

# **DLX Project**

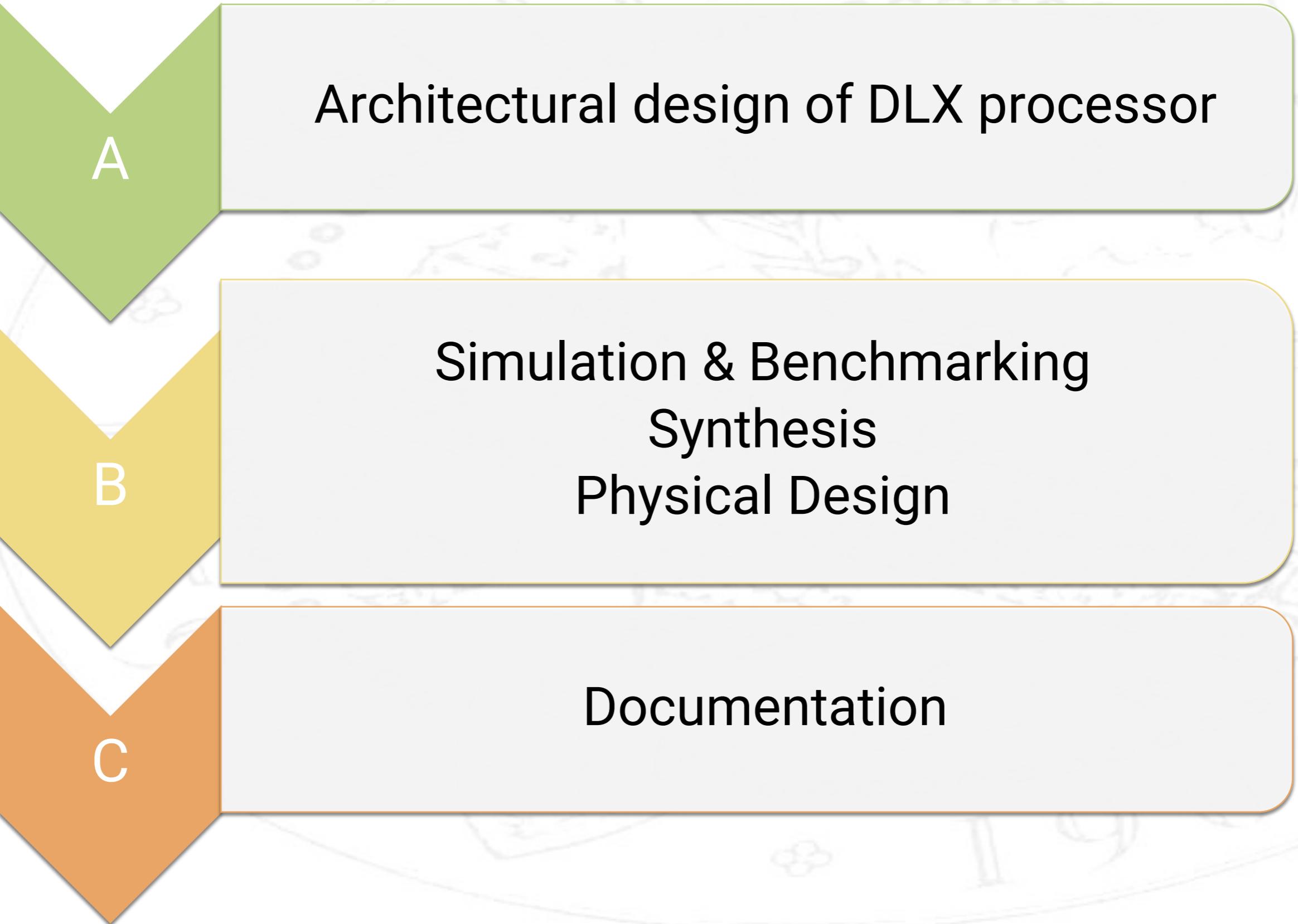
**Microelectronic Systems Course**



**POLITECNICO  
DI TORINO**

# DLX Processor

- The DLX (aka. "**DELUXE**") is a processor architecture introduced by John L. Hennessy and David A.
- This architecture is a **simpler version of the MIPS**
- The MIPS core belongs to the **RISC** family (Reduced Instruction Set Computer).
- This kind of architecture uses simpler **fixed-size instructions** and addressing modes compared to its main competitor, the **CISC** (Complex Instruction Set Computer).
- In this way decoding the instructions is **simpler**, but the size of the **code increases**.
- The RISC approach became very popular, especially in the **embedded market**, and it is still today.



## Possible versions:

DLX basic (max. project evaluation 28/30)

DLX pro (max. project evaluation 30L/30)

## The submission requires:

All VHDL files (with testbenches)

All the scripts required for simulation, synthesis, etc...

Full documentation

## Discussion deadlines:

Suggested: July 31<sup>st</sup>

Max: October 30<sup>th</sup>

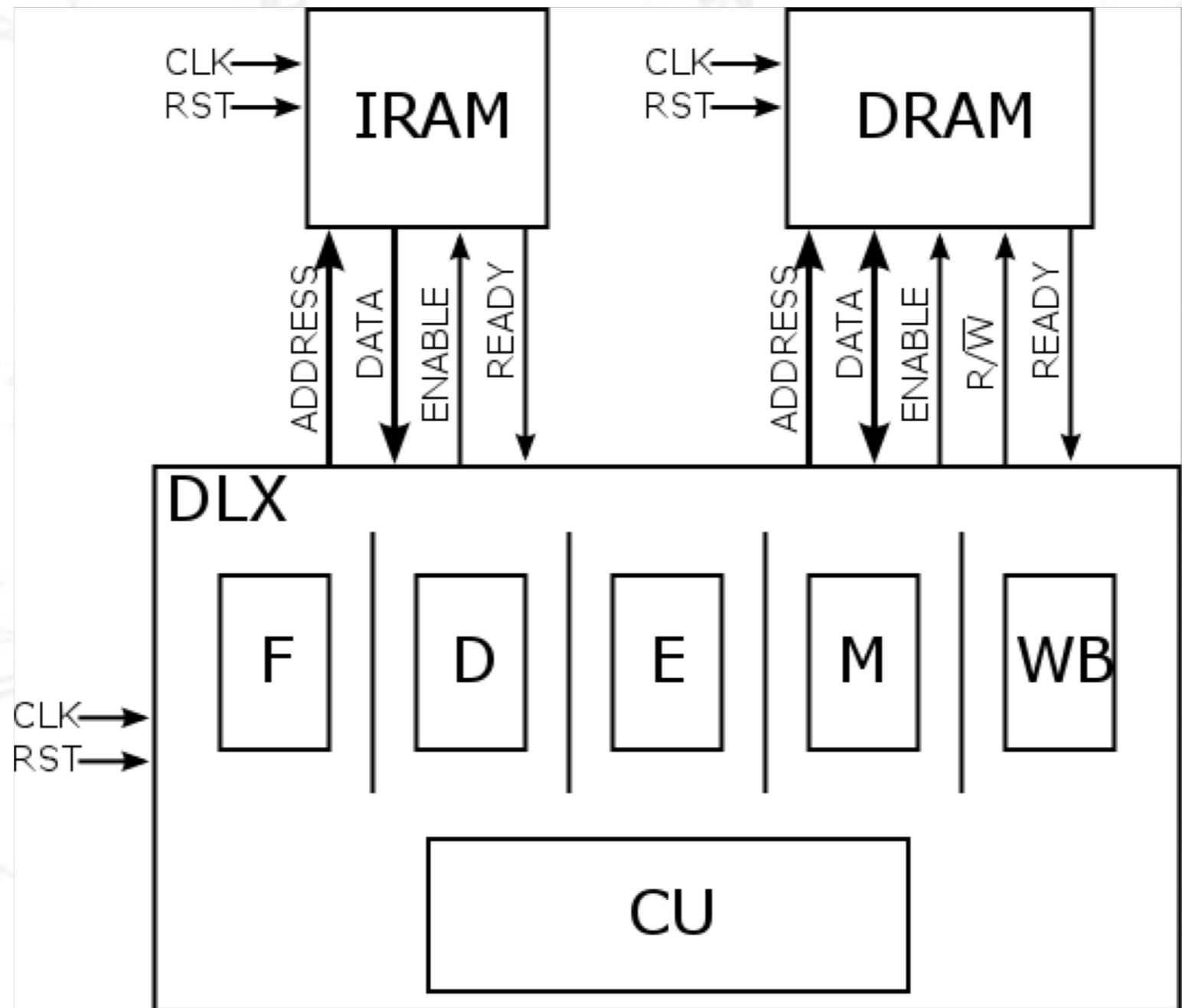
All groups are required to submit all files one week before the discussion

Remember to add comments within all files!!!!

# Discussion sessions

1. End of June
2. End of July
3. End of September
4. End of October

# Given DLX entity

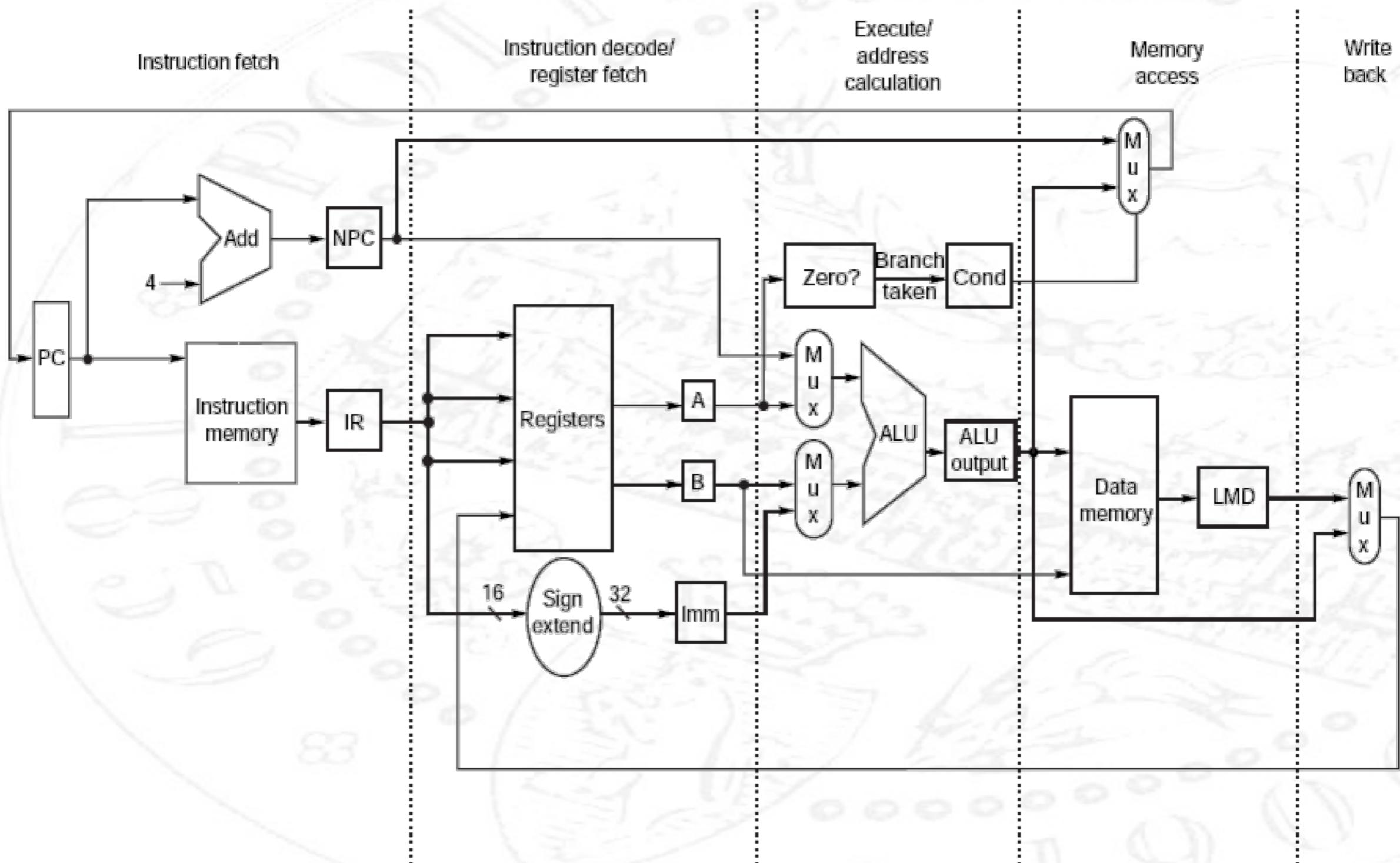


# Data Path

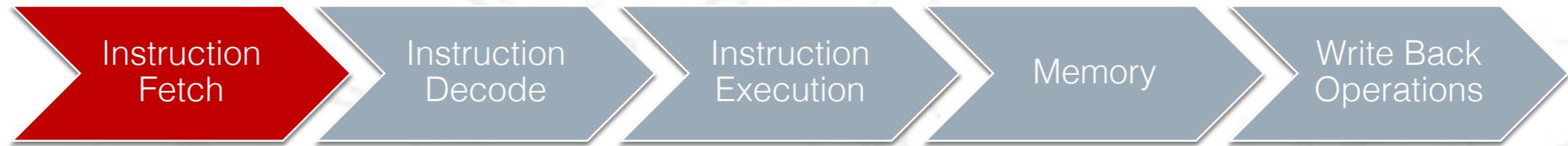


**POLITECNICO  
DI TORINO**

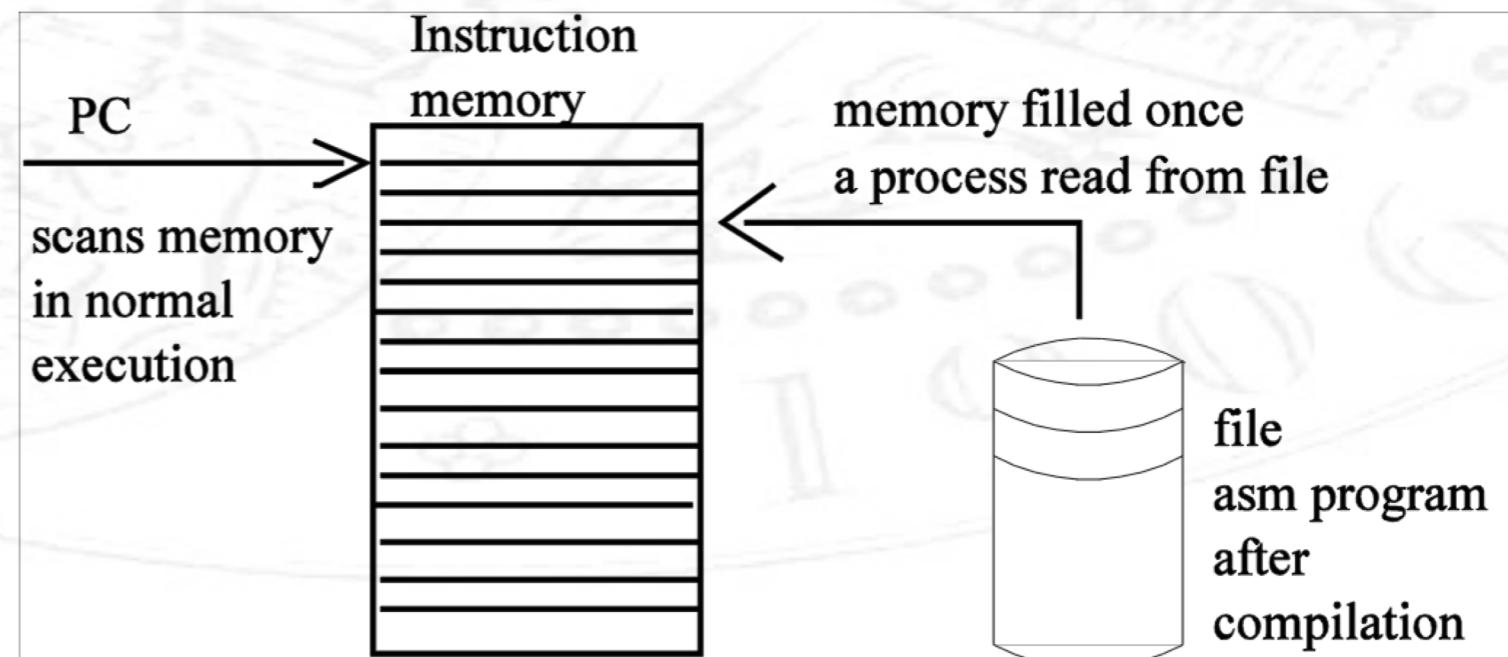
# DLX Data Path organisation



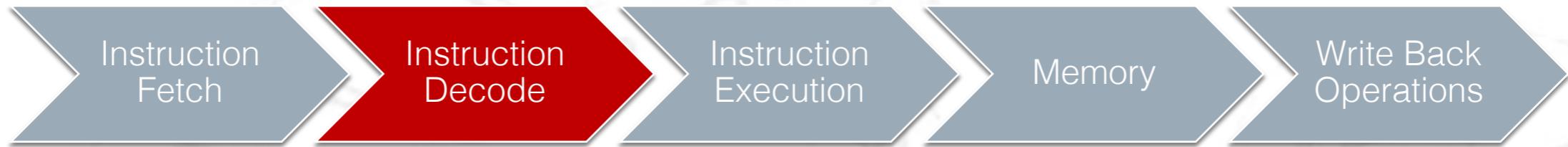
# Fetch



1. Send out the PC
2. Fetch the instruction from memory into the instruction register (IR)
3. Increment the PC (by 4) to address the next sequential instruction.



# Decode

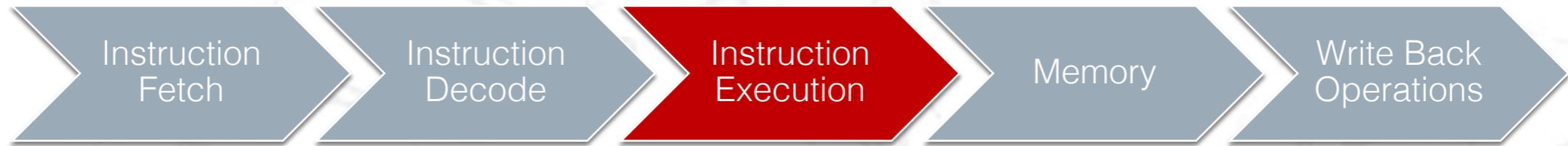


1. Decode the instruction
2. Access the register the (RF)
3. The outputs of the general-purpose registers are read into two temporary registers (A and B)

## Instruction set:

- R-type: operations between registers;
- I-type: operations between a register and an immediate value;
- J-type: jump/branch instructions.

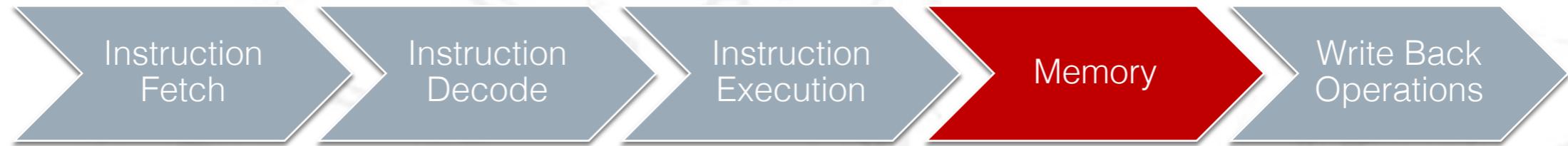
# Execution



1. The ALU operates on the operands (A and B) prepared in the previous cycle
2. The result is stored in the ALU Output register.

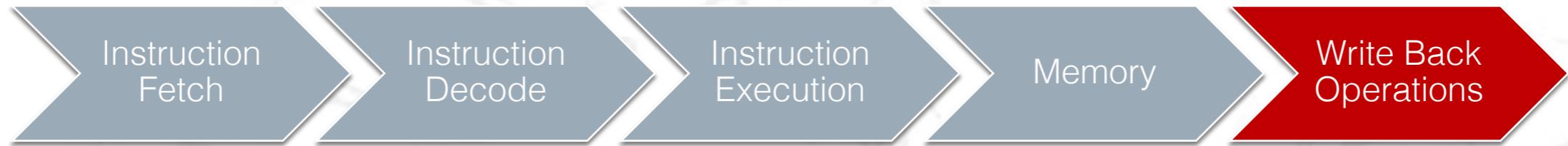


# Memory



- Access memory if needed.
- If the instruction is a load, data return from memory and is placed in the LMD (Load Memory Data) register.
- If it is a store, the data from the B register is written into memory.
- In both cases the used address is the one computed in the prior cycle and stored in the ALU Output register.

# Write Back



- Write the result into the register file, whether it comes from the memory system or from ALU

# BASIC vs. PRO Features



POLITECNICO  
DI TORINO

Basic  
DLX

# 5-stage Pipeline

## Limited Instruction Set

Data path

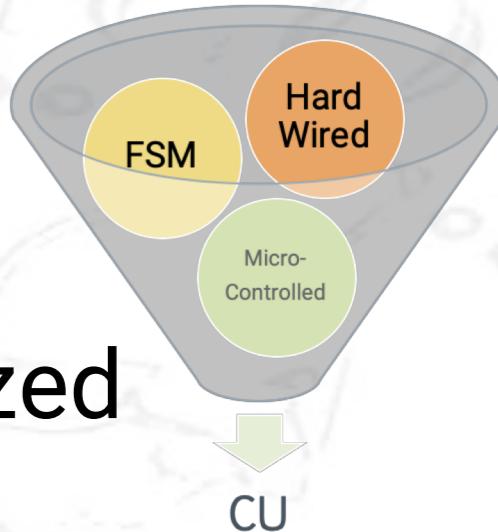
Synthesis

Physical design

Documentation

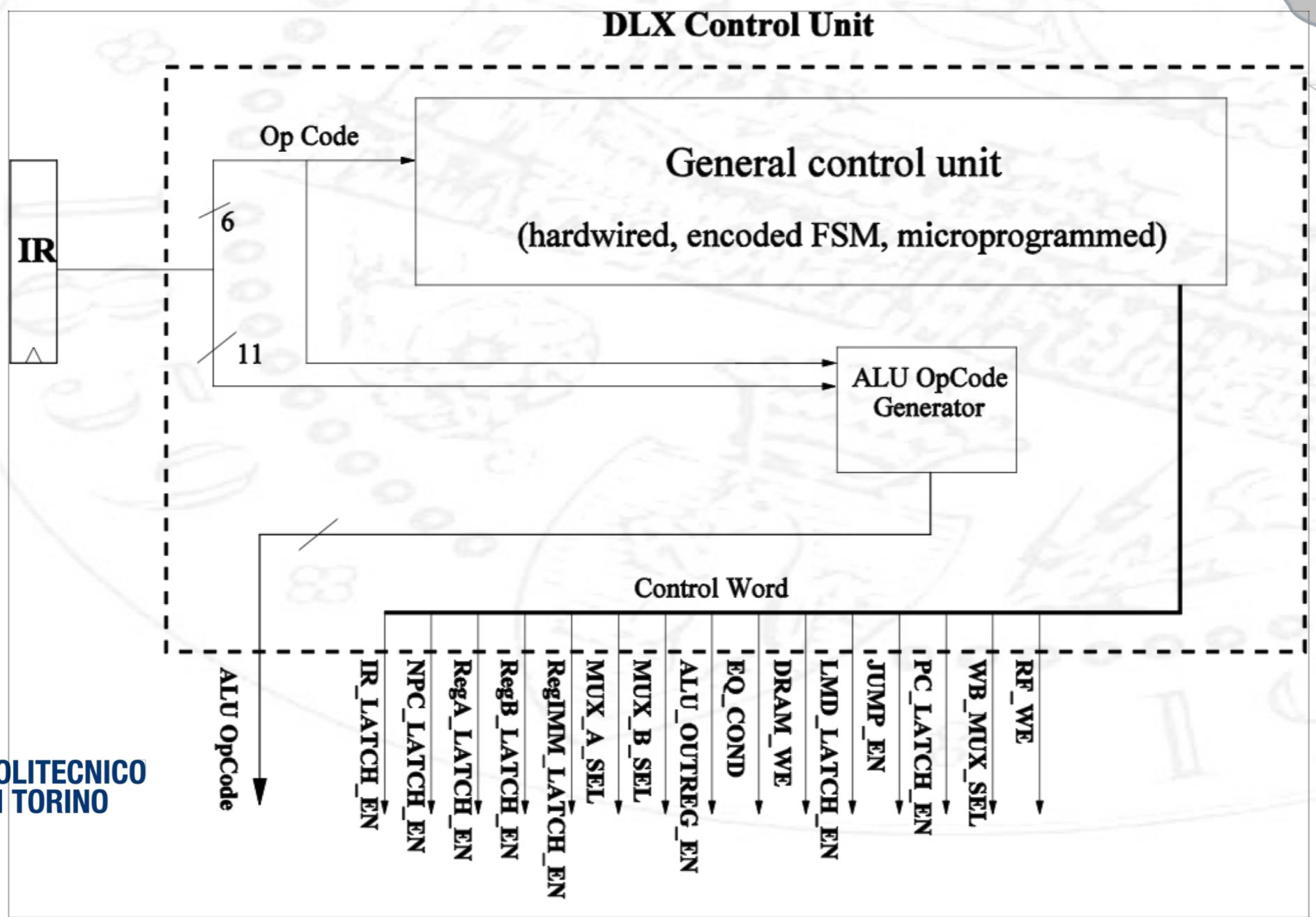
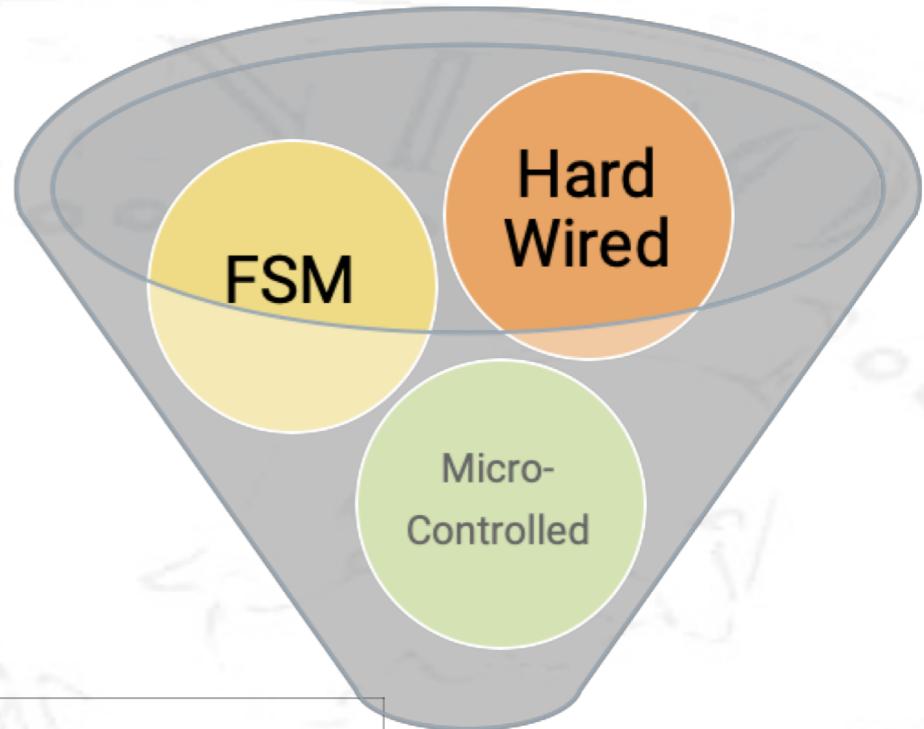


# Controlling a 5-stage Pipeline

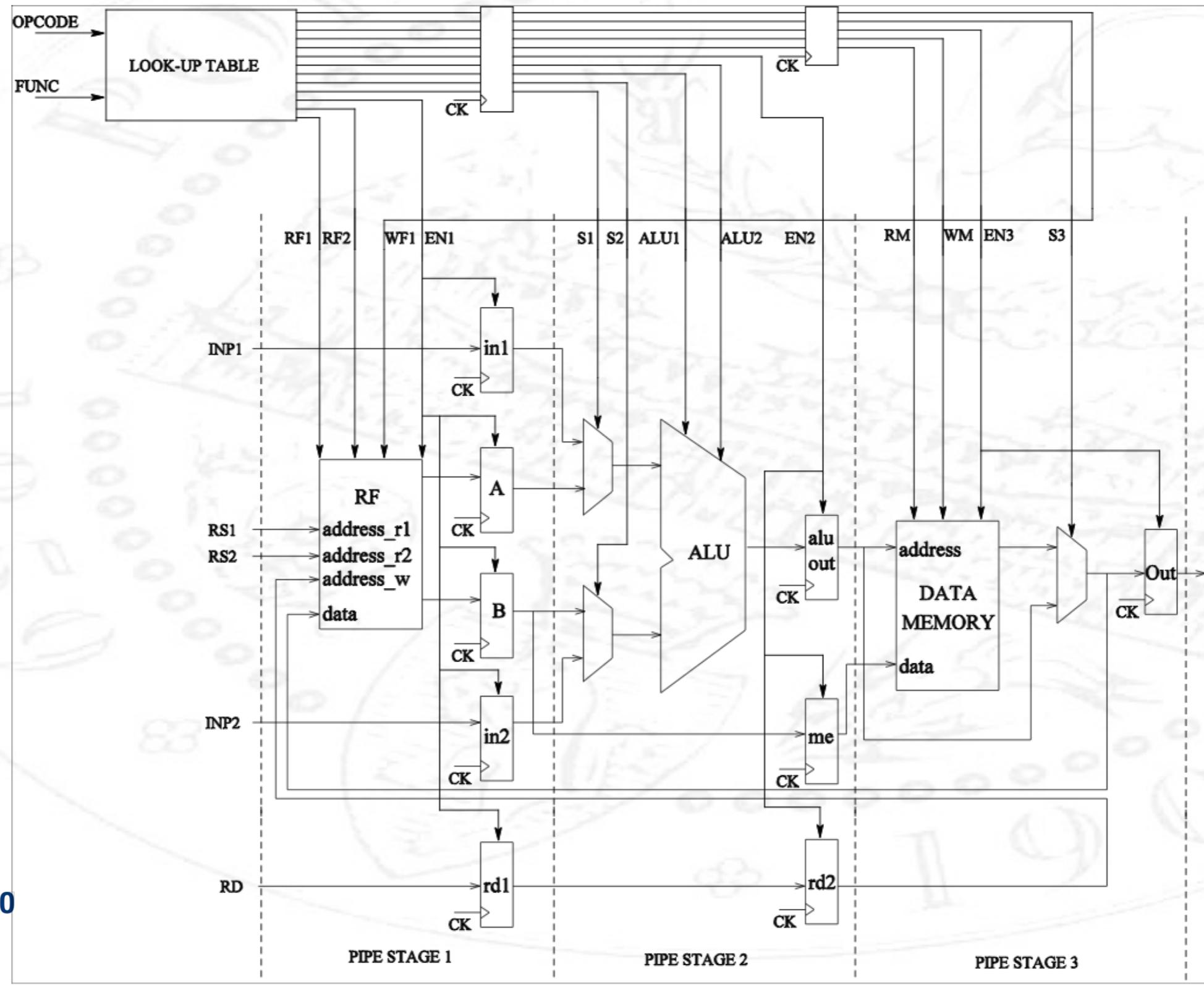


- The architecture and control must be organized following the five stages pipeline.
- You have three ways:
  - go further with microcontrolled CU,
  - or change to a clever FSM
  - or hardwired CU.
- Choose meaningful assembler programs so that pipeline and pipeline stall are underlined.

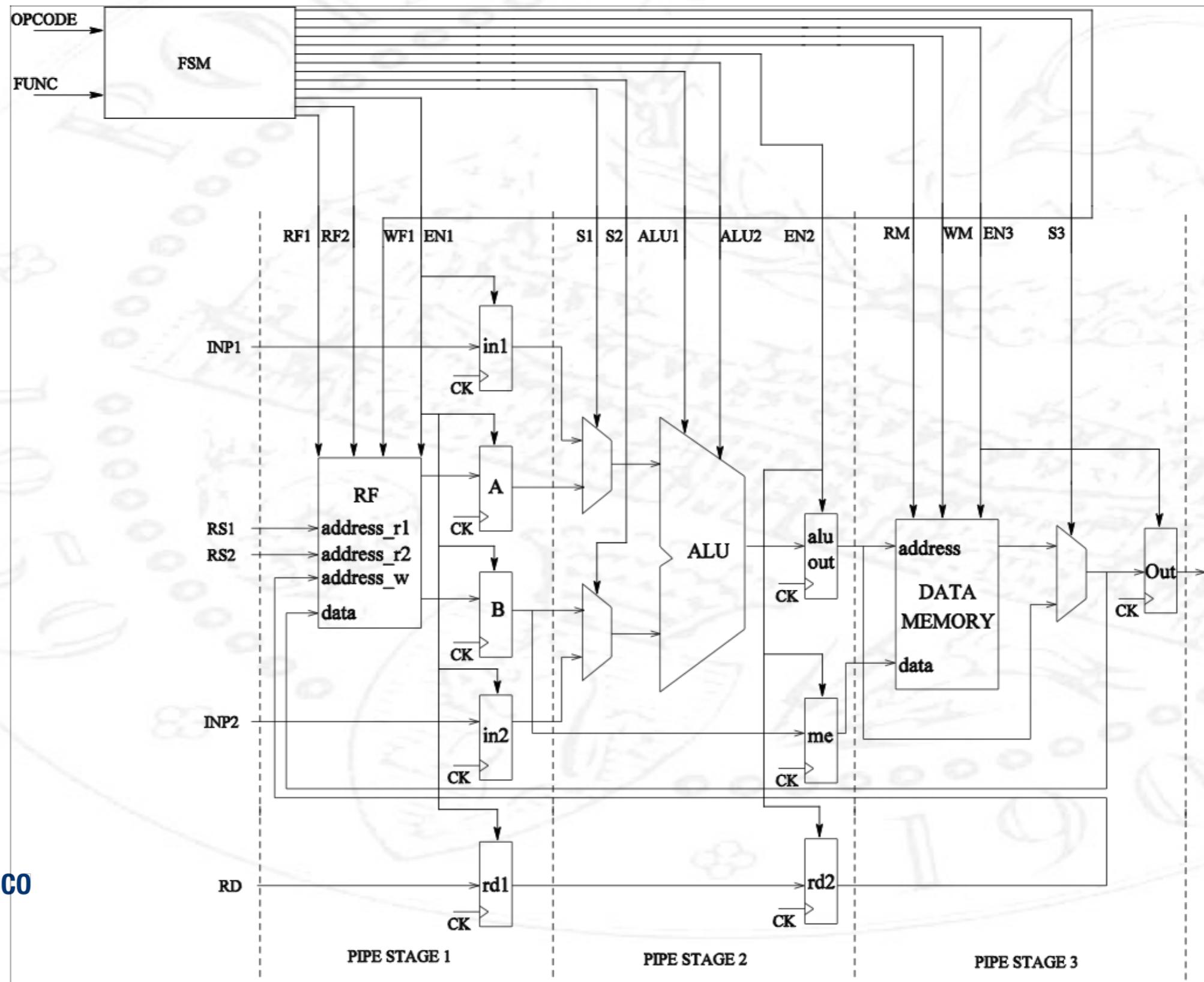
# Control Unit



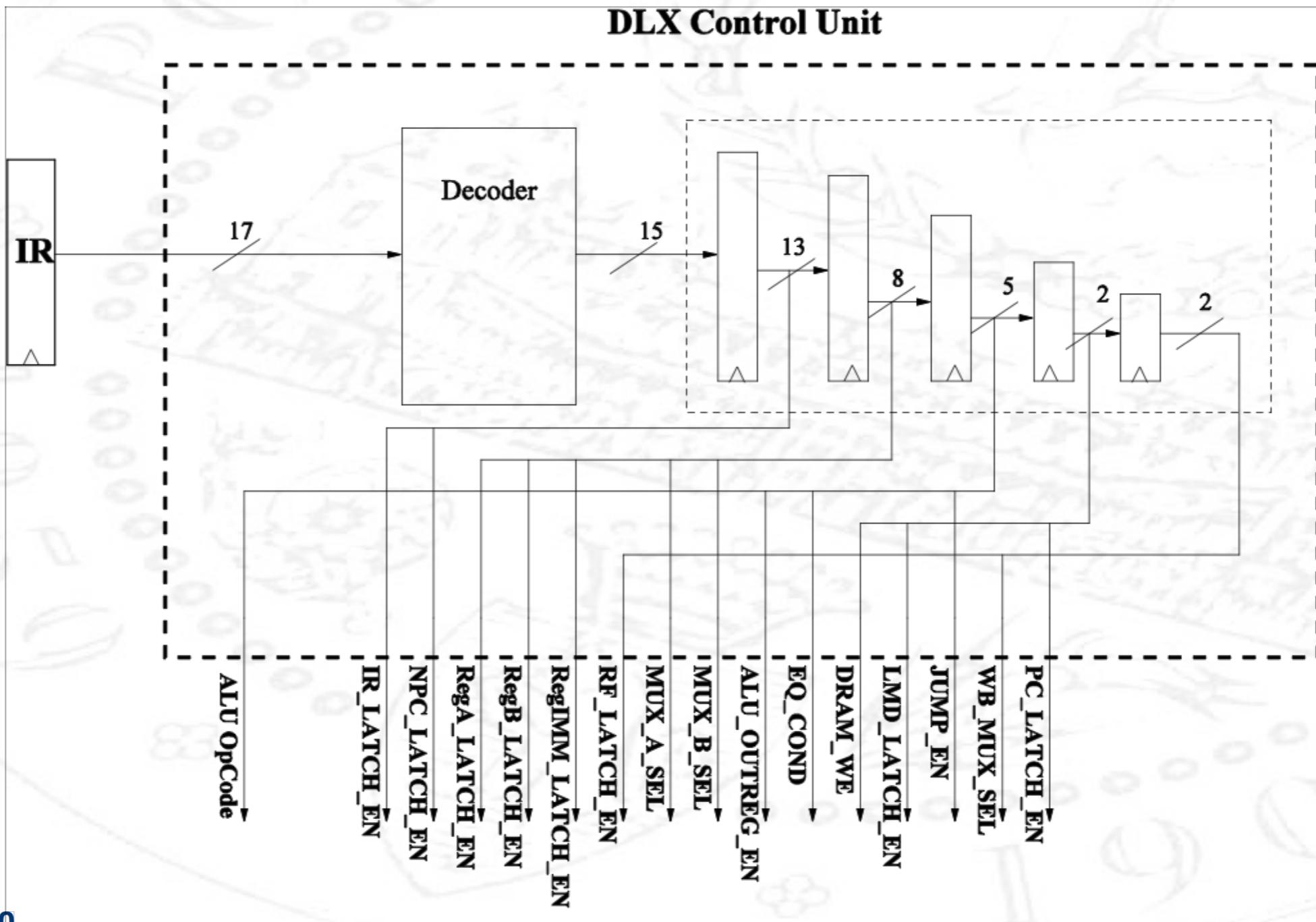
# Hardwired Control unit



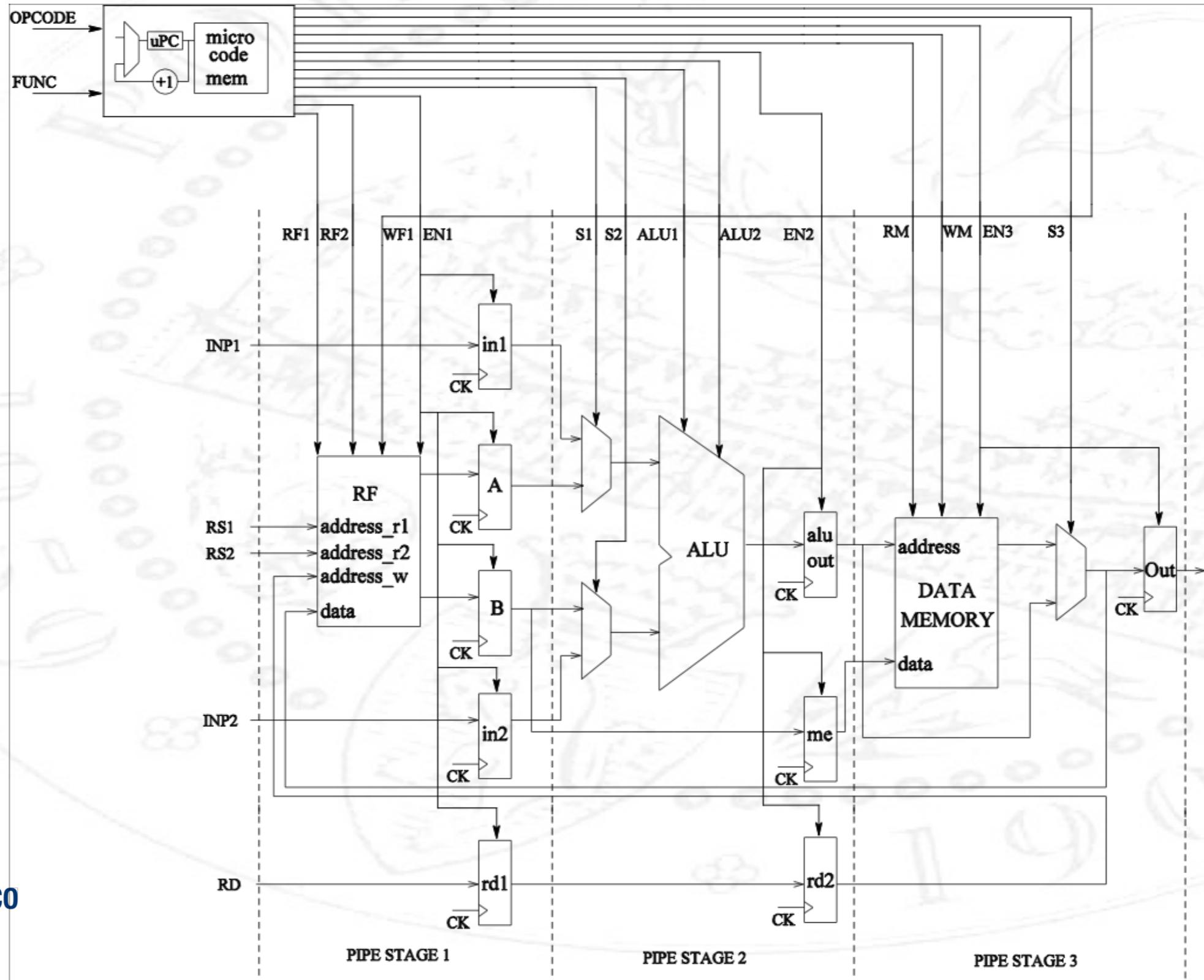
# FSM Control unit



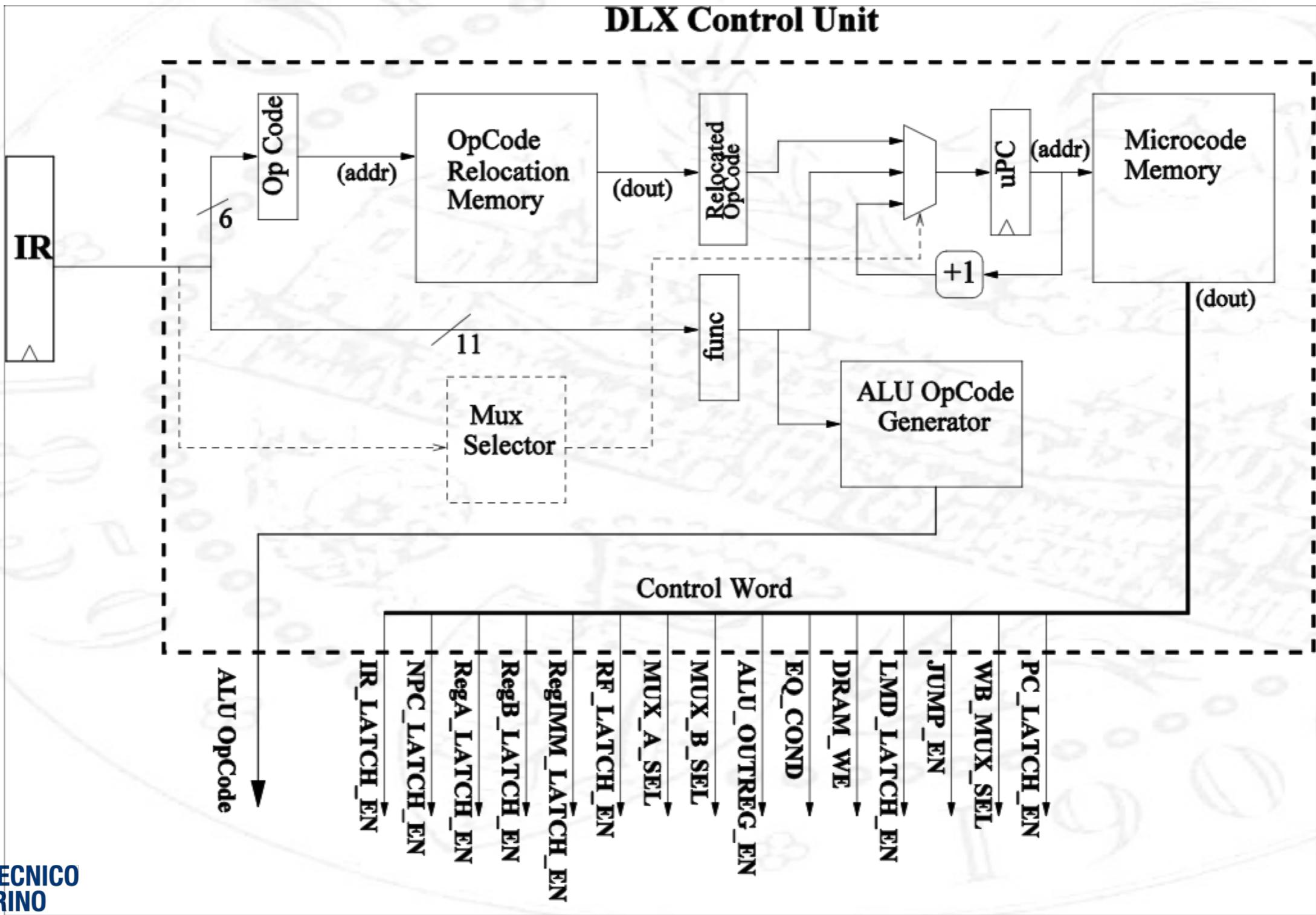
# FSM Control unit



# Micro-Programmed Control unit



# Micro-Programmed Control unit



# Limited Instruction Set

- add addi and andi
- beqz bnez j jal
- lw nop or ori
- sge sgei sle slei sll slli sne snei srl  
srli
- sub subi sw xor xori
- Design modify and comment one (or more) intelligent **assembler** programs so that these instructions can be meaningfully checked. Change the ALU OPCODE so that an enumeration type can be used (see appendices A and B).

# Data Path

- You should describe in **VHDL** at RT level all the data path components necessary to fulfill the instruction subset defined at previous points.
- Of course you can reuse the blocks you already described in previous **labs**.
- Describe one or more intelligent **assembler** programs (COMMENTED) so that these instructions can be meaningfully checked.

# Synthesis

- Both the **data path** and the **control unit** must be synthesized.
- **Different** synthesis results can be reported, commented and compared
- A final optimization for **frequency** must be performed.
- The **scripts** you use for the synthesis step must be reported and commented (memory should not be synthesized).

# Physical Design

- The synthesized design must be **placed and routed**.
- Post physical design performance must be reported:
  - delay,
  - thermal informations,
  - power...

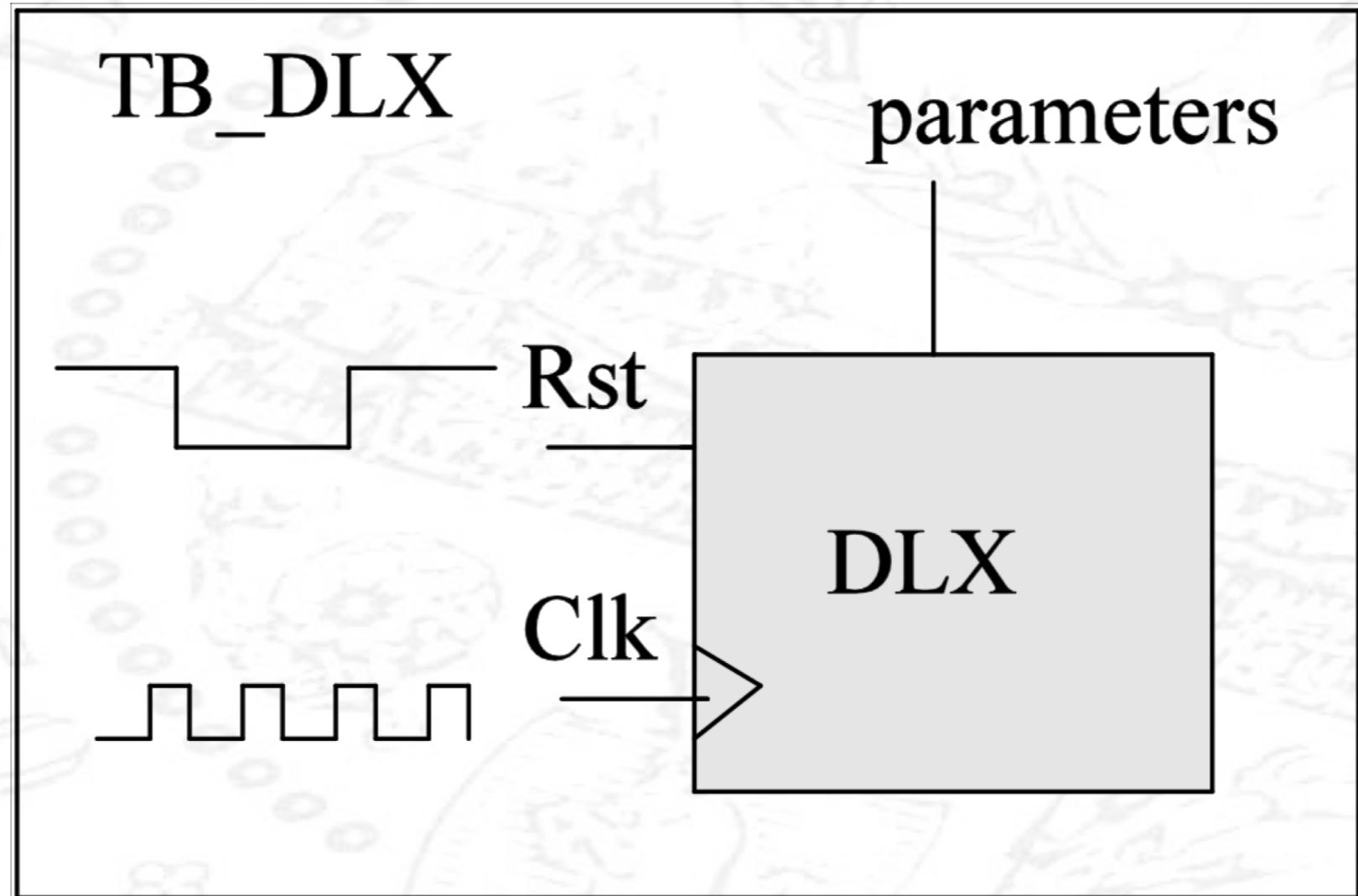
- Instructions subset: addu, addui, jalr, jr, lb, lbu, lhi, lhu, sb, seq, seqi, sgeu, sgeui, sgt, sgti, sgtu, sgtui, slt, slti, sltu, sltui, sra, srai, subu, subui, mult...
- **Data path:** The data path can be extended for each of the instruction you choose to add.
- **Windowed register file**
- **Control hazard:** Implement one or more of the techniques to prevent stall as mentioned during classes as instruction queue for jump (IQ), branch prediction using a small branch history table.
- **Power consumption optimization** from an architectural point of view
- **Caching:** Add a small data cache (RTL) between the main memory and the CPU;
- **Advanced synthesis:** Force further optimization to the design: try to reduce power consumption, perform a post synthesis VHDL simulation, so that realistic timing and power simulation with a real test bench can be performed
- **Physical design:** Post physical design simulations, clock tree synthesis and even crosstalk analysis would be appreciated in the final report.

# Testing the DLX



POLITECNICO  
DI TORINO

# Given Testbench



# Test your DLX

- Chose the assembler program to be executed:  
\*MANY examples are given DLX Project /asm example/  
(we recommend you to use the simplest one: test.asm)
- Compile it and convert to a readable format:  
`prompt> ./assembler.bin/dlxasm.pl test.asm`
- Convert the file format so that it can be easily read  
by the VHDL control unit:  
`prompt> ./assembler.bin/conv2memory test.asm.exe test.asm.mem`

# Test your DLX

Create your  
custom  
assembly

Convert the  
assembly into a  
binary file

Convert the  
binary file in a  
format  
compliant with  
your architecture



# First steps...

- 1) Understand how to generate assembler files
- 2) Run a basic **simulation** of the given structure
- 3) Try to add the management of a new **ASM instruction** (follow file) of the CU
- 4) Build a very simple **DATA PATH** by connecting all the elementary blocks already used in labs 1-2-3-4
- 5) Connect simple **DATA PATH**

# Report Guidelines



POLITECNICO  
DI TORINO

# Report Structure

- Header
- Summary
- Index of contents, figures and tables
- Body
  - **INTRODUCTION** (SPECIFICATION and FUNCTIONALITY)
  - **FUNCTIONAL SCHEMA** (Block diagrams)
  - **IMPLEMENTATION** (Technology, synthesis, optimization)
  - **DISCUSSION** and **CONCLUSIONS** (Result from the BenchMark, timing, area)
- References
- Appendix

LaTeX.

# Practical Info.

- Lab groups and Project groups can differ
- The basic vs. Pro choice is individual (please communicate the variation)
- You have to submit your files and final report at least one week before the discussion
- The .zip submission is made through the student web page
- Name format:
  - GRXX\_DLX\_basic.zip
  - GRXX\_DLX\_pro.zip
- The internal organization of folders and files must strictly respect the rules written in the PDF Project Guide.

# The final discussion

- The final discussion is individual
- You will be asked to:
  - Give an overview of the architecture
  - Demonstrate the correct behavior by using your custom ASM files
  - Be ready to execute (at run time) some random ASM program
  - Show the synthesis and physical design results
  - Final discussion with prof. M. Graziano