Exercise session 4

Liveness analysis, register allocation

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 Liveness analysis

First, reread the slides of the course with the help of your teaching assistant (slides 13 to 19):

`http://laure.gonnord.org/pro/teaching/MIF08_Compil1617/07-RegisterAlloc.pdf`

Let us recall the notations here: A variable at the left-hand side of an assignement is *killed* by the block. A variable that appears in this bloc is *generated*.

$$LV_{exit}(\ell) = \begin{cases} \emptyset & \text{if } \ell = \text{final} \\ \bigcup\{LV_{entry}(\ell') | (\ell', \ell) \in flow(G)\} \end{cases}$$

$$LV_{entry}(\ell) = \left(LV_{exit}(\ell) \backslash kill_{LV}(\ell)\right) \cup gen_{LV}(\ell)$$

The sets are initialised to $\emptyset$ and computed iteratively, until reaching a fixpoint.

EXERCISE #1 ▶ **Live variables**
Generate the CFG for the following program:

```
while d>0 then {
    a:=b+c;
    d:=d-b;
    e:=a+f;
    if e>0 then {
        f:=a+d;
        b:=d+f;
    }
    else{
        e:=a-c;
        }
    b:=a+c;
}
```

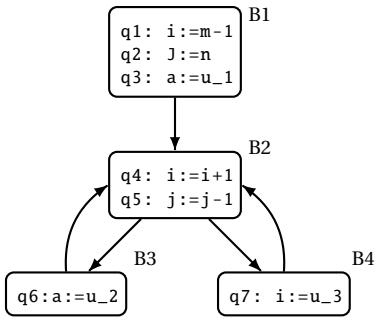
On this CFG:

- Compute Gen, $Kill$ for each block ℓ
- Compute $In(\ell) = LV_{entry}(\ell)$ and $Out(\ell) = LV_{exit}(\ell)$ iteratively.
- Suppress the dead code.

			Step 1		Step		Step		Step	
ℓ	$kill(\ell)$	$gen(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$

			Step		Step		Step		Step	
ℓ	$kill(\ell)$	$gen(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$

EXERCISE #2 ► **Live Variables**
After code generation, we obtain the following graph:



On this graph, perform liveness analysis and suppress the dead code.

			Step 1		Step		Step		Step	
ℓ	$kill(\ell)$	$gen(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$

			Step		Step		Step		Step	
ℓ	$kill(\ell)$	$gen(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$

4.2 Register Allocation

EXERCISE #3 ► **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.
- A "immediate load" instruction `ldi t1 4`.
- The variable n is stored in the stack slot referred as $[n]$ in the load instruction.

1. Generate a 3 address-code with temporaries and *LDR* 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 (`http://laure.gonnord.org/pro/teaching/MIF08_Compil1617/07-RegisterAlloc.pdf`, slides 27-30).
5. Give a register allocation with $K = 2$ registers using the *iterative* register allocation algorithm seen in course.