DLX-PRO

PROJECT

RICCARDO CUCCU

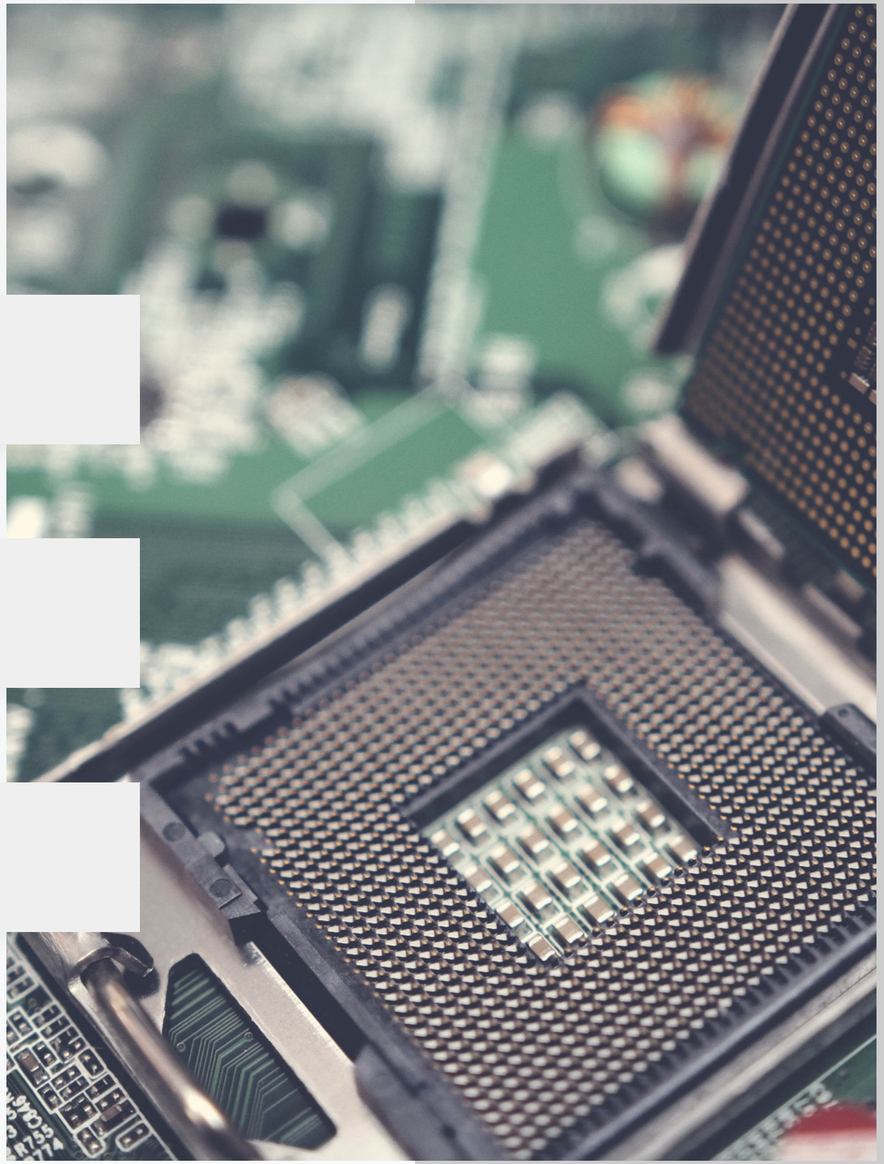
CONTENT

OVERVIEW

32-bit RISC Architecture

5-Stage In-Order Pipeline

Hardwired Control Unit



PRO FEATURES

EXTENDED ISA

Addition of **25 instructions**, making a total of 52.

RAW HAZARD MANAGEMENT

Addressing of **RAW data hazards** through the implementation of a **Forwarding Unit**.

OPTIMIZED ALU

Integration of a **Pentium 4 Adder** and a **Booth Multiplier**.

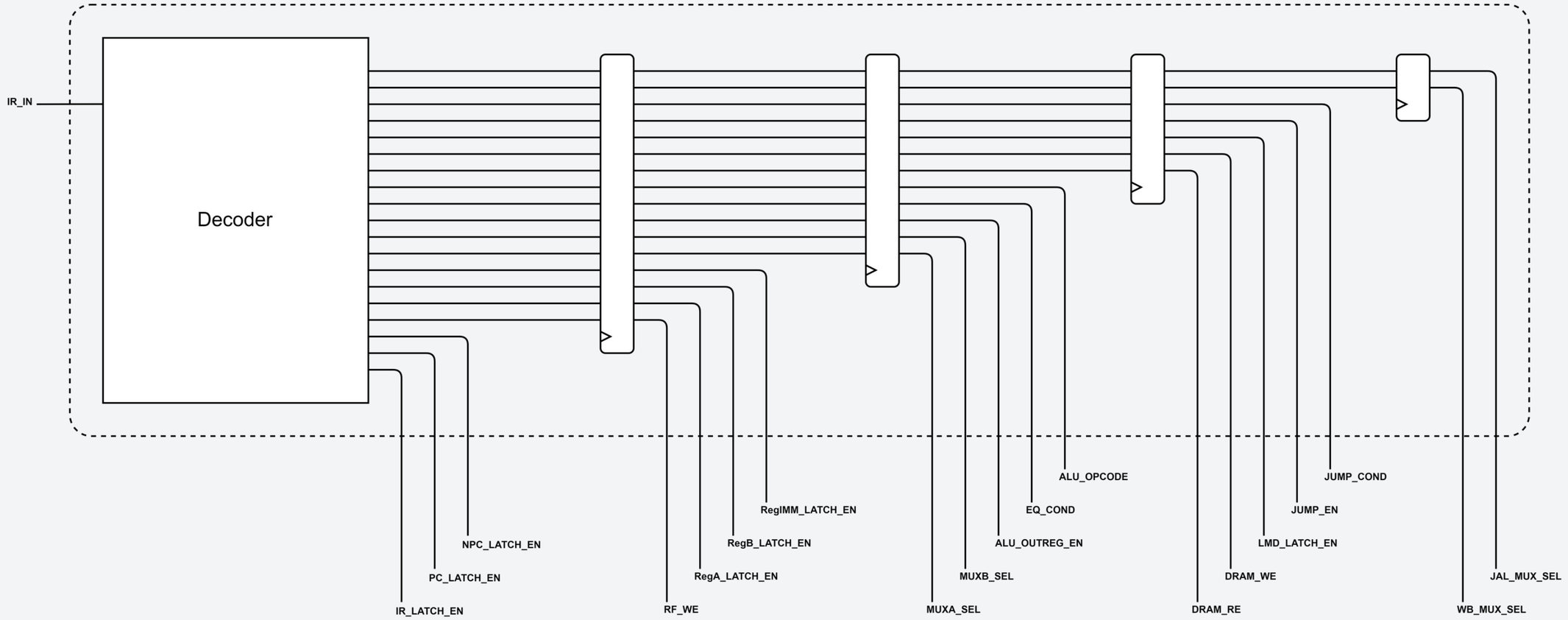
PARAMETRIC DESIGN

Complete parametric design manageable through a single file.

SEMI-AUTOMATED WORKFLOW

Utilization of scripts for **partial automation** in simulation, synthesis and place-and-route stages.

CONTROL UNIT



DETAILS

CONTROL UNIT ROLE

The CU orchestrates the **flow of control signals** throughout the pipeline, responding to the **instruction** from Instruction Memory (IRAM), determined by the current value of the Program Counter.

DESIGN APPROACH

A **hardwired** methodology, utilizing **two Look-Up Tables**, was employed for reliability and simplicity, generating an 18-bit **Control Word** signal and the **ALU operational code**.

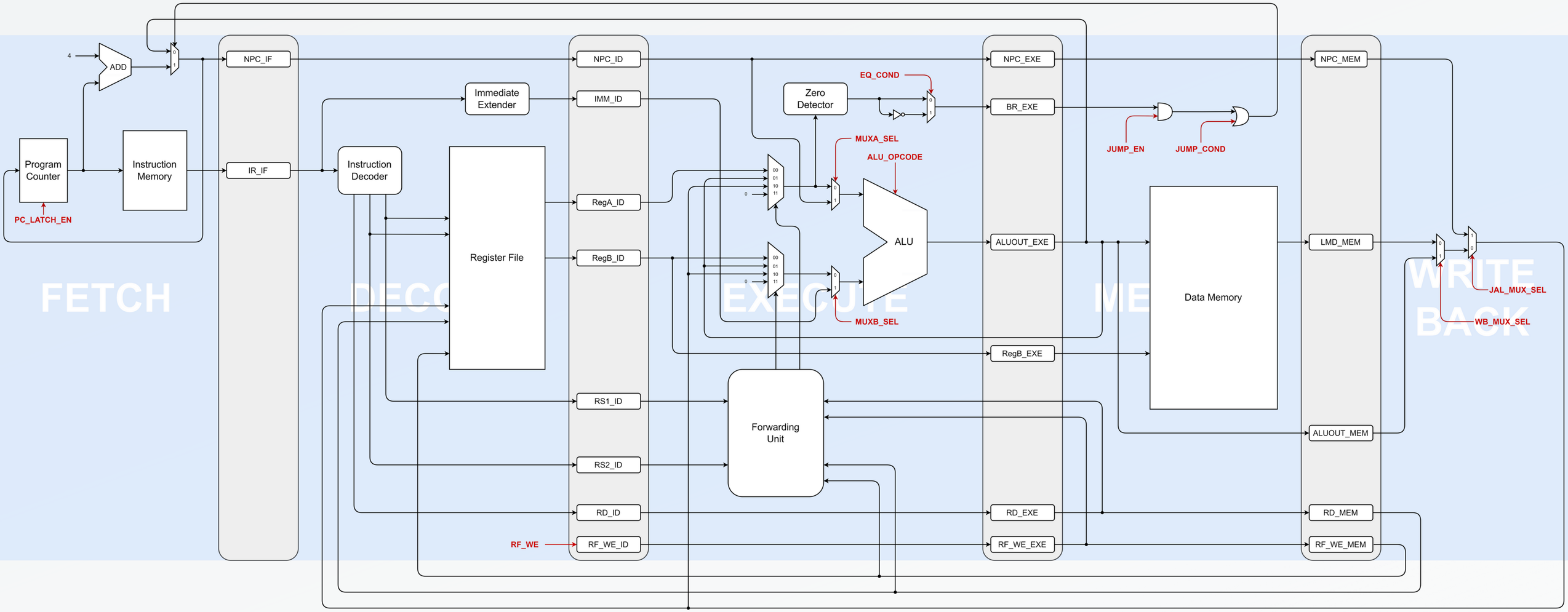
PIPELINE PROCESSING

The pipeline accepts a **new instruction** each clock cycle, **shifting** Control Word signals and the ALU operational code by one stage position to maintain **synchrony** with **pipeline stages**.

LIMITATIONS

The absence of **stall** or **branch prediction** necessitates **three NOP instructions** or **compiler modifications** for accurate execution during jump or branch operations.

DATAPATH


4
ADD
0
1
NPC_IF
NPC_ID
NPC_EXE
NPC_MEM
Immediate Extender
IMM_ID
Zero Detector
EQ_COND
BR_EXE
MUXA_SEL
ALU_OPCODE
JUMP_EN
JUMP_COND
Program Counter
Instruction Memory
PC_LATCH_EN
IR_IF
Instruction Decoder
Register File
RegA_ID
RegB_ID
00
01
10
11
ALU
ALUOUT_EXE
LMD_MEM
Data Memory
MUXB_SEL
JAL_MUX_SEL
WB_MUX_SEL
RegB_EXE
RS1_ID
RS2_ID
Forwarding Unit
ALUOUT_MEM
RD_ID
RD_EXE
RD_MEM
RF_WE
RF_WE_ID
RF_WE_EXE
RF_WE_MEM
FETCH
DECO
EXECUTE
ME
WRITE BACK

STAGES

FETCH

Retrieves instructions from Instruction RAM (IRAM).

DECODE

Extracts register values associated with specified addresses and extends the immediate value.

EXECUTE

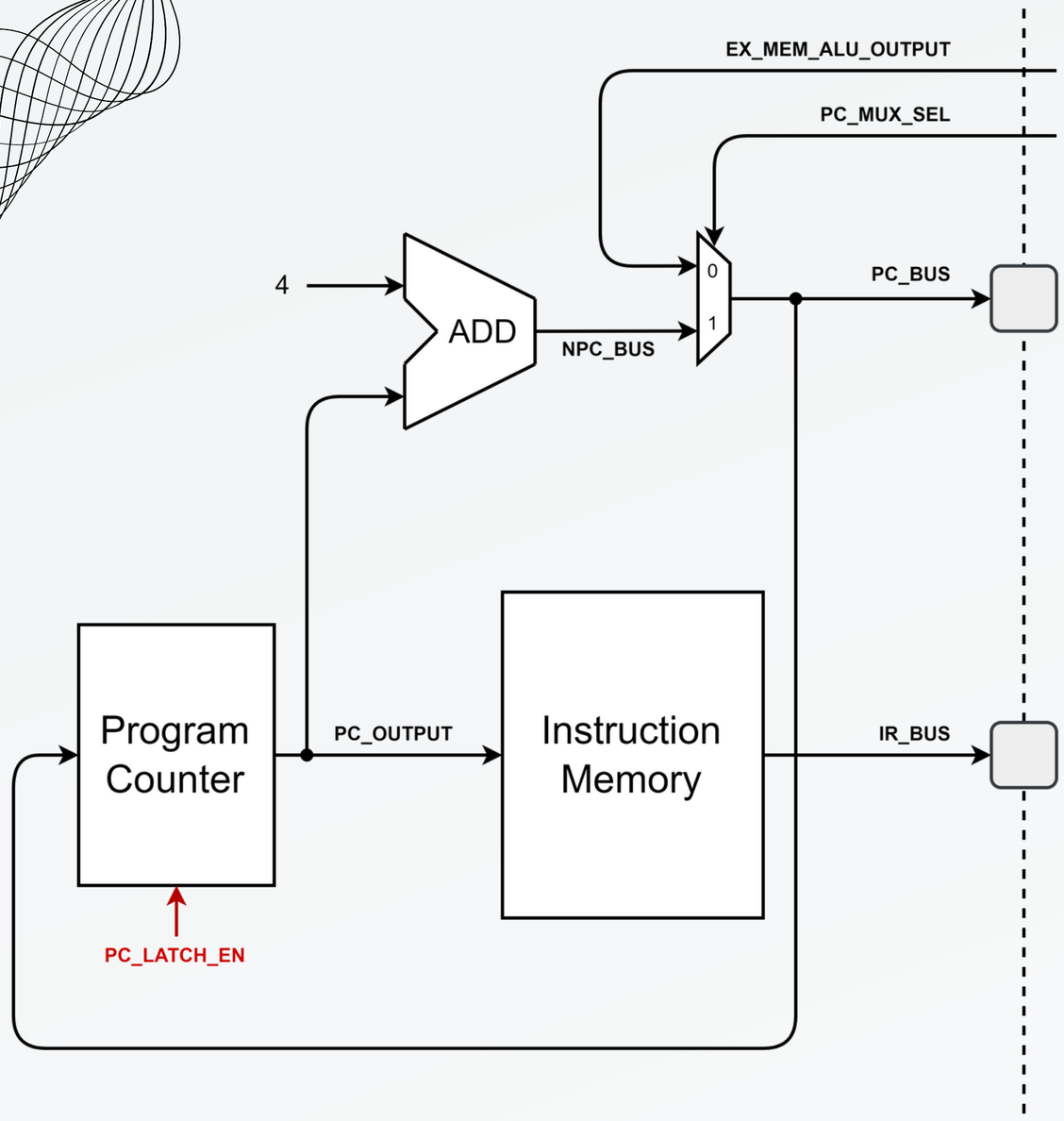
Performs arithmetic, logical, comparison and shift operations.

MEMORY

Manages read or write actions to Data RAM (DRAM).

WRITE BACK

Route the final result generated by the datapath.

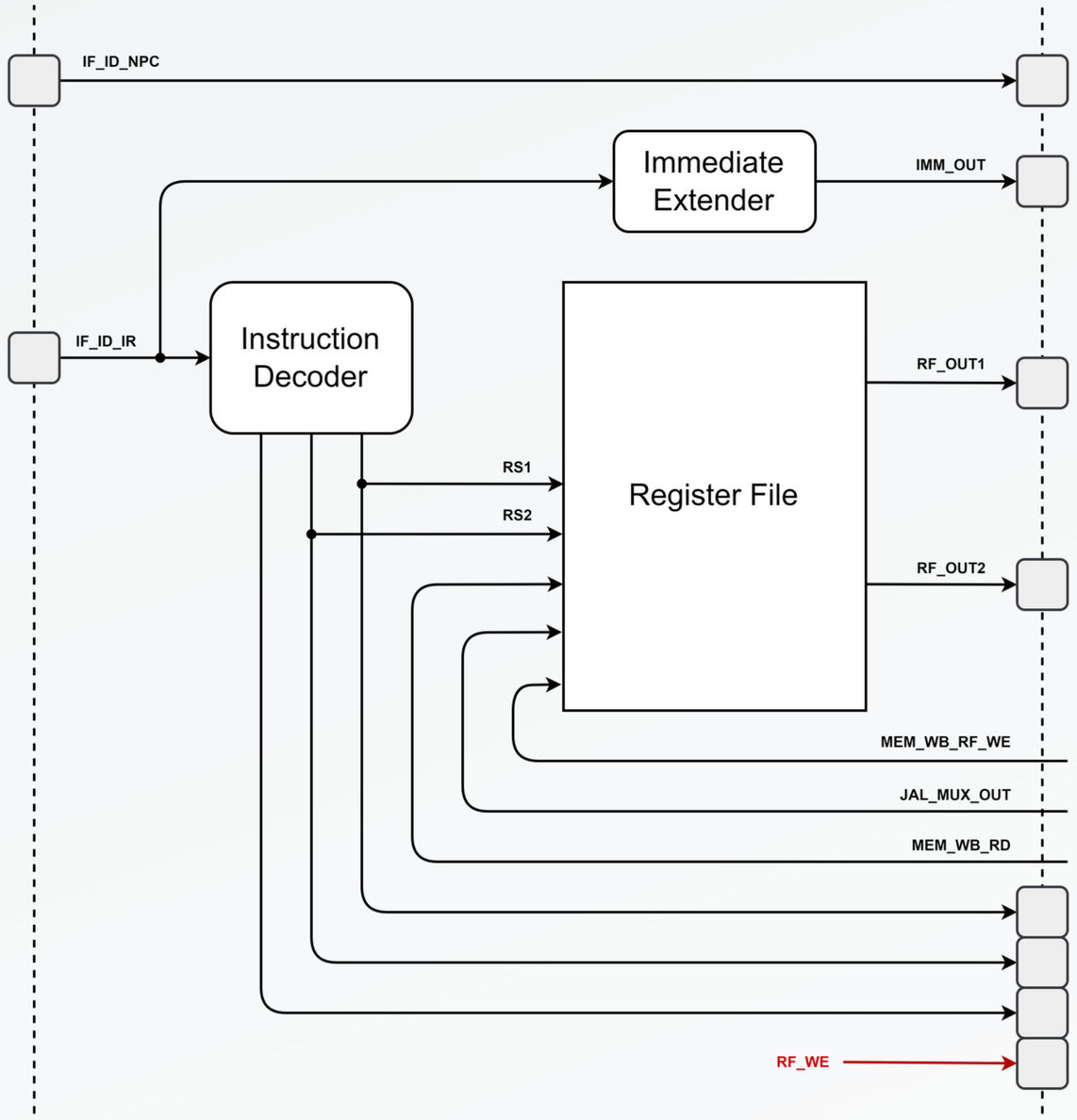


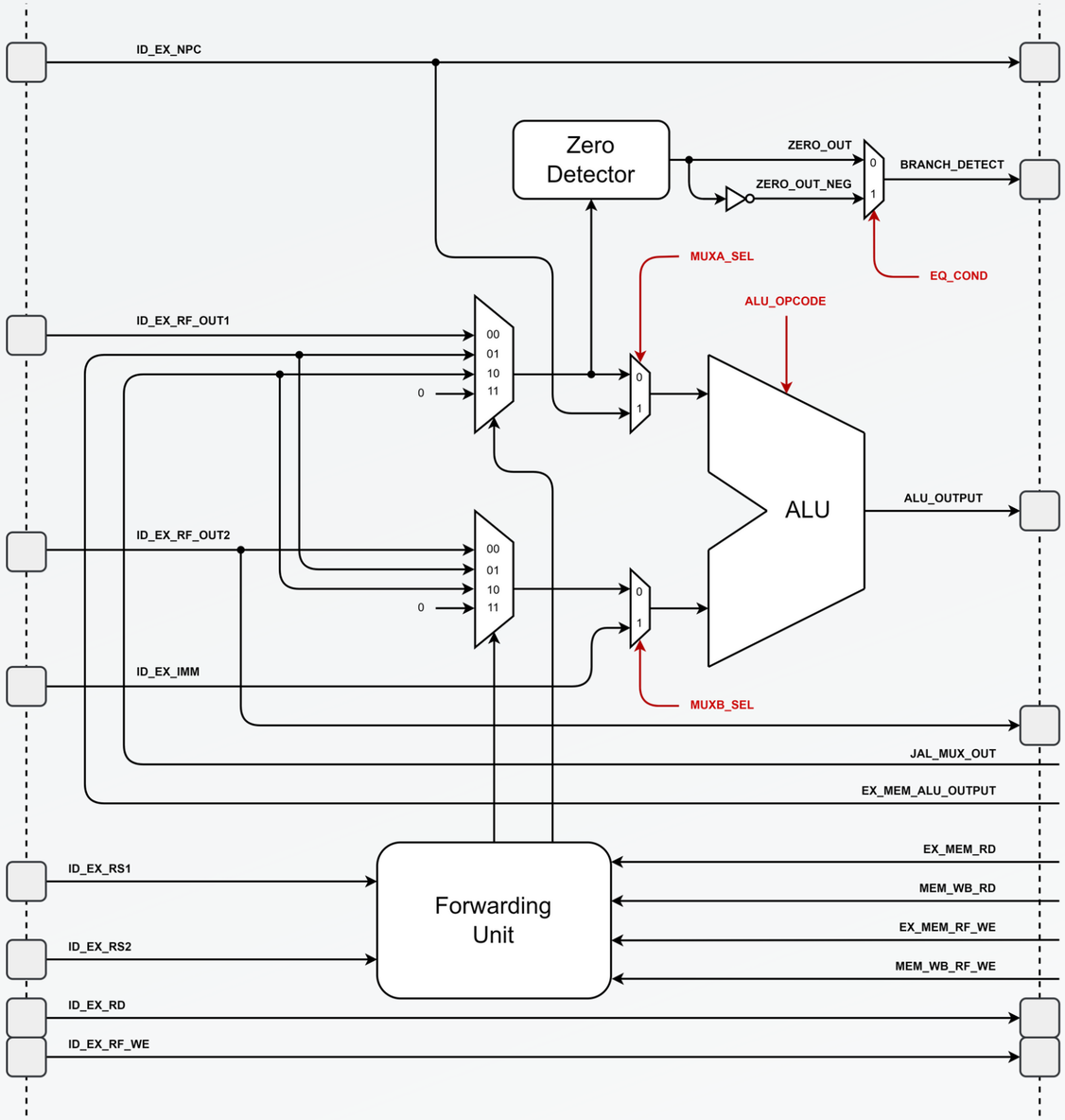
FETCH

- The next instruction is retrieved from the **Instruction Memory** (IRAM) using the address in the **Program Counter** (PC).
- The retrieved instruction is stored in the **Instruction Register** (IR).
- In sequential execution, **the PC is incremented by 4** to align with the 4-byte instruction encoding standard.
- During control flow instructions, the **multiplexer** is activated, directing the **output of the ALU** from the Memory stage to both the Program Counter (PC) and the Next Program Counter (NPC) Register.

DECODE

- The **instruction** is interpreted, and **operands** along with **immediate value** are extracted for utilization in the subsequent stages.
- The **Instruction Decoder** decomposes the instruction to **extract the operands** that will act as register addresses.
- The **Immediate Extender** performs **sign-extension** on immediate values.
- The **Register File** stores the processor's registers, ensuring the values are accessible in the subsequent stages.





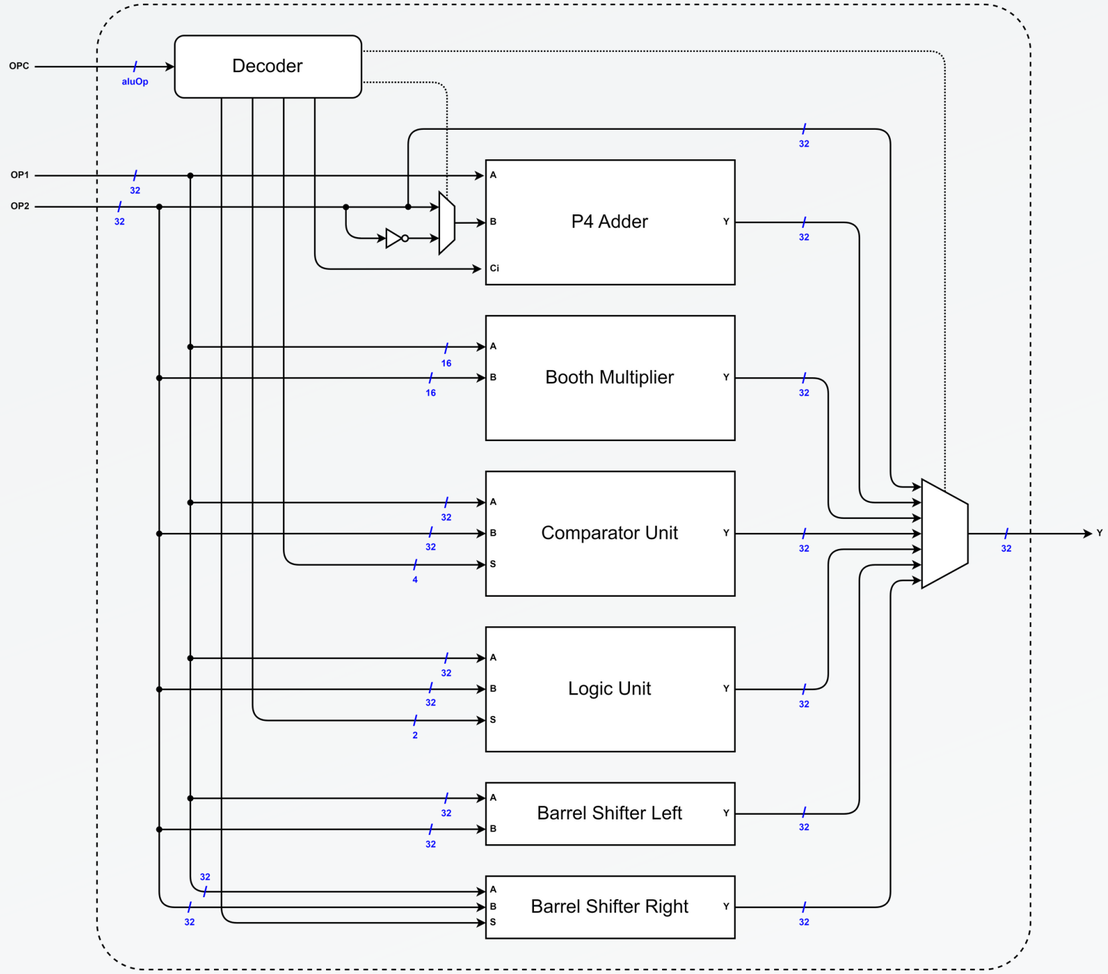
EXECUTE

- **Routes operands** to the **Arithmetic Logic Unit** (ALU) for computational tasks, encompassing arithmetic and bitwise logical operations.

- Utilizes a **tree structure of multiplexers** in conjunction with the **Forwarding Unit** to manage operand delivery from subsequent pipeline stages, avoiding **read-after-write (RAW) hazards** when results are yet to be written back to the Register File.

- Incorporates a specialized **Zero Detector** module to ascertain if the first operand is zero, priming the pipeline for potential **conditional branches**.

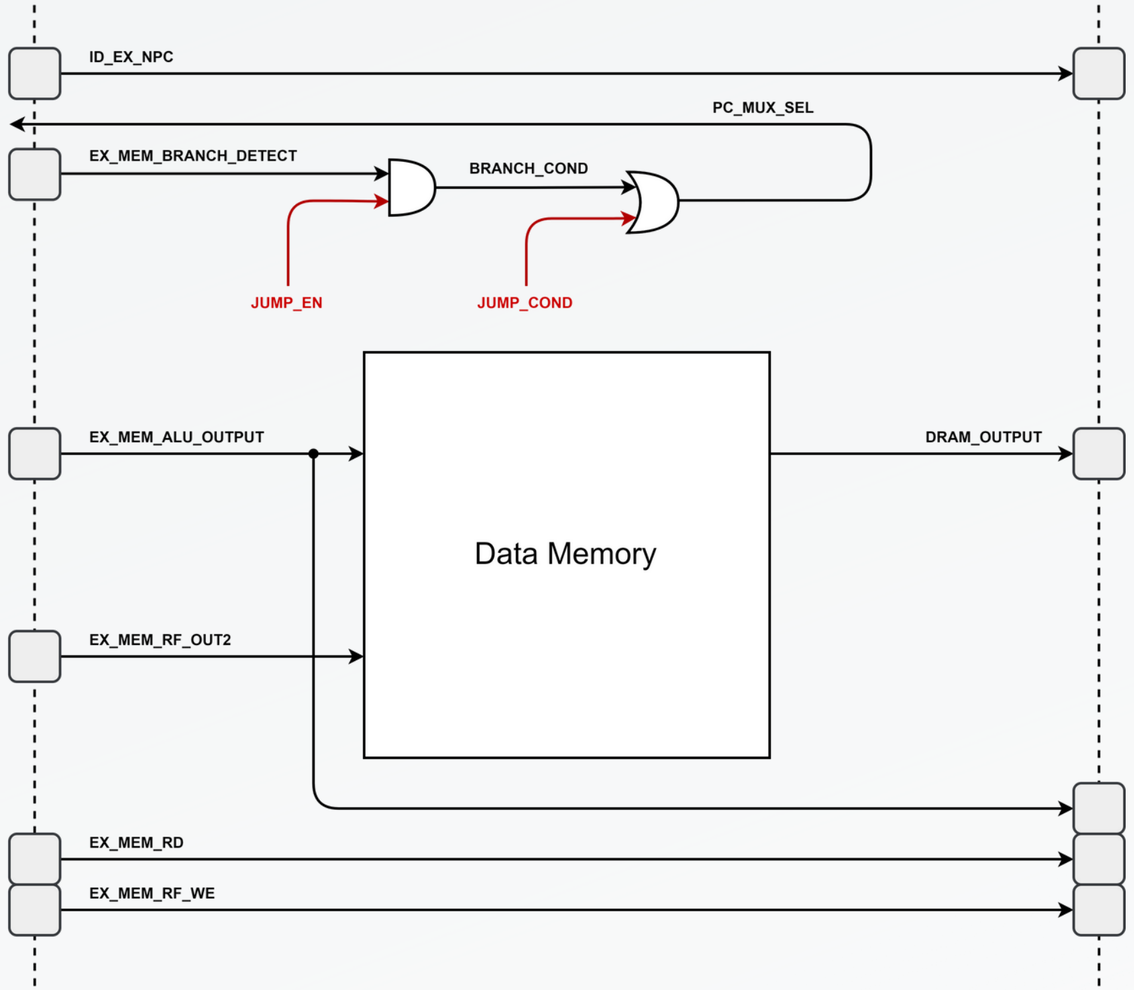
ALU DETAILS



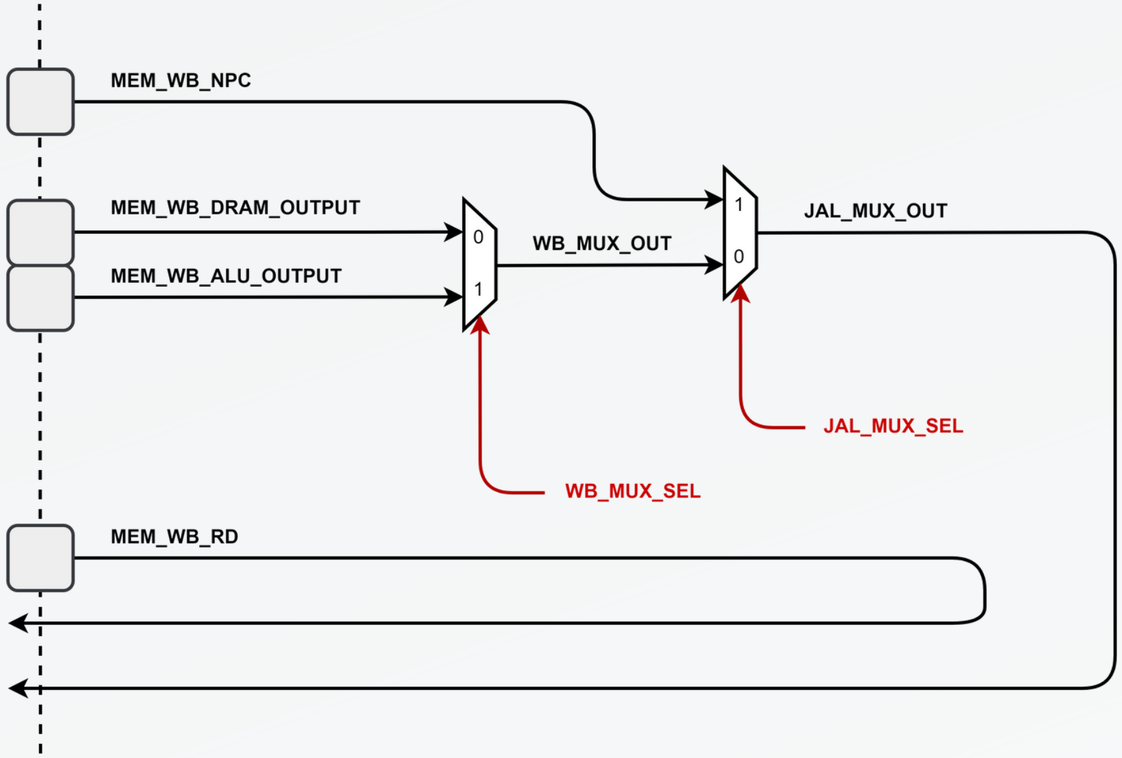
- Manages a broad spectrum of operations, encompassing **logical** and both **signed** and **unsigned arithmetic**, courtesy of multiple functional components:
  - **Pentium 4 Adder** module for addition and subtraction.
  - **Radix-4 Booth Multiplier** for multiplication tasks.
  - Specialized **Comparator Unit** for comparing the two operands.
  - Dedicated **Logic Unit** for bitwise AND, OR and XOR operations.
  - Two distinct **Barrel Shifters** for left and right shifts.

MEMORY

- Manages data primarily through **load** and **store operations**.
- Utilizes an **AND logic gate** in conjunction with the Control Unit signal to manage **branch conditions** during BEQZ or BNEZ instructions.
- Employs an **OR logic gate** to direct the input of the Program Counter during **branches** or **jumps**.
- The **Data Memory** (DRAM) serves as a storage hub for data, allowing both **read** and **write operations**.



WRITE BACK



- Simplest stage within the architecture, utilizing a **two-level multiplexer tree** to deliver the required value to the rest of the system.
- The first-level multiplexer chooses between the **outputs of the ALU** and the **DRAM**.
- The second-level multiplexer selects between the output of the preceding multiplexer and the **value of the Next Program Counter**.
- The write-enable signal is routed to the **Register File**, allowing the acceptance of the new value.

FURTHER IMPLEMENTATIONS

Compared to the existing architecture, various functions can be implemented to enhance its performance and broaden its range of applications.

Implementing the **‘Always Taken’** technique is a simple way to enhance throughput and reduce pipeline stalls with minimal hardware adjustments.

BRANCH PREDICTION

Some hardware modifications to the execution model can optimize **data hazard handling** and facilitate parallel instruction execution.

O-O-O EXECUTION

Implementing **floating-point operations** would extend the architecture's applicability to more high-precision computational tasks.

FP OPERATIONS

Enhancing the current ISA can meet **specialized needs** but requires modifications to strengthen processor architectures.

ISA EXTENSIONS

Improvements can establish a more integrated and robust **automation framework**, enhancing reliability and design verification.

SCRIPT ENHANCEMENT

And s
and so
so on,
on, and
and so
so on,
o

AND

GROUP 32

Riccardo Cuccu

s253986

THANK'S FOR YOUR ATTENTION