# DLX PROCESSOR - PRO VERSION

*Microelectronic systems - 2015/2016*


Authors: Lorenzo Chelini, Christian Palmiero

Referents: Prof. Mariagrazia Graziano, Giulia Santoro

# Summary

In this project we have refined the DLX processor from RTL down to the synthesis and to the physical design phase. Our project consists of four modules connected together according to the structural approach: a complex data path, an hardwired control unit, a data memory and an instruction memory.
Why should this project be considered a PRO version?

- Instruction subset: other than the basic instructions, the DLX is able to perform 8 additional instructions (SEQ, SEQI, SGT, SGTI, SLT, SLTI, MUL and partially the DIV).
- Data path: the microarchitecture is optimised so that the critical path and the power consumption is reduced and most of the modules, from the basic ones to the arithmetical ones, recall the ones analysed during lectures and suggested during labs (T2 ALU, P4 adder, Booth MUL, divider module with the non-restoring algorithm).
- Control hazard: the control unit is able to implement the static branch prediction technique called "predict untaken".
- Data hazard and forwarding: the hazard unit integrated in the decode stage is able to solve RAR, WAR hazards, while the forwarding unit integrated in the execution stage is able to solve EX/MEM hazards and MEM/WB hazards.
- Power consumption: two main techniques have been applied in order to reduce power, clock gating and guarded evaluation.
- Physical design: a complete and accurate physical design has been executed according to all the instructions of the fifth lab of the course.

# Table of contents

# Introduction

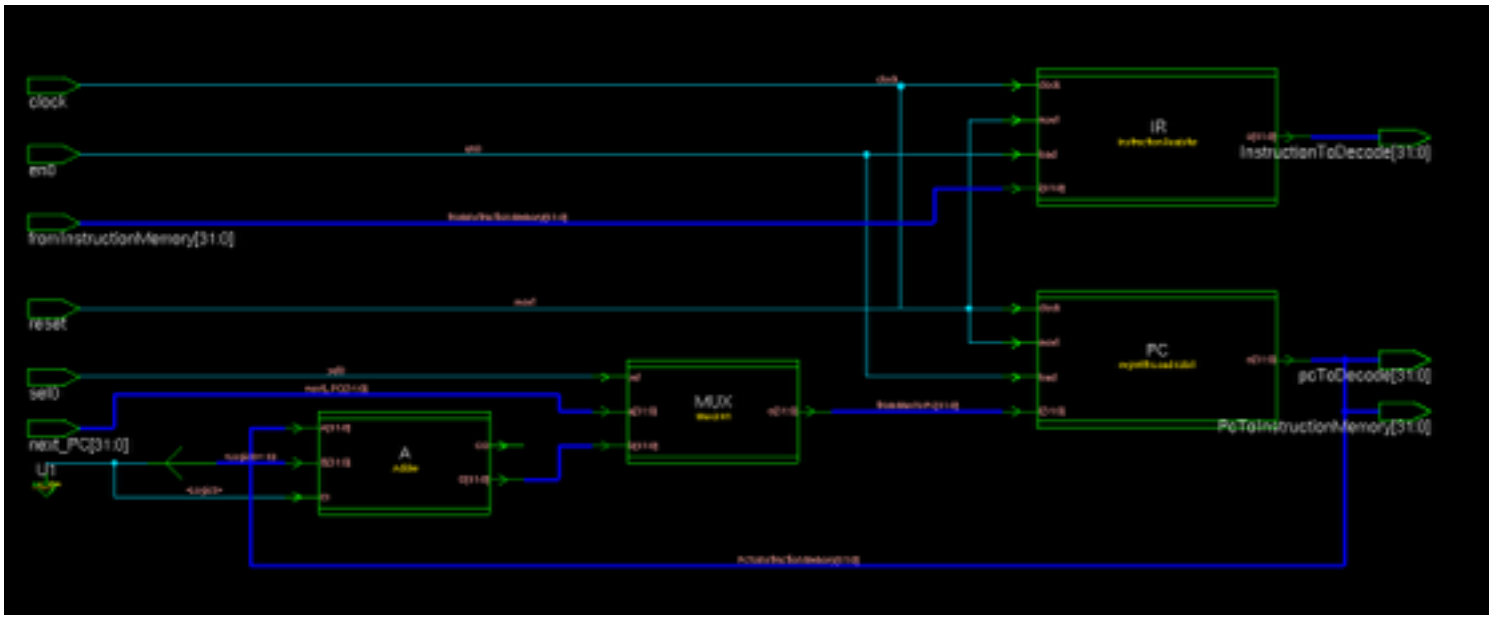
## What is it and what does it do?

The DLX (pronounced "Deluxe") is a RISC processor architecture designed by John L. Hennessy and David A. Patterson, the principal designers of the Stanford MIPS and the Berkeley RISC designs.
The DLX, a cleaned up and a modernised simplified MIPS CPU, has a simple 32-bit load/store architecture and emphasises a design for a pipelining efficiency and an easily decoded instruction set.
The DLX has thirty-two 32-bit general purpose registers (GPRs), named R0, R1, ..., R31 (the value of R0 is always 0) and thirty-two 64-bit floating-point registers (FPRs), named F0,F2,…,F30.
The DLX operations work on 32-bit integers and 32- or 64-bit floating point. The memory is byte addressable with the Big Endian mode, the address is on 32 bits and the available addressing modes are the immediate one and displacement one. Furthermore, all memory accesses must be aligned.
The instructions have a fixed length on 32 bits and can be broken down into three subsets: I-type, R-type, J-type.
The DLX, like the MIPS design, bases its performance on the use of an instruction pipeline. The pipeline contains five stages:

- IF – Fetch unit
- ID – Decode unit
- EX – Execution unit
- MEM – Memory unit
- WB – Write back unit

# Functional Schema

## Fetch stage



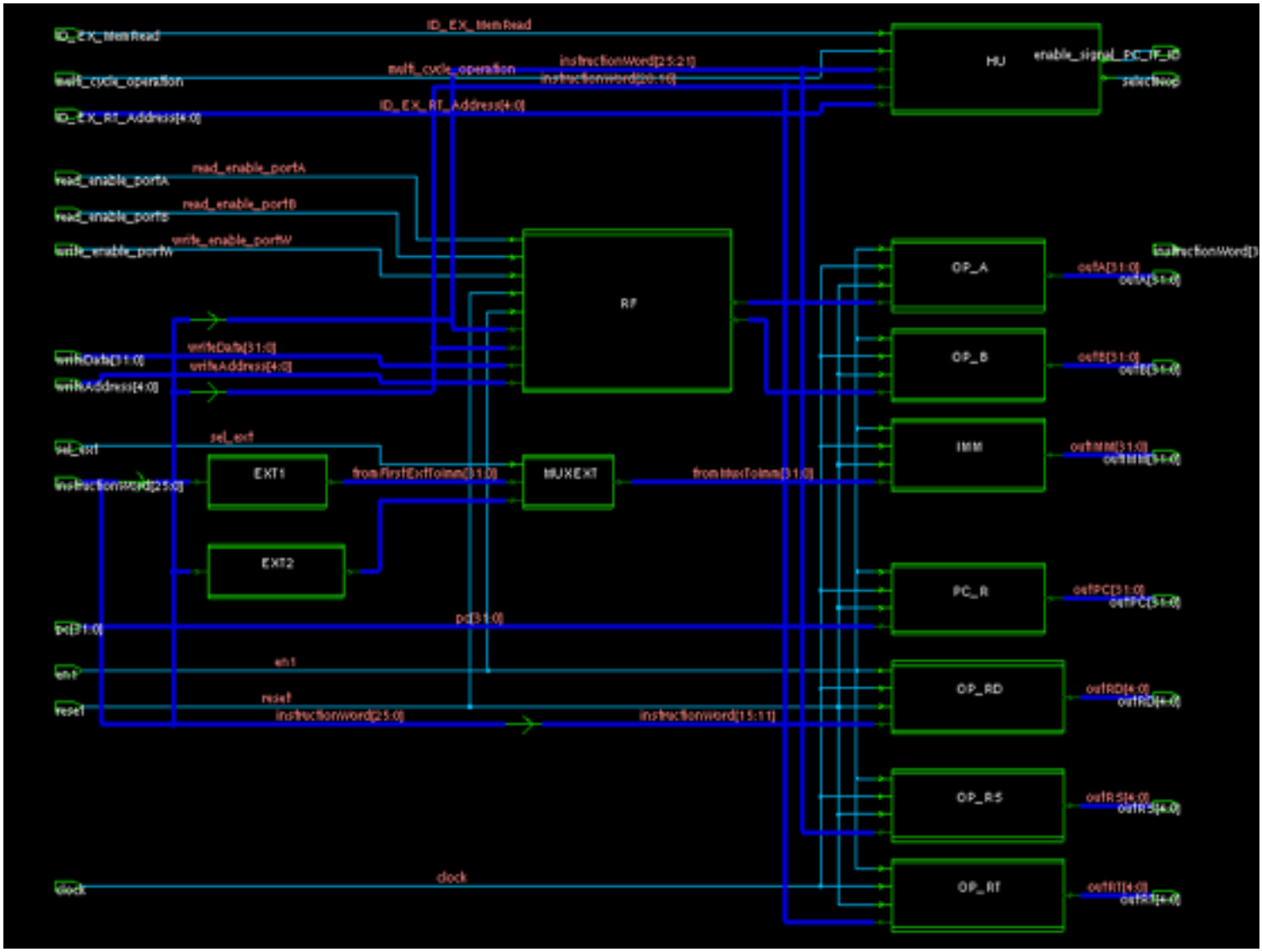

The fetch stage consist of:

- the program counter (PC), a register that stores the address for accessing the instruction memory;
- the instruction register (IR), a register that stores the instruction coming from the instruction memory;
- an adder, that computes the NPC value (PC+4);
- a multiplexer, that selects among the incremented PC and the new value of the PC (if the next instruction is a jump or a taken branch).

## Instruction memory

The instruction memory is an asynchronous ROM memory, initialised with the binary code of the assembler program to be executed. According to the project guidelines, it has not been synthesised.

# Decode stage

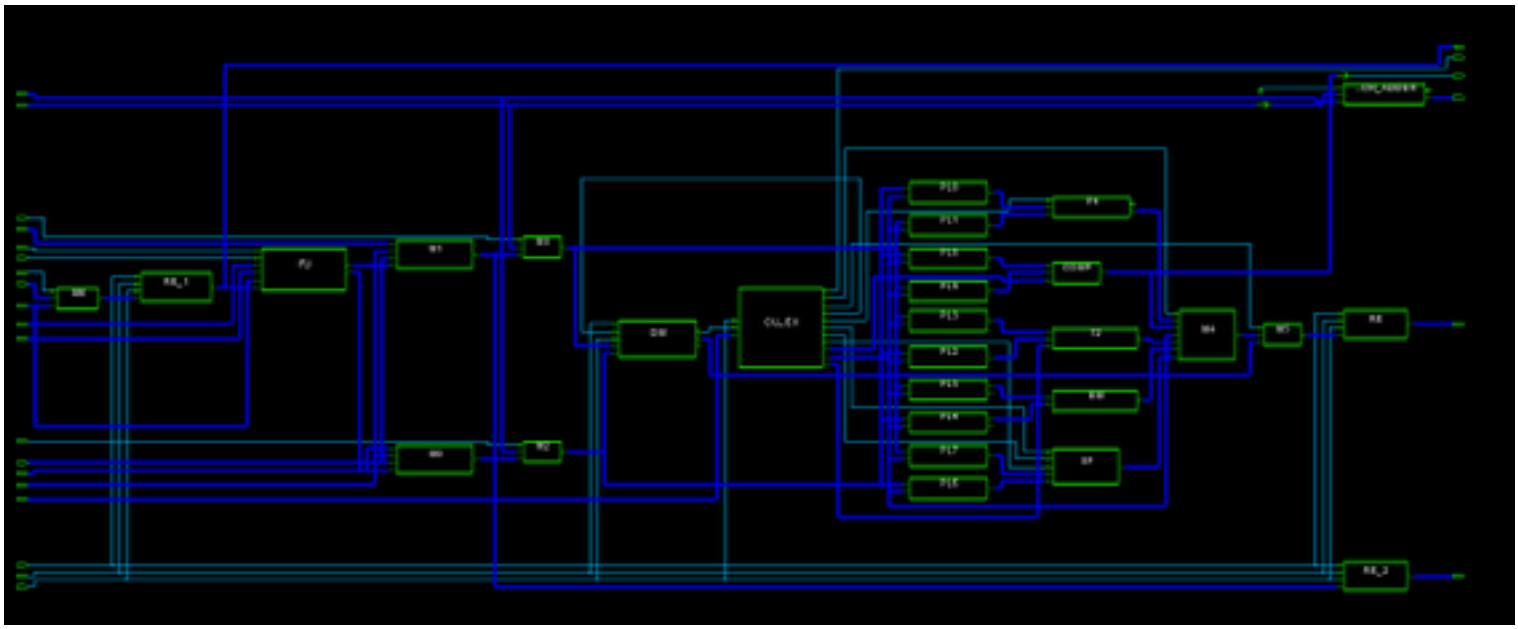



The decode stage is composed of:

- three registers, OP_RD, OP_RS, OP_RT, each of which receives the corresponding instruction word field;
- a 1 write port, 2 read port-register file, that is accessed asynchronously both in reading and in writing and that is composed of thirty-two 32-bit general purpose registers (R0, R1, ..., R31);
- two registers, OP_A and OP_B, storing the output operands coming from the register file;
- an IMM register, that takes the immediate operand extended on 32 bit;

- a pipeline register called PC_R, that takes the next program counter and forwards it to the next stage;
- the hazard detection unit, a module that, by inspecting three inputs signals, generates two outputs: the former is used to disable the IF/ID register in case an hazard is detected; the latter is used as a condition signal for the control unit, that will force a NOP instruction if this signal is asserted.

# Execution stage



The execution stage is composed of different sub-systems:

- The structural ALU, that in turn uses five different modules:
  - the Pentium 4 adder on 32 bits, that performs addiction and subtraction;
  - the Booth multiplier, that performs the multiplication of two 16-bit operands and gives a result on 32 bits;
  - the T2 logical unit, that performs all the logical operations using two NAND gates levels;
  - a shifter, that performs logical and arithmetic shift operations;
  - a comparator.

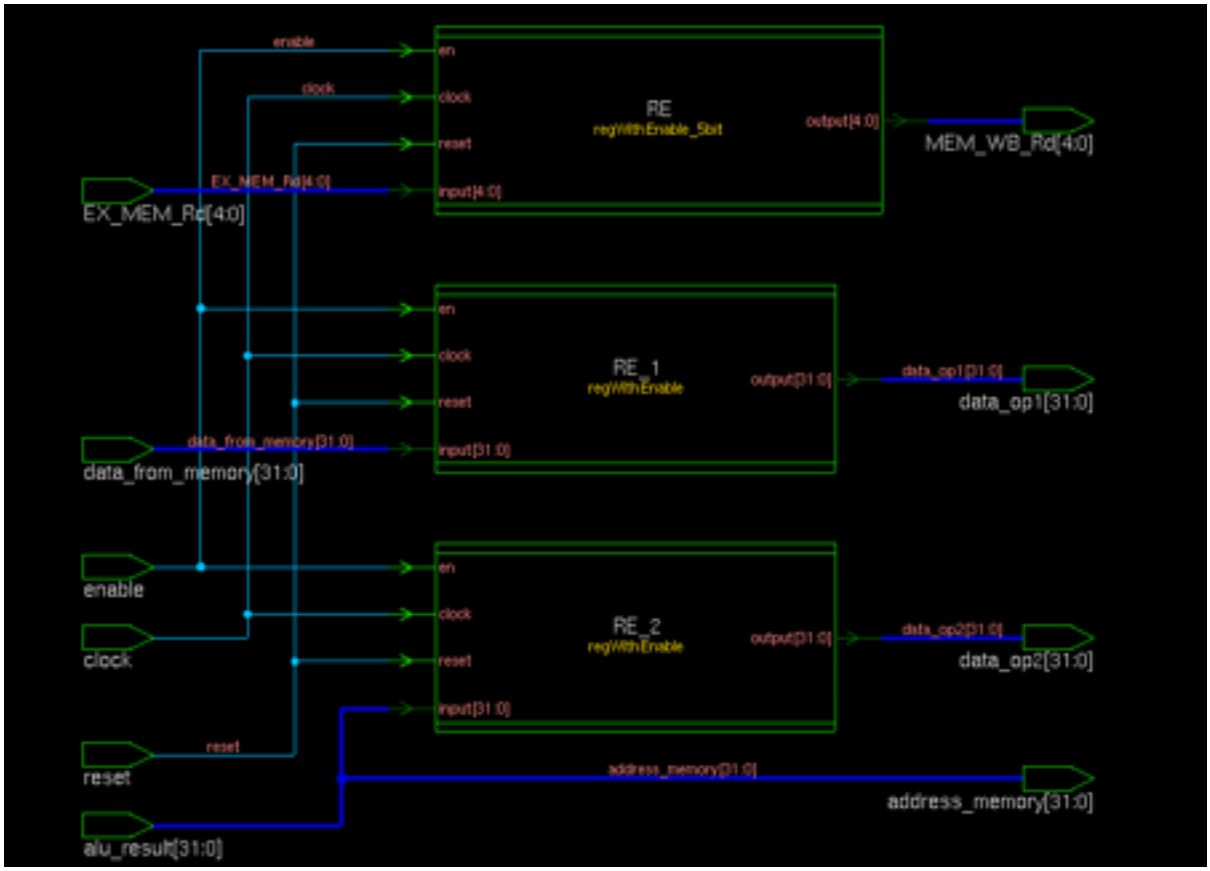
  By now, is worthy to mention that, in order to achieve better performances in terms of power consumption, a guarded evaluation is implemented. Additional latches and some combinational logics are inserted to all the inputs of each combinational block in order to set an idle condition.
- The division module, that is implemented according to the non restoring algorithm structure. Division is a long latency operation (34 clock periods),

hence this module is separated from the structural ALU, where all operations are done in just one clock cycle.[1]

- The exe control unit, shortly EXE_CU, that, according to the value of the FUNC signals, sends the proper configuration to all the sub-units. Since the division is a long latency operation, the EXE_CU sends a conditioning signal to the main control unit in oder to force it to insert a NOP operation and to preserve the in-order flow of instructions.
- The forwarding unit, that eliminates data hazards involving arithmetic instructions. It detects hazards by comparing the destination registers of the previous instruction with the source registers of the current instruction. Hazards are avoided by grabbing results from the pipeline registers before they are written back into the RF.

[1] NB: the division algorithm we have implemented is the one we have analysed during lectures. It works properly only on floating point numbers and under certain mathematical conditions. Since we do not have FPRs in our register file and according to the fact that we have experienced some problems in handling multi-cycle operations, we have decided NOT to implement the DIV instruction in the control unit. Anyway, all the modules are kept connected together and the DIV instruction has been tested in a test bench that takes into account only the division module itself.

# Memory stage



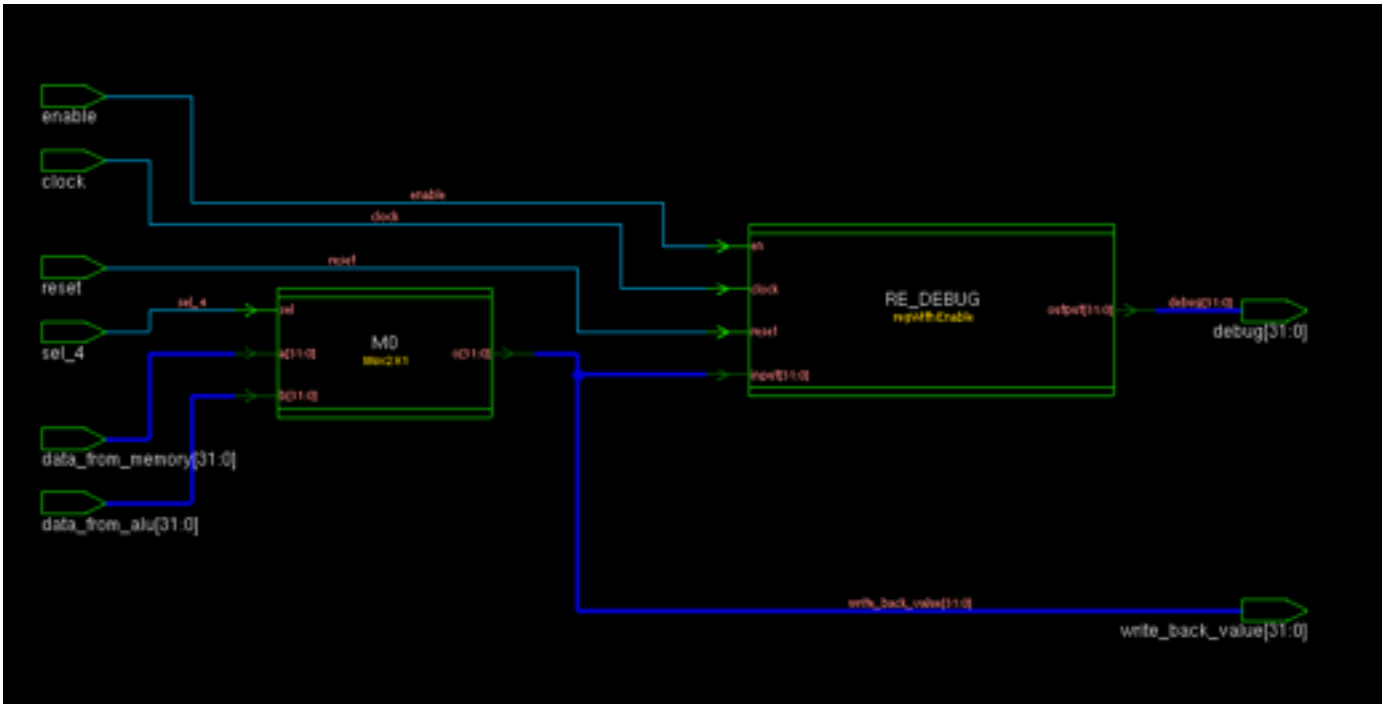

The memory stage consist of three registers:

- RE, a pipeline register that stores the address for writing to a specific register file location;
- RE_1, a MEM/WB pipeline register that stores the ALU result;
- RE_2, a MEM/WB pipeline register that stores the data coming from the data memory.

# Data memory

The data memory is a synchronous Write, asynchronous Read RAM memory with an active high reset signal, an active high read enable signal and an active high write enable signal. According to the project guidelines, it has not been synthesised.

# Write back stage




The write back stage only consist of two entities:

- a 2x1 multiplexer, that takes as input signals both the memory output and the ALU output and sends the output signal back to the decode stage and to the execution stage;
- a register called RE_DEBUG, that has debug purposes.

# Control Unit

The control unit is one of the most complex module of the entire project and it has been designed according to the HARDWIRED approach. The control unit receives the OPCODE (6 bit) and the FUNC (11 bit) signals and generates the control signals. The control word is read from a simple Look-up table, while the control signals for the exe control unit are generated by a process based on a VHDL case statement. In a fully pipelined structure, the control signals are delayed to match the delay of the data path.

The input port relies on the following signals:

- the clock;
- the reset, an active high reset;
- a stall signal from the hazard unit, that cause the control unit to introduce a stall if this signal is equal to '0';
- a jump flag, used for branches;
- the opcode and the "FUNC" field of the current instruction;

On the output side, each stage of the pipeline takes its own control signals:

- the decode stage takes an enable signal for its registers, one enable signal for each of the two read ports of the register file, a signal that selects either the immediate on 16 bits or the immediate on 26 bits;

```
--DECODE
en1 : out std_logic; --Decode registers enable
en2 : out std_logic; --PortA read enable
en3 : out std_logic; --PortB read enable
sel_ext : out std_logic; --Mux (IMM26/IMM16) selection signal
```

- the execution stage takes an enable signal for its registers, an hazard condition signal equal to 1 if the instruction is a load, four selection signals for four multiplexers (PC/A, IMM/B, RT i-type/RD r-type, PC+1/NPC), a FUNC field for the exe stage control unit;

```
--EXECUTION
hazard_condition : out std_logic; --Hazard condition (1 if the istruction is a LOAD)
sel1 : out std_logic; --Mux (PC/A) selection signal
sel2 : out std_logic; --Mux (IMM/B) selection signal
sel3 : out std_logic; --Mux (RT i-type/RD r-type) selection signal
en5 : out std_logic;  --Exe registers enable
exe_func : out std_logic_vector(FUNC_SIZE-1 downto 0); --FUNC field for the EXE_CU
sel0 : out std_logic; --Mux (PC+1/NPC) selection signal
```

- the memory stage takes an enable signal for its registers, a memory read enable signal, a memory write enable signal, a control signal (ex_mem_write) equal to 1 if if the instruction writes on RF;

```
--MEMORY
ex_mem_write : out std_logic; --1 if the instruction writes on RF
en6 : out std_logic; --Memory registers enable
write_data_mem : out std_logic; --Memory write enable
read_data_mem : out std_logic; --Memory read enable
```

- the write back stage takes an enable signal for its registers, one enable signal for the write port of the register file, a selection signal for the ALU/MEM multiplexer, a control signal (mem_wb_write) equal to 1 if if the instruction writes on RF.

```
--WRITEBACK
en4 : out std_logic;  --PortA write enable
en7 : out std_logic;      --Writeback register enable
sel4 : out std_logic;  --Mux (ALU/MEM) selection signal
mem_wb_write : out std_logic --1 if the instruction writes on RF
```

The hardwired control unit handles jump and branches instruction through two finite state machines. The first one, designed for jump instructions, introduces two stalls into the pipeline as soon as a jump instruction is decoded. Each stall corresponds to a NOP instruction. The second one, designed for branch instructions, implements the “predict untaken” static prediction technique. The hardware is allowed to continue as if the branch was not executed; if the branch is taken, the fetch restarts at the branch target address. This causes all the instructions following the branch to stall two clock cycles.

# Implementation

The synthesis of both the data path and the control unit has been performed using Synopsis. The library is a 45 nm provided by ST Microelectronics.

## Datapath

After a first synthesis without constraints, these are the results about timing, area, power:

- Data arrival time = 1.47 ns;

- Combinational area = 5430.124023;
- Non combinational area = 4002.768083;
- Total cell area = 9432.892578;

- Cell Internal Power = 4.6093 mW (49%);
- Net Switching Power = 4.8194 mW (51%);
- Total Dynamic Power = 9.4287 mW;
- Cell Leakage Power = 192.7212 uW;

The final optimisation for frequency gives, instead, the following results:

- Data arrival time = 0.70 ns;

- Combinational area = 5685.218012;
- Non combinational area = 4002.768083;
- Total cell area = 9687.986328;

- Cell Internal Power = 4.6650 mW (49%);
- Net Switching Power = 4.7679 mW (51%);
- Total Dynamic Power = 9.4329 mW;
- Cell Leakage Power = 197.9602 uW;

## Control unit

The synthesis is executed without constraints and gives the following results:

- Data arrival time = 0.17 ns;

- Combinational area = 82.194000;
- Non combinational area = 45.219998;
- Total cell area = 127.414001;

- Cell Internal Power = 50.7479 uW (47%)
- Net Switching Power = 56.4464 uW (53%)
- Total Dynamic Power = 107.1943 uW;
- Cell Leakage Power = 2.8610 uW;

# Conclusions

After the optimisation phase, the max frequency for the data path is about 1.40 GHz. Considering that an ideal pipeline processor completes one instruction per clock cycle, the obtained throughput is $140 \cdot 10^7$ instructions per second (only considering the data path). The power and the total cell area are more or less the same in both the scenarios, before and after the optimisation phase.
Moving forward to the control unit, the result of the synthesis highlights that the power consumption is high and the dynamic power is an important percentage of it. This overhead is due to the complex logic that implements the FSMs and the control unit processes.
The DLX has been tested with some benchmarks, in order to check the right behaviour of all the instructions. Some of them are related to jump (both conditional and absolute), others to arithmetical instructions, bitwise and shift operations and memory instructions (load and store). In all the benchmarks, forwarding and hazards give correct results.
In this project, everything has been developed with a modular approach for future improvements. Possible improvements could be interrupts and exception handling, the execution of multi-cycle operations, the introduction of a cache and the introduction of FPRs and a floating point unit.

# References


- Microelectronic Systems (Lecture Notes), Mariagrazia Graziano
- Wikipedia, http://www.wikipedia.org/
- University of Washington Computer Science & Engineering - Machine Organization & Assembly Language, https://courses.cs.washington.edu/courses/cse378/07au/lectures/
- University of Texas at Austin CS352H - Computer Systems Architecture Fall 2009 Don Fussell, https://www.cs.utexas.edu/~fussell/courses/cs352h/lectures/9-MIPS-Pipeline-Hazards.pdf