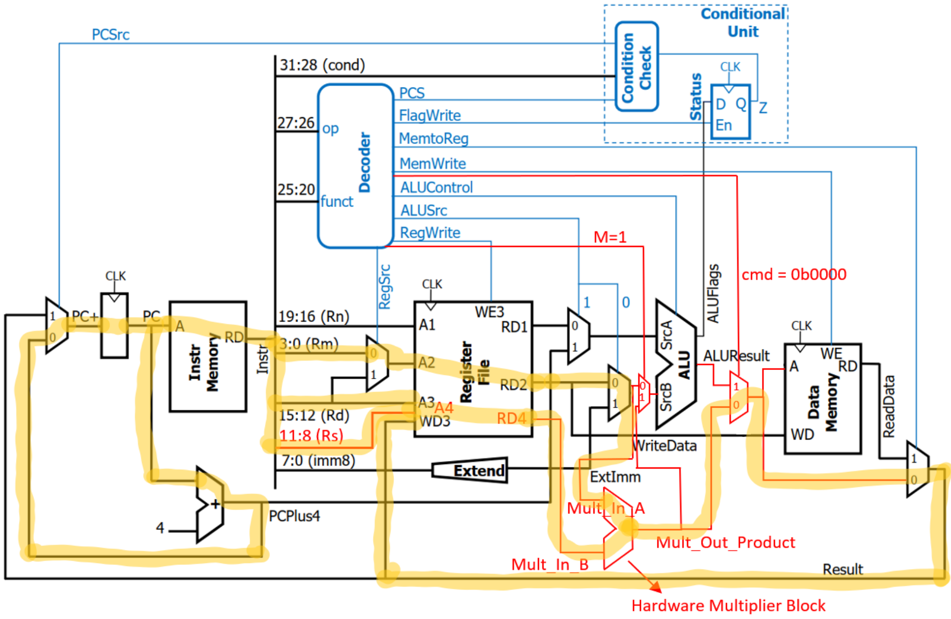

Figure 2: MUL

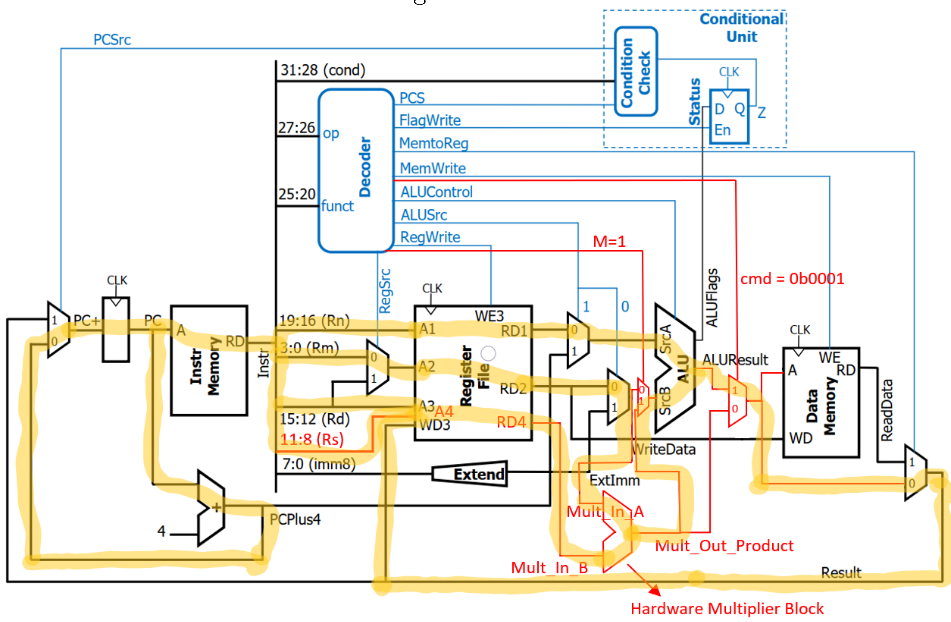

Figure 3: MLA

- $MultControl == 0000$ if $((op == 00)$ && $(I = 0)$ && $(M == 1)$ && $(cmd == 0000))$
- $MultControl == 0001$ if $((op == 00)$ && $(I = 0)$ && $(M == 1)$ && $(cmd == 0001))$

Furthermore, we have an additional input register R_s (bits 11:8 of the instruction). This is why the register file have one extra read port (A4) and output port (RD4). Consequently, to ensure that the ALU is able to perform addition as a part of the MLA instruction and also continue to perform all other normal data processing instructions, the $ALUControl$ has to be modified from the original into the following:

- $ALUControl = (op == 00)\ ?\ ((I == 0$ && $M == 1)\ ?\ 0100 : cmd) : (U\ ?\ 0100 : 0010)$

References

[1] "Procedure call standard for the arm architecture," https://developer.arm.com/documentation/ihi0042/latest, Arm Limited, Tech. Rep., July 2020.