TD 3

3-address Code Generation

3.1 Code generation with temporaries

The code we generate will have an unbounded number of temporaries (`tmp0`, `tmp1`, ...) but actual RISCV instructions (`add`, `and`, ...) except for the conditional Jump.

The instruction set and documentation for the RISCV machine can be found in on the website (we already used it in previous sessions).

The code generation functions (see Section 3.3) have the following signatures:

`GenCodeExpr` : Expression → Code* × Temporary
`GenCodeSmt` : Statement → Code*

where $Code*$ is a sequence of 3-address instructions (RISCV with temporaries). As a side effect, while processing variable declarations, the code generation for statements might update a map `symbol_table` $Var \to \mathbb{N}$ (program variable to a temporary where to find its current value).

Auxiliary functions:

`new_tmp()` : → Temporary
`new_label()` : → Label

EXERCISE #1 ▶ **By hand!**
Using the code generation rules for the RISCV machine, generate the three-address RISCV code for the following (MiniC) program:

```
int main(){
    int a,n;

    n = 1;
    a = 7;
    while (n < a) {
      n = n+1;
    }

    return 0;
}
```

3.2 Language extensions

EXERCISE #2 ▶ **A new operator for expressions**
Write a code generation rule for the `xor` Boolean operator **without using the native** RISCV **operator.**

EXERCISE #3 ▶ **A new langage construct**
Write a code generation rule for the `repeat S until e` statement.

EXERCISE #4 ▶ **Lab 4**
If time allows, start lab 4 on code generation, you have all you need now :-).

3.3 Code Generation Rules

c	`dest <- new_tmp()`
	`code.add("li dest, c")`
	`return dest`

x	`# get the temporary associated to x.`
	`reg <- symbol_table[x]`
	`return reg`

e_1+e_2	`t1 <- GenCodeExpr(e_1)`
	`t2 <- GenCodeExpr(e_2)`
	`dest <- new_tmp()`
	`code.add("add dest, t1, t2")`
	`return dest`

e_1-e_2	`t1 <- GenCodeExpr(e_1)`
	`t2 <- GenCodeExpr(e_2)`
	`dest <- new_tmp()`
	`code.add("sub dest, t1, t2")`
	`return dest`

true	`dest <-new_tmp()`
	`code.add("li dest, 1")`
	`return dest`

$e_1 < e_2$	`dest <- new_tmp()`
	`t1 <- GenCodeExpr(e1)`
	`t2 <- GenCodeExpr(e2)`
	`endrel <- new_label()`
	`code.add("li dest, 0")`
	`# if t1>=t2 jump to endrel`
	`code.add("bge endrel, t1, t2")`
	`code.add("li dest, 1")`
	`code.addLabel(endrel)`
	`return dest`

Figure 3.1: 3@ Code generation for numerical or Boolean expressions

x = e	`dest <- GenCodeExpr(e)`
	`loc <- symbol_table[x]`
	`code.add("mv loc, dest")`

S1; S2	`# Just concatenate codes`
	`GenCodeSmt(S1)`
	`GenCodeSmt(S2)`

`if` *b* `then` *S*1 `else` *S*2	`lelse <- new_label()`
	`lendif <- new_label()`
	`t1 <- GenCodeExpr(b)`
	`#if the condition is false, jump to else`
	`code.add("beq lelse, t1, 0")`
	`GenCodeSmt(S1) # then`
	`code.add("j lendif")`
	`code.addLabel(lelse)`
	`GenCodeSmt(S2) # else`
	`code.addLabel(lendif)`

`while(` *b* `){` *S* `}`	`ltest <- new_label()`
	`lendwhile <- new_label()`
	`code.addLabel(ltest)`
	`t1 <- GenCodeExpr(b)`
	`code.add("beq lendwhile, t1, 0")`
	`GenCodeSmt(S) # execute S`
	`code.add("j ltest") # and jump to the test`
	`code.addLabel(lendwhile) # else it is done.`

Figure 3.2: 3@ Code generation for Statements