TD 4

Register allocation and final code generation

In the following exercises we are looking for compiler independant optimisations (on the 3-address code). The goal here is to allocate registers with as few "spilled variables" as possible.

4.1 Register Allocation

EXERCISE #1 ▶ **Code production and register allocation**
Consider the expression $E = ((n * (n+1)) + (2 * n)$. We assume that we have:

- The variable n is stored in the stack slot referred as $[n]$ in the load (ld) instruction.

1. Generate a 3 address-code with temporaries and $rmem$ instruction to load n. Do it as blindly as possible (no temporary recycling).
2. (Without applying liveness analysis) Draw the liveness intervals. How many registers are sufficient to compute this expression?
3. Draw the interference graph (nodes are variables, edges are liveness conflicts).
4. Color this graph using the algorithm seen in the course (unbounded number of colors).
5. Give a register allocation with $K = 2$ registers, and rewrite code.

EXERCISE #2 ▶ **Register allocation, adapted from Exam, 2016**

We consider (in two columns) the following RISCV code. The tmp_i are temporaries to be allocated (in registers, in memory). For this exercise, we consider that we have two instructions that are capable to directly read/write at memory labels (`ld , sd`).

```
[...]
ld  tmp_1, label1
ld  tmp_2, label2
sub tmp_3, tmp_1, tmp_2
ld  tmp_4, label3
ld  tmp_5, label4
sub tmp_6, tmp_4, tmp_5
add tmp_7, tmp_6, 0
add tmp_8, tmp_3, tmp_7
sd  tmp_8, label5
ret
```

```
;;données/résultats (.dword = 8 bytes)
label1 : .dword 2
label2 : .dword 3
label3 : .dword -1
label4 : .dword 7
label5 : .dword 0
```

1. What is the computed expression? Where will it be stored?
2. Fill the following table with stars: put a star for a given temporary at a given line if and only if it is alive *at the entry of the instruction*. After the last store, all temporaries are supposed to be dead.

code	tmp_1	tmp_2	tmp_3	tmp_4	tmp_5	tmp_6	tmp_7	tmp_8
`ld tmp_1, label1`								
`ld tmp_2, label2`								
`sub tmp_3, tmp_1, tmp_2`								
`ld tmp_4, label3`								
`ld tmp_5, label4`								
`sub tmp_6, tmp_4, tmp_5`								
`add tmp_7, tmp_6, 0`								
`add tmp_8, tmp_3, tmp_7`								
`sd tmp_8, label5`								

3. Draw the interference graph.

4. Color the graph with the algorithm from the course with an infinite number of color (green, blue, red, black , ... in this order).

5. (We make as it we had only three available registers). We decide to spill the tmp_3 register and place it in memory.

   Generate the final code with two registers (t_3,t_4), sp and fp for the stack, t_0, t_1, t_2 for the spill management.