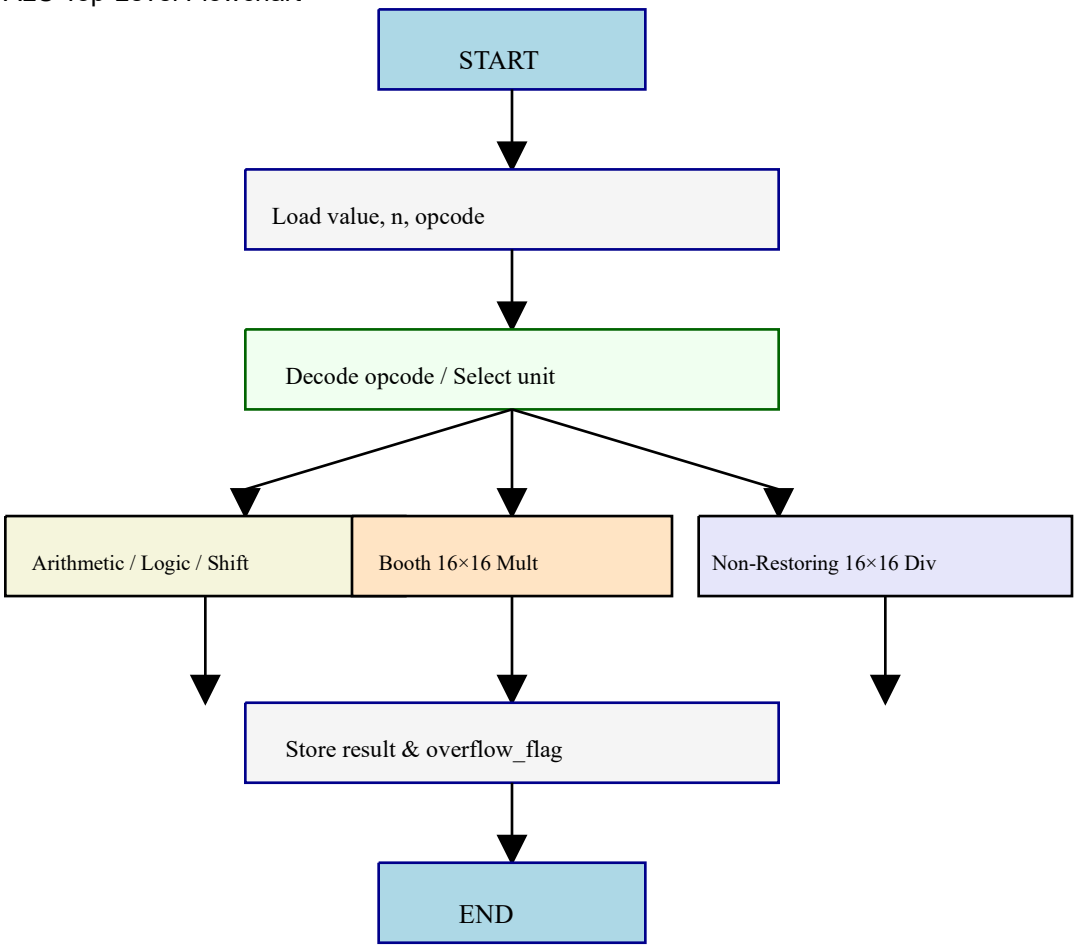
32-bit ALU Design on Cortex-M4

With Booth Multiplication, Non-Restoring Division, Bit Manipulation & Extended Operations

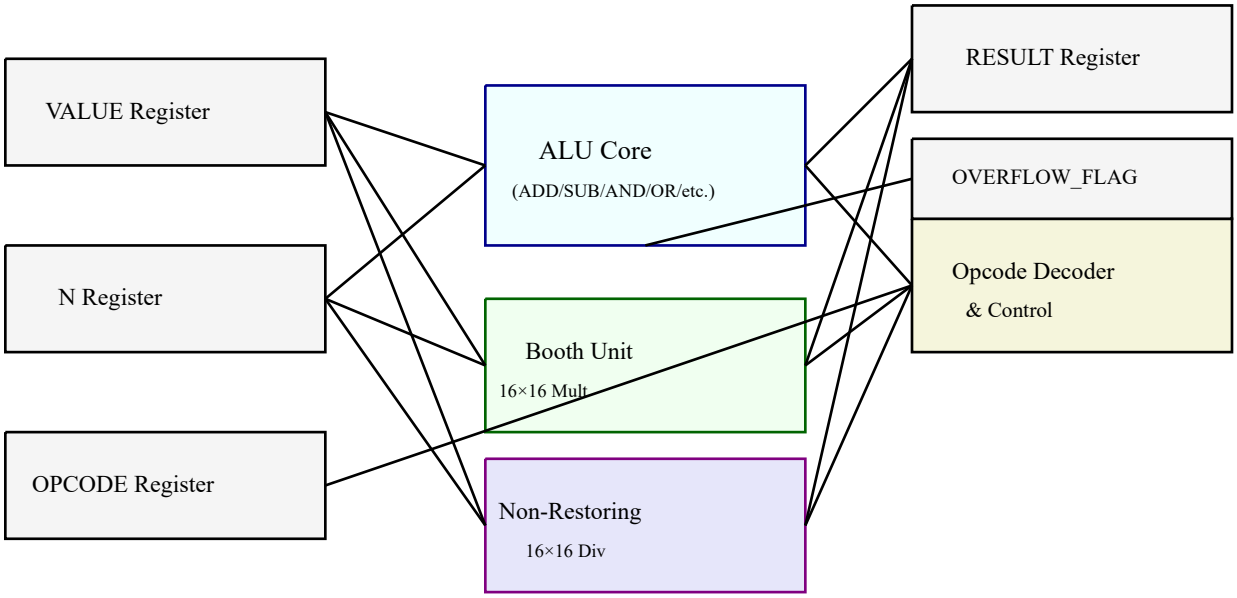
Author: Mayesha Binte Liton
Course Code: CSE-2105
Course Title: Microprocessor and Computer Architecture

This report documents the design and implementation of a 32-bit Arithmetic Logic Unit (ALU) in ARM assembly for the Cortex-M4 processor. The ALU supports 30 different operations, including arithmetic, logic, shifting, rotation, advanced multiplication (Booth's algorithm), non-restoring division, and detailed bit manipulation.

ALU Top-Level Flowchart



1. ALU Datapath Diagram



2. Booth's Multiplication Algorithm (16×16)

Booth's algorithm is an efficient method for multiplying signed or unsigned integers in two's complement form. Instead of naively adding the multiplicand for each '1' bit in the multiplier, Booth's algorithm looks at pairs of bits (Q0, Q-1) of the multiplier and decides whether to add, subtract, or do nothing with the multiplicand. Key ideas:

- The multiplicand (M), multiplier (Q), and an extra bit Q-1 are kept in registers A, Q, and Q-1.
- At each step, only the pair (Q0, Q-1) is examined:
  - 0 1 → A = A + M (transition 0→1)
  - 1 0 → A = A − M (transition 1→0)
  - 0 0 or 1 1 → no change to A
- Then an arithmetic right shift is performed on the combined (A, Q, Q-1) register set.
- After 16 iterations (for 16-bit operands), the product is contained in the concatenation of A (high 16bits) and Q (low 16 bits).

In this project, Booth's algorithm is implemented in a separate function BOOTH. It supports both:

- Unsigned 16×16 multiplication (BOOTH_U, opcode 17)
- Signed 16×16 multiplication (BOOTH_S, opcode 18) using sign extension of the 16-bit operands.

3. Non-Restoring Division Algorithm (16×16)

Non-restoring division is an iterative algorithm for binary division that avoids repeatedly 'restoring' the remainder. It divides a 16-bit dividend by a 16-bit divisor to produce a 16-bit quotient and a remainder.
Key steps:

1. Initialize accumulator A = 0 and quotient Q = dividend.
2. Repeat 16 times:
    - Left shift the pair (A, Q) by one bit.
    - If the previous A was ≥ 0, compute A = A − divisor (M).
    - If the previous A was < 0, compute A = A + divisor (M).
    - If the new A is ≥ 0, set the least significant bit of Q to 1, otherwise set it to 0.
3. After all iterations, if A is negative, a final correction A = A + M is performed.

In this project, the NONREST function implements non-restoring division on 16-bit magnitudes. Two modes are supported:

- Unsigned 16×16 division (NREST_U, opcode 24)
- Signed 16×16 division (NREST_S, opcode 25), where the signs of dividend and divisor are handledseparately, magnitudes are divided, and then the sign of the quotient is applied at the end using two's complement if required.

4. Opcode Table

Arithmetic & Logic:

0 – ADD (R0 = R0 + R1)
1 – SUB (R0 = R0 − R1)
2 – AND (bitwise AND)
3 – OR (bitwise OR)
4 – XOR (bitwise XOR)
5 – NOT (bitwise NOT of R0)
6 – SHL (logical shift left)
7 – SHR (logical shift right)
8 – RSB (R0 = R1 − R0)
9 – MUL (hardware 32×32 multiply)
10 – UDIV (hardware unsigned divide)
11 – SDIV (hardware signed divide) 12 – NEG (two's complement negate)

Comparison & Bit Reversal:

13 – EQ (R0 = 1 if R0 == R1 else 0)
14 – GT (R0 = 1 if R0 > R1, signed)
15 – LT (R0 = 1 if R0 < R1, signed)
16 – RBIT (reverse all 32 bits)
21 – REV (reverse byte order in 32-bit word)

22 – REV16 (reverse bytes in each halfword) 23 – REVSH (reverse low halfword and sign-extend)

Advanced Multiply/Divide:

17 – BOOTH_U (Booth unsigned 16×16 multiplication)
18 – BOOTH_S (Booth signed 16×16 multiplication)
24 – NREST_U (non-restoring unsigned 16×16 division)
25 – NREST_S (non-restoring signed 16×16 division)

Rotate & Bit Operations:

19 – ROL (rotate left R0 by R1 bits)
20 – ROR (rotate right R0 by R1 bits)
26 – SET_A_BIT (set bit position R1 in R0)
27 – CLEAR_A_BIT (clear bit position R1 in R0)
28 – CHECK_A_BIT (R0 = 1 if bit R1 is set, else 0)
29 – TOGGLE_A_BIT (toggle bit R1 in R0)