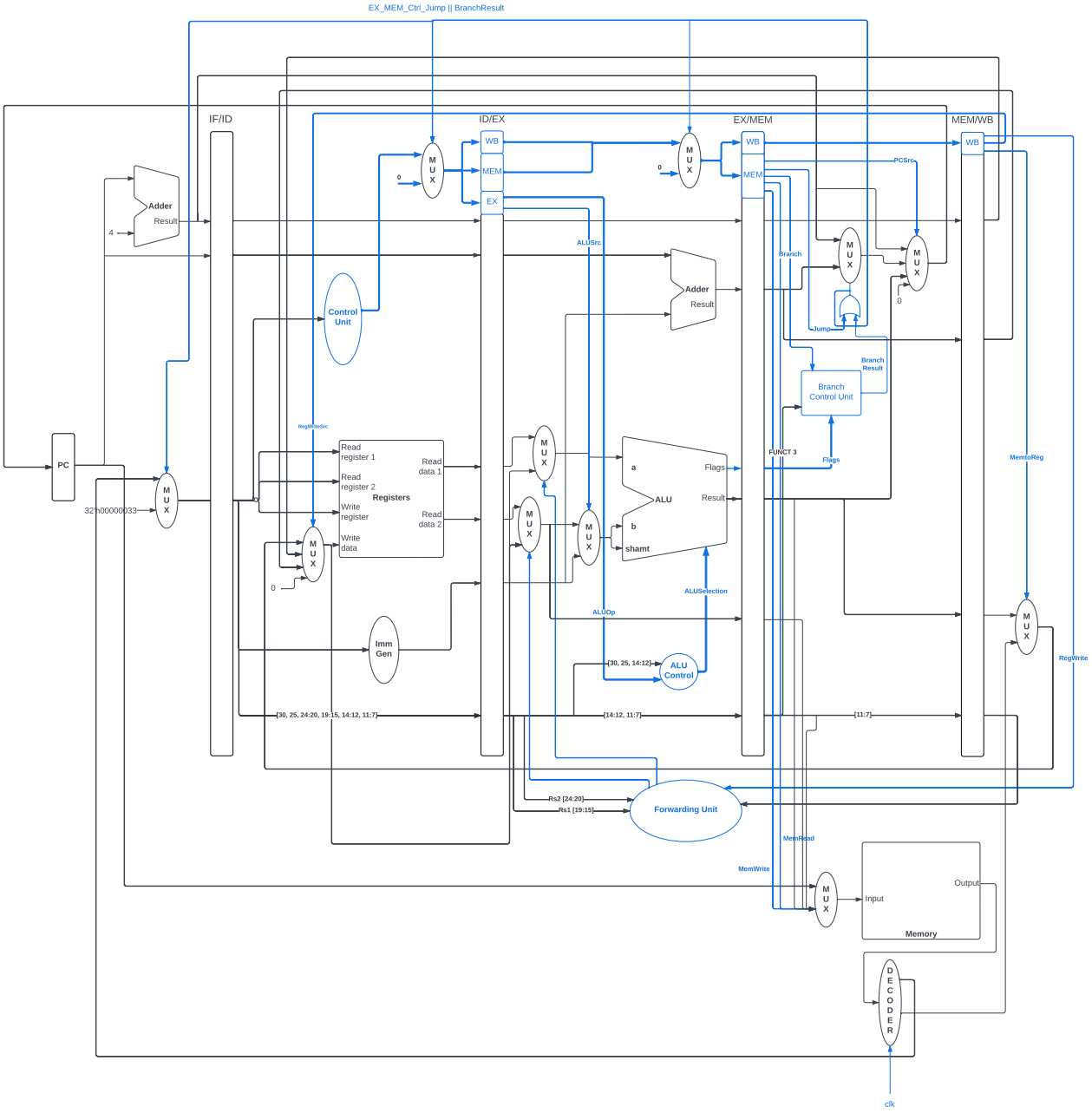
# RISCV Processor:
# Pipelined Implementation

Sara Mohamed 900203032
Mohamed Noureldin 900203758

## Introduction

For this milestone, we implemented the pipelined RISC-V processor that supports the full 40 RV32I instructions and the RV32M instructions with the correct handling of hazards. The implemented processor features a single ported byte-addressable memory and works successfully on the FPGA board.

### Diagram




The above block diagram shows our pipelined datapath, including control units and control signals.Due to

the complexity of the design and the presence of many control wires, the load signals of registers were not included; for simplicity and readability.

### Additions and Explanations

The primary differences between our single-cycle implementation and the pipelined implementation are the following:

- The addition of the IF/ID, ID/EX, EX/MEM and MEM/WB registers to pipeline the instructions by loading and storing values inside these registers.
- The replacement of the instruction memory and data memory with a single memory for both, which resulted in an addition of a multiplexer to select whether to load an instruction or data and the addition of a decoder to select whether to output the data to the IF stage or to the MEM stage.
- The addition of three multiplexers (one before the IF/ID register, one at the control signals of the ID/EX register and one at the control signals of the EX/MEM register) to flush the instructions in the pipeline when needed.
- The addition of the forwarding unit to detect the need to forward an updated content of registers from the MEM stage to the EX stage which resulted in the addition of two multiplexers (one for each input of the ALU) to select whether to compute using the values from the register file or the before previous instruction result.
- The replacement of the PCLoad control signal with its negation PCNotLoad in order to have a default value of 0 at reset to make sure that the PC will load initially.
- The update of the ALU and ALUCU (addition of instruction bit 25 as input) to support the RV32M instructions (not shown in the diagram since these are internal edits)

# Implementation

To implement the pipelined RISCV processor, we needed to implement the following Verilog descriptions (modules):

- **Some Simple Dependencies**:
  - ***FullAdder.v***: Simple full adder module that adds two one-bit numbers and an optional input carry and outputs their sum and carry out signals.
  - ***RCA.v***: Simple ripple carry adder with a parameter $N$ of default value 8 for the size of the two numbers to be added. It adds two numbers and an optional input carry and outputs their sum and carry out signals.
  - ***defines.v***: Verilog file with processor constants and definitions to be used in other files.
  - ***Nbit_2x1mux.v***: A 2x1 multiplexer of default input size 8, with a parameter $N$ to change size of inputs. It takes the two inputs to select between, a one-bit select line, and outputs the selected value.
  - ***Nbit_4x1mux.v***: A 2x1 multiplexer of default input size 8, with a parameter $N$ to change size of inputs. It takes the two inputs to select between, a two-bit select line, and outputs the selected value.
  - ***Nbit_reg.v***: Simple register of default size 8, with a parameter $N$ to change size of register. It takes the one-bit signals clock, reset, load, and shift enable (to shift the content of the register by one bit to the left), as well as the input data, and outputs the output data of the register.
  - ***Nbit_1x2Decoder***: A 1x2 decoder of default input size 32, with a parameter $N$ to change size of inputs. It takes an input and a selection line and outputs the input data on one of the output wires according to the selection line.
  - ***BCD.v***: A module to convert a 13-bit number to binary-coded digits. It takes as input the 13-bit number and outputs the binary-coded digits of the thousands, hundreds, tens and ones of the input number.
- **ALUCU.v**: Control unit for the Arithmetic logic unit (ALU). This module takes as input the ALUOp, funct3 and a part of funct 7 and **bit 25** to determine the ALU operation to be executed.
- **BranchCU.v**: Control unit for the branch operations to determine whether the processor should branch to a new instruction or not. This module takes as input the funct3 (branch type) of the instruction, ALU output flags and the Branch control signal to output a bit, indicating whether to branch or not.

- **CU.v**: Processor's main control unit. This module takes as input the opcode of the instruction (neglecting the first 2 bits) and instruction bit 20. The module then outputs the Branch, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite, PCNotLoad, Jump, ALUOp, RegWriteSrc and PCSrc control signals.
- **SingleMem.v**: Little Endian Byte Addressable Read-Write data memory that outputs the content of memory as words of size 32 bits. The module takes as input a clock, MemoryRead control signal (MemRead), MemoryWrite control signal (MemWrite), target address and function3 of the current instruction. The model then outputs the read data at the data_out signal. The mode of the memory can be selected using the MemRead and MemWrite control signals.
- **RegFile.v**: Three ports registers file for the processor holding the 32 main registers of parameterized size N (defaulting to 32 bits). The module takes as input the two read register numbers, the write target register number, the data to be written, the RegWrite control signal, a Reset signal and a Clock. The module then outputs the two read data.
- **rv32_ImmGen.v**: Immediate generator that takes as input the whole instruction word and outputs the correct 32-bit ordered immediate.
- **CPU.v**: Top model for the whole processor. This module holds and connects the whole modules to implement the whole schematic provided in the document. This module takes as input clock and reset signals and has no output signals.
- **ForwardingUnit.v**: Forwarding unit that detects the need to forward contents of a register from the MEM stage. The model takes as input the source registers of the currently being executed instruction (ID/EX Source register1 and ID/EX Source register 2), the destination register of the before previous instruction (MEM/WB Destination register) and the RegWrite control signal from the MEM/WB register. The model outputs two selection lines (forwardA and forwardB) which are used in two muxes to determine what to be provided as input to the ALU or memory.
- **Four_Digit_Seven_Segment_Driver.v**: The driver module for the seven-segment display in the FPGA board. This module takes as input a clock and a 13 bit number to be displayed. It outputs an Anode to determine which of the four digits to be on or off, and the seven segment on/off signals.
- **FPGATop_const.xdc**: Constraint file for the implementation of the processor on the FPGA board. The module connects the FPGATop module input and output ports with switches, buttons, LEDs and seven segment displays on the basys board.
- **FPGATop.v**: Top model for the whole processor to be implemented on the FPGA board. This module holds and connects the whole modules to implement the whole schematic provided in the document just like the **CPU.v** module but adds the additional logic of receiving input and outputting on the FPGA board. The module takes as input a clock for the processor, a clock for the seven-segment displays, a reset signal, a selection for what should be output on the less, and a selection for what should be displayed on the seven-segment displays. It outputs the signals for the 16 LEDs and the seven-segment displays.

# Testing

To make sure that the processor was implemented correctly, we created and used several test cases to make sure that all the forty instructions and the bonus RV32M instruction were being properly executed. Mainly the following seven test files were used:

- **test_Branch**: File tests the eight branch instructions (BEQ, BNE, BLT, BGE, BLTU and BGEU). The file first loads values 17 and 9 in registers x1 and x2 respectively. The code then keeps checking on both taking and not taking branches to make sure that all branch instructions are working as expected.
- **test_fencebreakcall**: File tests the EBREAK, ECALL, and FENCE instructions. (We commented out each instruction every iteration of our running, so we could check that all three worked).
- **test_I-Arithmetic**: File tests all the arithmetic immediate instructions (ADDI, ANDI, ORI, XORI, SLLI, SLTI, SLTIU, SRLI, and SRAI) instructions. The file first loads values 17 and 9 in registers x1 and x2 respectively. Then, it performs all the operations on the two registers and saves the result in x3.
- **test_JumpAUIPC**: File tests the jump instructions (JAL and JALR) along with AUIPC. The file first add the upper of 365 to the current PC and stores the value in the register x1. Two jumps (JALR and JAL) are then taken consecutively while storing the values of PC in registers x3 and x5.

- **test_LoadStore**: File tests the load and store instructions (LW, LB, LBU, LH, LHU, SW, SH, SB). We first initialize the data memory to contain some values. The file first tests the load instruction by using every instruction and loading values into registers. Then, the file tests the store instructions by using them to store values into the memory.
- **test_LUI**: File tests the LUI instruction by loading the upper of the immediate 365 in the register x20.
- **test_Rformat**: File tests the ten R-type instructions (ADD, SUB, SLL, SLT, SLTU, XOR, SRL, SRA, OR and AND). The test begins by loading the values 17 and 9 in registers x1 and x2 respectively. Each one of the ten instructions is then carried on x1 and x2 and its result is stored in the register x3. All values are stored in the x3 register to make allow easy debugging and the need to only check one register for simplicity.
- **test_MultDivRem**: File tests the multiplication, division and remainder instructions by putting the value 10 in the x1 register and the value -3 in the x2 register then test the different instructions and makes sure that the results are correct.
- **test_Fib**: File tests the logic of the processor by implementing the Fibonacci algorithm using the recursive method. This program computes the value of Fibonacci(5) and stores it in the a0 register.

Along with the test files, a testbench **CPU_tb.v** is provided in the *"Verilog"* folder. This testbench simply creates a clock with a period of 2ns, resets the processor and then keeps it running for 1000ns.
To use any of the test files provided, the **SingleMem.v** file should be edited, and the absolute path of one of the test files in the *"hex_split"* should be provided inside the initial statement.

Please note this processor was implemented as a byte-addressable, little-endian memory-based processor. To ensure this was maintained, we converted our test assembly code into machine code, and used a script we wrote in C++ to split the code up and reverse it bit-wise. For example, if the code had two instructions,

```
32'habcd1234
32'h56789eff,
```

it would be represented in the input file for the instruction memory as

```
34
12
cd
ab
ff
9e
78
56
```

Each one of the test files mentioned is available in different formats in three different directories. The actual RISC-V text code is available in the *"txt"* directory. The equivalent hex machine code is stored as 32-bit instructions in the *"hex"* directory. The split hex machine code (one byte per line) is stored in the *"hex_split"* directory so that it can be directly provided to the Instruction memory. A hazard-less version of the test program (separates instructions by three nops) also exists in the *"hex_split_nops"* directory

# Bonus Features

The two bonus features we chose to implement were the **RV32-M** instruction set for the multiplication, division, and remainder operations; as well as implementing the processor on the Nexys A7 **FPGA board**. For the first feature, we added the aforementioned additions in the ALU, and the ALUCU. For the second bonus, we created the new module ***FPGATop.v***.

Implementation on the FPGA board is detailed in the video attached in the zip folder for this submission showing the R-for.