Exercise session 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:

- A multiplication instruction `mul t1,t2,t3` that computes t1 := t2*t3.
- The variable n is stored in the stack slot referred as $[n]$ in the load (rmem) 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 with three colors using the algorithm seen in the course.
5. Give a register allocation with $K = 2$ registers using the *iterative* register allocation algorithm seen in course.

We recall the following algorithm for final register allocation after coloring:

- For non-spilled variable: replace the temporary with its associated color/register.
- For a spilled variable (say, $temp5$ here, assigned to color 2):
  `ADD temp6 temp1 temp5`
  becomes (we use $R0$, $R1$, $R2$ to make load and stores for spilled variables):
  ```
  SUB R0, R6, 2
  RMEM R1, [R0]
  ADD alloc(temp6), alloc(temp1), R1
  ```
  where $alloc(temp1)$ denotes the allocation of $temp1$ (if it is a spilled variable, we have to first load its value in $R2$).

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

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

```
.set R6 spinit
[...]
ld  t1, label1
ld  t2, label2
sub t3, t1, t2
ld  t4, label3
ld  t5, label4
sub t6, t4, t5
add t7, t6, 0
add t8, t3, t7
st  t8, label5
jump 0
```

```
;;données/résultats                          ;;gestion de la pile
label1 : .word 2                             spinit:
label2 : .word 3                             .reserve 42
label3 : .word -1                            stackend: ;;  adresse du fond de la pile
label4 : .word 7
label5 : .reserve 1                          .END
```

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	t1	t2	t3	t4	t5	t6	t7	t8
`ld t1, label1`								
`ld t2, label2`								
`sub t3,t1,t2`								
`ld t4, label3`								
`ld t5, label4`								
`sub t6,t4,t5`								
`add t7, t6, 0`								
`add t8,t3,t7`								
`st t8, label5`								

3. Draw the interference graph.
4. Color the graph with the algorithm from the course with 3 colors (green, blue, red, in this order).
5. We decide to spill the t_3 register and place it in memory. Color the rest of the graph with 2 colors (green, blue).
6. Generate the final code with two registers (r_3,r_4), r_6 for the stack, r_0, r_1, r_2 for the spill management.