DLX Project

POLITECNICO
DI TORINO

Design steps

Architectural design of DLX processor

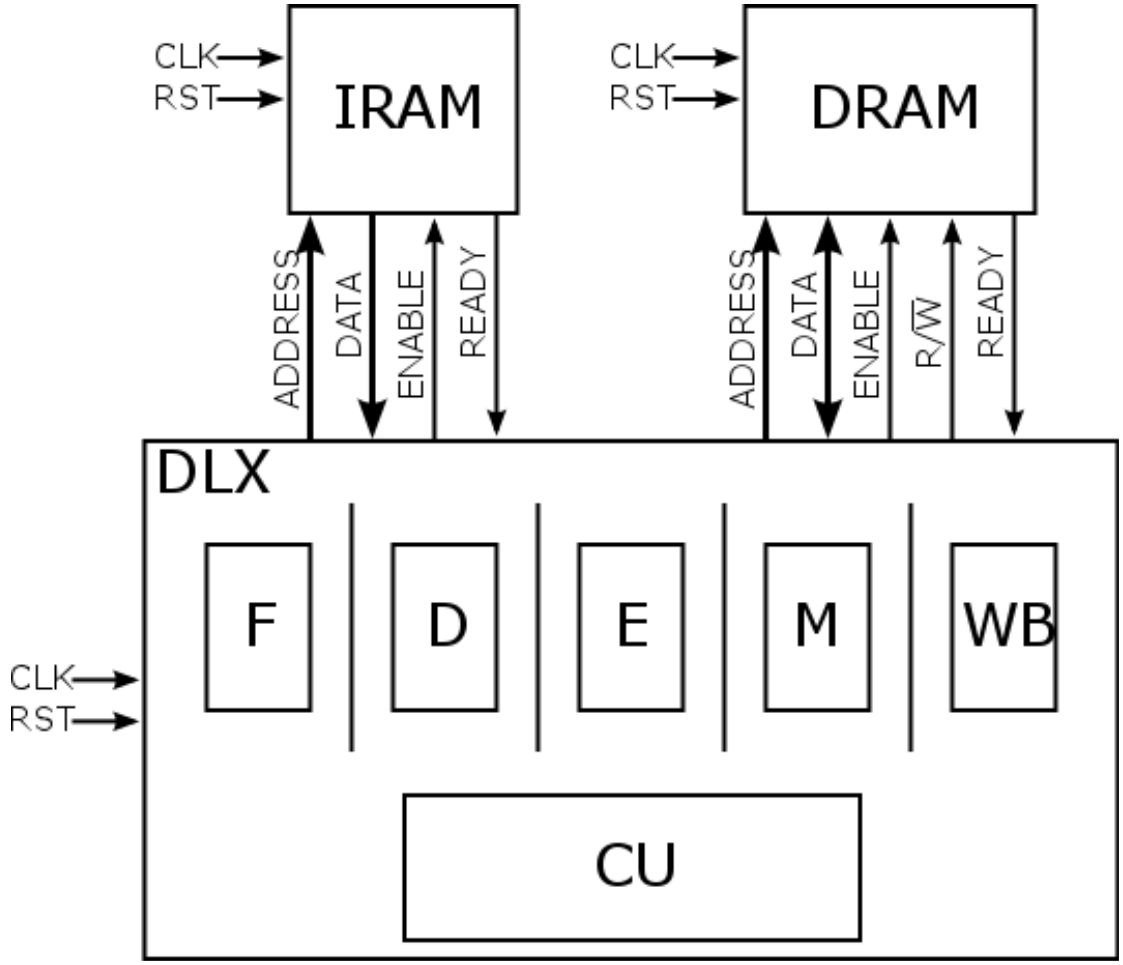
Simulation & Benchmarking

Synthesis

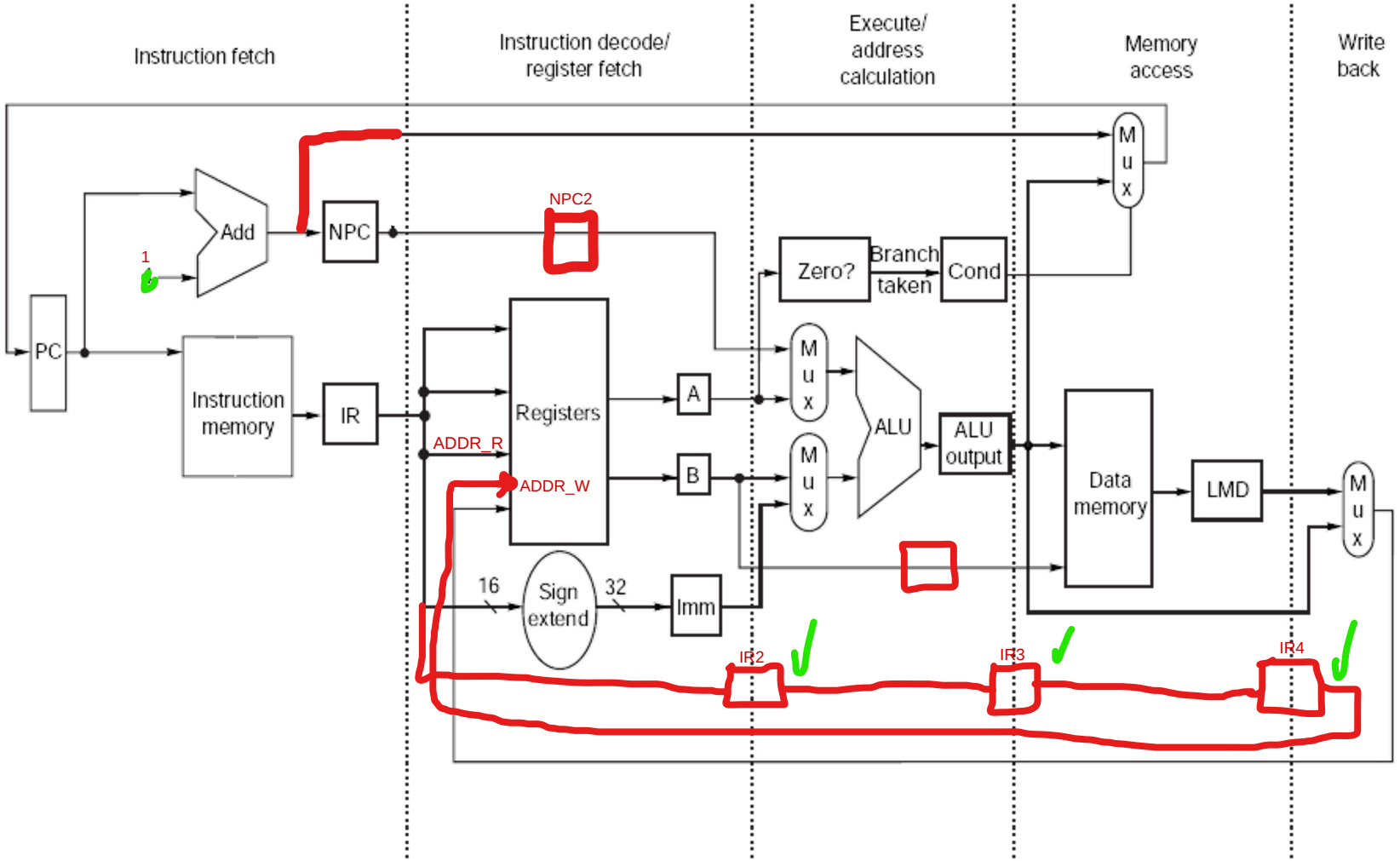
Physical Design

Documentation

Data Path

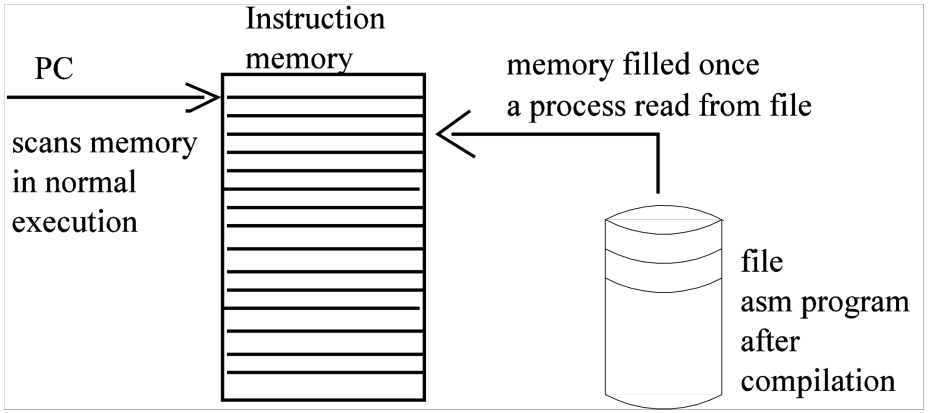

CLK
RST
IRAM
CLK
RST
DRAM
ADDRESS
DATA
ENABLE
READY
ADDRESS
DATA
ENABLE
R/W
READY
DLX
F
D
E
M
WB
CLK
RST
CU

Basic Architecture



Fetch Phase

- Send out the PC
- Fetch the instruction from memory into the instruction register (IR)
- Increment the PC (by 4) to address the next sequential instruction. The NPC register is used to hold the next sequential PC.

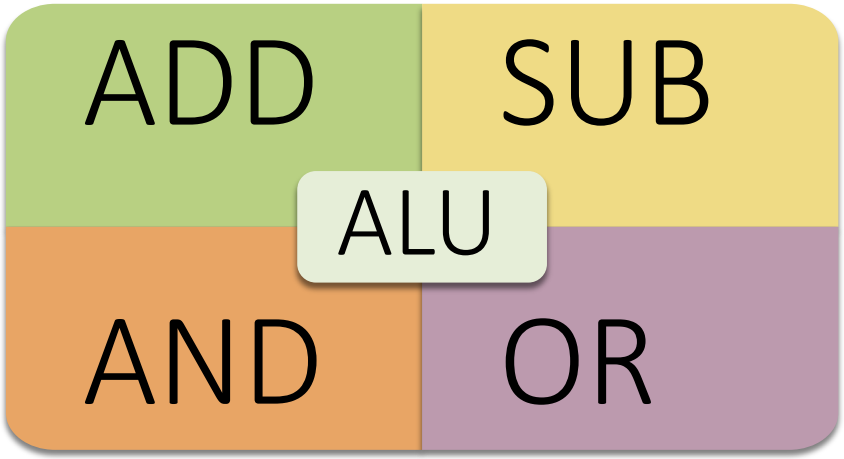


Decode

- Decode the instruction
- Access the register le (RF) to read the registers.
- The outputs of the general-purpose registers are read into two temporary registers (A and B).
  - I-type: Loads/Stores and conditional branch instructions.
  - R-type: Register-register ALU operations. ALU operation is defined in the extra 11-bit field func.
  - J-types: Jump and jump link instructions

Execution

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



Result Storage

- 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-version Features

- 5 stages pipeline
- Limited instruction set
- Basic Datapath
- Basic Synthesis
- Basic Pyhisical Design
- Complete documentation

5-Stage Piepline

- The architecture and control must be organized following the five stages pipeline.
- You have three ways to control it:
  - Mcrocontrolled CU,
  - FSM
  - Hardwired CU.
- Choose meaningful assembler programs so that pipeline and pipeline stalls are underlined.

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.

Basic 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.
- 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.

Basic Sythesis

- Both the data path and the control unit must be synthesized following the instruction you will see during the Labs.
- Different synthesis results can be reported.
- 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).

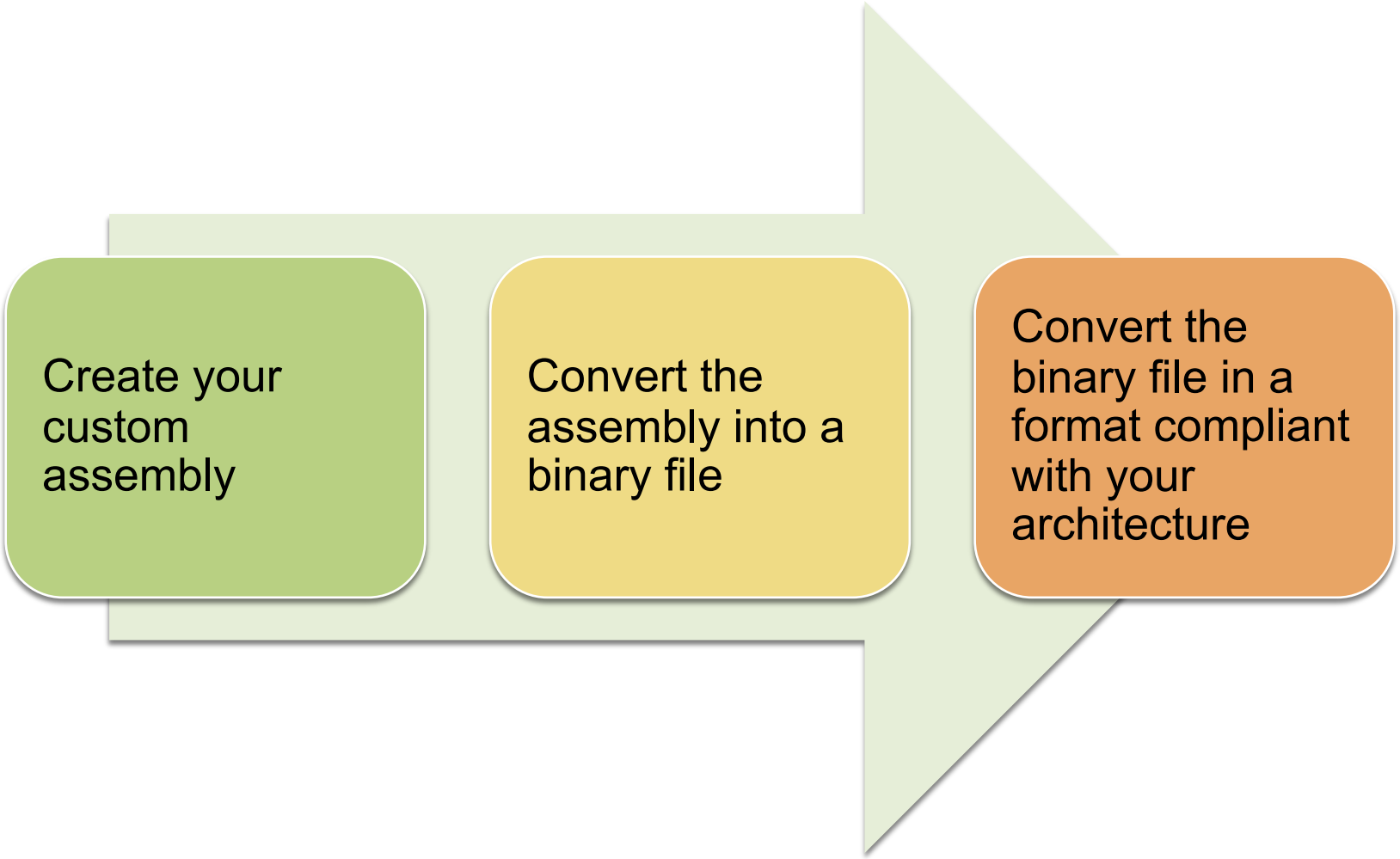
Physical Design

- The synthesized design must be placed and routed.
- Post physical design performance must be reported:
  - Delay,
  - EM,
  - Thermal informations,
  - Power...

Pro-feature Hins

- Extend the intruction set to support other operations
- Add the windowing register file
- Try to control Hazards
- Try to optmize power consumption
- Create advaced synthesis scripts
- Improve the physical design
- Create custom script to automate simulations etc…

Test you DLX


Create your custom assembly
Convert the assembly into a binary file
Convert the binary file in a format compliant with your architecture

Custom *asm* files

- Chose the **assembler** program to be executed: many examples are given DLX Project /asm example/ (to start, we recommend you to use the simplest one: test.asm)

- **Compile** it and convert to a readable format (per script):
  - `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`

Steps to start

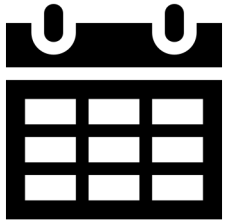
- Try to interconnect all the elementary blocks already used in labs
- Understand how to generate assembler files
- Run a basic simulation of the given structure
- Try to add the management of a new ASM instruction (follow file) of the CU

Marks and Deadlines

- **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 31st
  - Max: October 30th
  - All groups are required to submit all files one week before the discussion

Discussion sessions

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



Precise scheduling will be given at the beginning of each month.

Writing the report

- Header
- Summay
- 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

General information

- Lab groups and Project grops can differ
- The basic vs. Pro choice is individual (please communicate the variation)
- You must 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.
- All code files must contain comments

Final discussion

- Althoug the you will discuss you project with your teammates, the final discussion is individual
- You will be asked to:
  - Give an **overview** of the architecture in **10 minutes** (you can use some slides or any other material you want…)
  - You will be asked to demonstrate the correct behavior by using your **custom *asm*** files
  - Be ready to execute and **simulate** (at run time) some random *asm* program
  - Show the **synthesis** and physical design results

People

Organization questions:
giovanna.turvani@polito.it

andrea.coluccio@polito.it

Technical questions and consultancy:
andrea.coluccio@polito.it

andrea.marchesin@polito.it