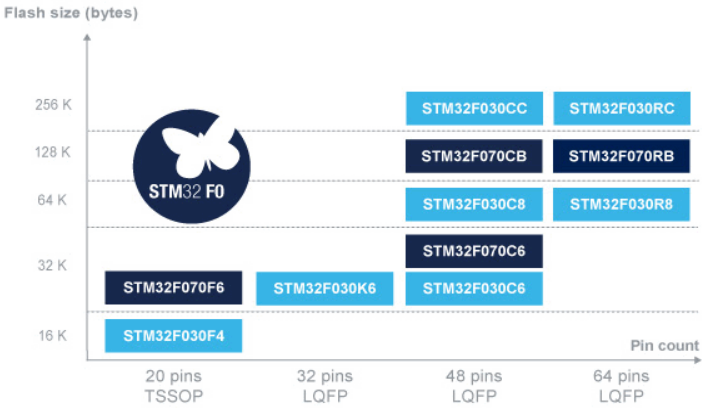
Computer Organization – Verilog – HW1

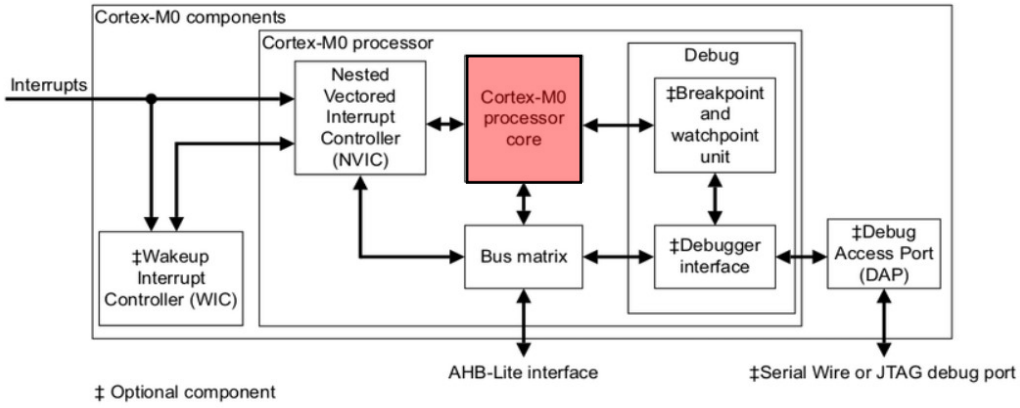
Email: iustCompOrg+4021@gmail.com

Welcome to the exciting world of Embedded Systems!

In the “Microprocessor's lab” course at our university, you'll have the opportunity to work with the renowned **STM32F0 microcontroller** family. These microcontrollers have gained great popularity and recognition within the industry.

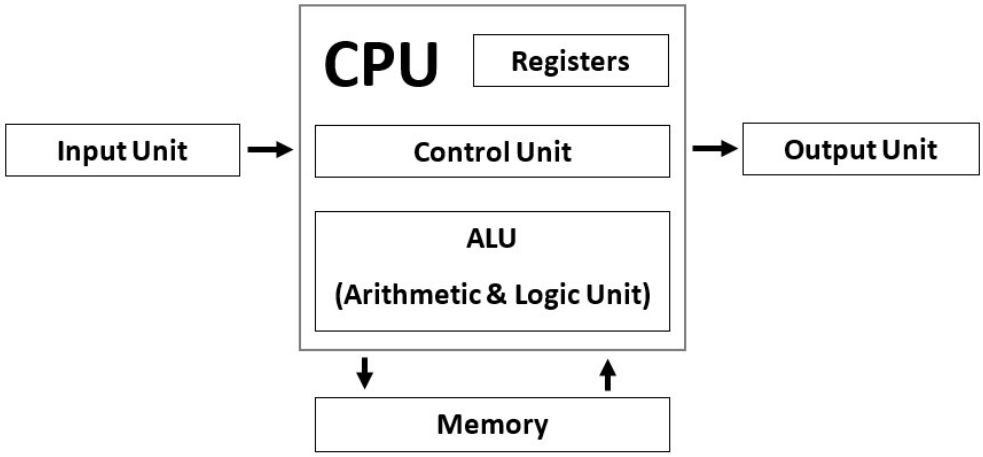


The STMF0 series microcontrollers are a family of microcontrollers that are based on the **ARM Cortex-M0** core. The Cortex-M0 core is the heart of these microcontrollers and plays a crucial role in their performance. The Cortex-M0 core is designed to provide a balance between simplicity and efficiency, making it ideal for embedded systems and low-power applications. It has a 32-bit architecture, which means it can process data and instructions in chunks of 32 bits at a time.



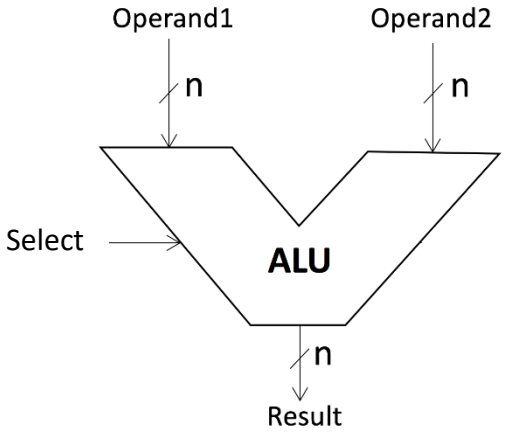
In “Computer Organization” course, we’re not focusing on programming microprocessors and microcontrollers; We are actually focusing on analysis and design of the processing units and the architecture of an embedded device.

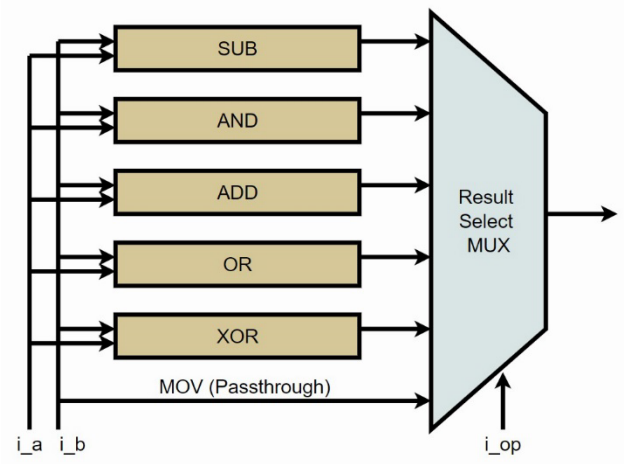
This block diagram is a small and basic figure of the CPU components:



ALU stands for Arithmetic Logic Unit. It is a fundamental component of a computer's central processing unit (CPU). The ALU performs arithmetic operations (such as addition and subtraction) and logical operations (such as AND, OR, and NOT) on binary data. It is responsible for executing the core mathematical and logical operations that support the processing capabilities of a computer.

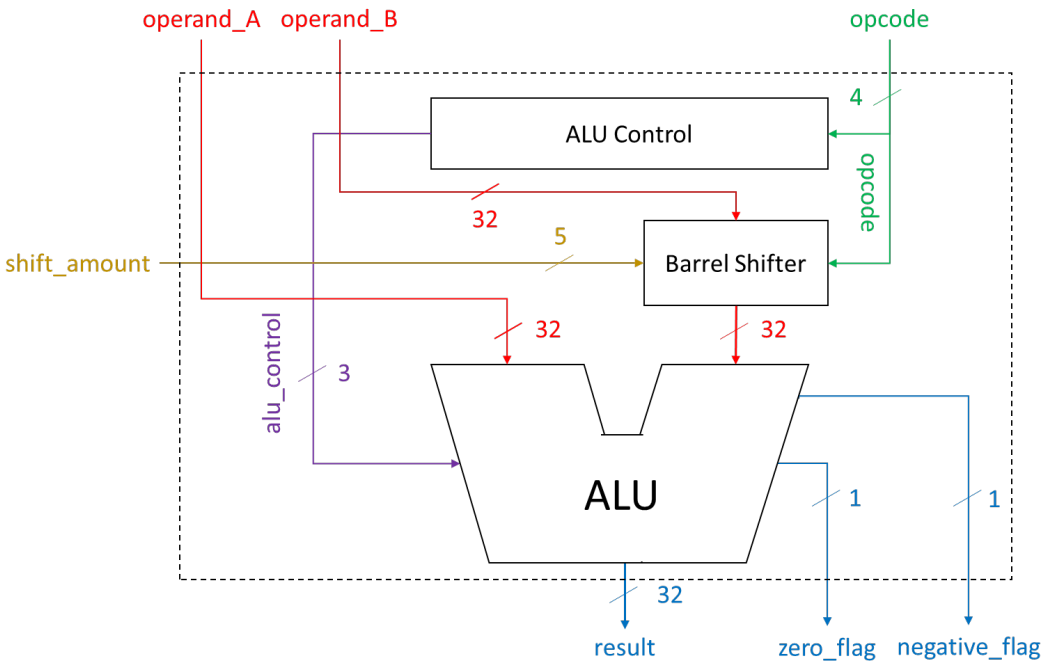
This is a picture of a sample ALU hardware structure:





In this assignment you are going to design the **Arithmetic Logic Unit (ALU)** of an ARM Cortex-M0 core in order to be used in an STM32F0 microcontroller.

This is a schematic of the execution unit which you are going to design for this ARM core:



This execution unit have 4 inputs and 3 outputs:

- `Input: opcode (4-bits)`
- `Input: operand_A (32-bits)`
- `Input: operand_B (32-bits)`
- `Input: shift_amount (5-bits)`
- `Output: result (32-bits)`
- `Output: zero_flag (1-bit)`
- `Output: negative_flag (1-bit)`

Here is a list of opcode field of ARM assembly instructions which your ALU must support and execute:

- `4'b0000: AND`
- `4'b0001: XOR`
- `4'b0010: SUB`
- `4'b0100: ADD`
- `4'b1000: ORR (Bitwise OR)`
- `4'b1010: LSL (Logical Shift Left)`
- `4'b1011: LSR (Logical Shift Right)`

Modules structure:

1. The `ALU_Control` module must receive the 4-bit `opcode` signal and create a 3-bit `alu_control` signal which goes directly to the ALU. The reason of this conversion is the shift operations (left/right) which will be taken place in the `Barrel_Shifter` module, not in the ALU. This means that for `LSR` and `LSL` opcodes, ALU have to pass `operand_B` directly to the `result` without any other operation.

2. The `Barrel_Shifter` is a digital circuit that can shift a data word by a specified number of bits without the use of any sequential logic, only pure combinational logic, i.e. it inherently provides a binary operation. You can implement it using an `always@(*)` block and Verilog shift operator(>> or <<) in a simple way. Our barrel shifter is going to check the `opcode` signal to determine if it is going to shift the input (`operand_B`) with the amount of `shift_amount` to left, right or no shift at all (for unrelated opcodes) and pass the result to the ALU.

3. The ALU is just a simple module (can be implemented with one Verilog `case` statement) which will take two 32-bit inputs and a 3-bit `alu_control` signal to switch the result within AND, ADD, SUB, OR and XOR operations, or just `pass operand_B` to `result` directly if the action is LSL or LSR. You should also assign two flag signals as outputs to show whether the result is equal to zero, or the result is negative. We shall call these signals `zero_flag` and `negative_flag`.

4. In the end, combination of this three modules will be making a top module which is going to be the execution unit of this ARM Cortex-M0 processor. You may need to design 4 modules (3 modules + 1 top module) for design of this processor, and write a testbench for the final (top) module to showcase your design. Any additional features (according to ARM standard ISA) will be appreciated and highly rewarded!

Notes:

Send all your assignment related files (*modules* and *Testbench* [.v] and [.vvp] and [.vcd] files) along with a detailed report in [.pdf] format all in one zip file to the email address of the class.

For this assignment you can work in groups of two or three, each participant must submit the assignment individually with their respective name and student number.

If you have any questions regarding this assignment, feel free to contact us.

Please submit your homework, simulations and projects in the following format:

Name_StudentNumber_Verilog_HW1 (BillGates_12345678_Verilog_HW1)

Good Luck!