TD 1

RISCV Architecture, Lexical Analysis and Grammars

1.1 The RISCV architecture

We give you the "RISCV cheat sheet". The objective is to recall concepts from the architecture course and manipulate the assembly code of the architecture you will compile to.

Your teaching assistant will make a demo of the RISCV simulator during this session.

EXERCISE #1 ▶ **TD**
On paper, write (in RISCV assembly language) a program which initializes the $R0$ register to 1 and increments it until it becomes equal to 8.

EXERCISE #2 ▶ **C to** RISCV- **Skip if you are late**
Translate into RISCV code the following C-code:

```
x=5;
if (x>12) y=70; else y=x+12;
```

1.2 Lexical Analysis

A bit of ANTLR4 syntax (for lexing rules) is given here: `https://github.com/antlr/antlr4/blob/master/doc/lexer-rules.md`. Essentially, rules are of the form:

```
rule_name: regular_expression { action };
-- rule_name lower case for lexer rules
-- action: piece of code in the host language (Python, ...)
```

EXERCISE #3 ▶ **Regular expressions for lexing**
Use the ANTLR4 syntax to define ANTLR4 macros to define:

1. Identifiers : any sequence of letters, digits and _ that does not begin by a digit nor _.
2. Floats like -3.96 (the sign is optional, but the dot is not).
3. Scientific notation like $-1.6E-12$.

EXERCISE #4 ▶ **Romans numbers**
Write an ANTLR4 lexical file that reads and interprets Roman numerals : $IV \rightarrow 4$ ...You can use the fact that the lexical analysis always takes the rule to match the longest subchain.

1.3 Grammars

All grammars will use the ANTLR4 syntax.

EXERCISE #5 ▶ **Well-founded parenthesis**
Write a grammar and files to accept any text with well-formed parenthesis ')' and '['.