Exercise session 3

Typing + 3-address Code Generation

3.1 Typing

We recall (some of) the rules of the course for Typing:

$$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}} \qquad \frac{\Gamma(x) = t \quad t \in \{\texttt{int}, \texttt{bool}\}}{\Gamma \vdash x : t}$$

$$\frac{\Gamma \vdash e : t \quad \Gamma \vdash x : t \quad t \in \{\texttt{int}, \texttt{bool}\}}{\Gamma \vdash x := e : \texttt{void}} \qquad \frac{\Gamma \vdash b : \texttt{bool} \quad \Gamma \vdash S : \texttt{void}}{\Gamma \vdash \texttt{while}\ b\ \texttt{do}\ S\ \texttt{done} : \texttt{void}}$$

EXERCISE #1 ▶ **Well typed**
Type the mini-while program:

```
x1 := 3 ;
while (not x3) do
  x1 := x2 + 1 ;
  x3 := x3 and true
done
```

Under the following context: $\Gamma(x_1) = \Gamma(x_2) = \texttt{int}$, and $\Gamma(x_3) = \texttt{bool}$.

3.2 Code generation

The code we generate will have an unbounded number of temporaries (`tmp0`, `tmp1`, ...) but actual LC-3 instructions (`ADD`, `AND`, `BR`...). We recall the instruction set for the LC-3 machine:

syntaxe	**action**	**NZP**	**codage**															
			opcode				arguments											
			F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
NOT DR,SR	DR ← not SR	*	1	0	0	1	DR			SR			1	1	1	1	1	1
ADD DR,SR1,SR2	DR ← SR1 + SR2	*	0	0	0	1	DR			SR1			0	0	0	SR2		
ADD DR,SR1,Imm5	DR ← SR1 + SEXT(Imm5)	*	0	0	0	1	DR			SR1			1	Imm5				
AND DR,SR1,SR2	DR ← SR1 and SR2	*	0	1	0	1	DR			SR1			0	0	0	SR2		
AND DR,SR1,Imm5	DR ← SR1 and SEXT(Imm5)	*	0	1	0	1	DR			SR1			1	Imm5				
LEA DR,label	DR ← PC + SEXT(PCoffset9)	*	1	1	1	0	DR			PCoffset9								
LD DR,label	DR ← mem[PC + SEXT(PCoffset9)]	*	0	0	1	0	DR			PCoffset9								
ST SR,label	mem[PC + SEXT(PCoffset9)] ← SR		0	0	1	1	SR			PCoffset9								
LDR DR,BaseR,Offset6	DR ← mem[BaseR + SEXT(Offset6)]	*	0	1	1	0	DR			BaseR			Offset6					
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] ← SR		0	1	1	1	SR			BaseR			Offset6					
BR[n][z][p] label	Si (cond) PC ← PC + SEXT(PCoffset9)		0	0	0	0	n	z	p	PCoffset9								
NOP	No Operation		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RET (JMP R7)	PC ← R7		1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
JSR label	R7 ← PC; PC ← PC + SEXT(PCoffset11)		0	1	0	0	1	PCoffset11										

Table 3.1: LC3 simplified instruction set

The code generation functions have the following signatures:

`GenCodeExpr` : AExp → Code* × ℕ
`GenCodeSmt` : Inst → Code*

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

Auxiliary functions:
`newTemp()` : $\rightarrow \mathbb{N}$
`newLabel()` : $\rightarrow \mathbb{N}$

e ::= c	`#not valid if c is too big`
	`dr <-newTemp()`
	`code.add(InstructionAND(dr, dr, 0))`
	`code.add(InstructionADD(dr, dr, c))`
	`return dr`

e ::= x	`#get the place associated to x.`
	`regval=getTemp(x)`
	`return regval`

e ::= e_1+e_2	`t1 <- GenCodeExpr(e_1)`
	`t2 <- GenCodeExpr(e_2)`
	`dr <- newTemp()`
	`code.add(InstructionADD(dr, t1, t2))`
	`return dr`

e ::= e_1-e_2	`t1 <- GenCodeExpr(e_1)`
	`t2 <- GenCodeExpr(e_2)`
	`dr <- newTemp()`
	`code.add(InstructionNOT(dr, t2))`
	`code.add(InstructionADD(dr, dr, 1))`
	`code.add(InstructionADD(dr, dr, t1))`
	`return dr`

e ::= true	`dr <-newTemp()`
	`code.add(InstructionAND(dr, dr, 0))`
	`code.add(InstructionADD(dr, dr, 1))`
	`return dr`

e ::= $e_1 < e_2$	`dr <-newTemp()`
	`t1 <- GenCodeExpr (e1-e2)  #last write in register`
	`(lfalse,lend) <- newLabels()`
	`code.add(InstructionBRzp(lfalse))  #if =0 or >0 jump!`
	`code.add(InstructionAND(dr, dr, 0))`
	`code.add(InstructionADD(dr, dr, 1))  #dr <- true`
	`code.add(InstructionBR(lend))`
	`code.addLabel(lfalse)`
	`code.add(InstructionAND(dr, dr, 0))  #dr <- false`
	`code.addLabel(lend)`
	`return dr`

Figure 3.1: Code generation for expressions

x := e	`dr <- GenCodeExpr(e)`
	`#a code to compute e has been generated`
	`if x has a location loc:`
	`  code.add(instructionADD(loc,dr,0))`
	`else:`
	`  storeLocation(x,dr)`

S1; S2	`#concat codes`
	`GenCodeSmt(S1)`
	`GenCodeSmt(S2)`

`if` *b* `then` *S*1 `else` *S*2	`dr <-GenCodeExpr(b)  #dr is the last written register`
	`lfalse,lendif=newLabels()`
	`code.add(InstructionBRz(lfalse) #if 0 jump to execute S2`
	`GenCodeSmt(S1)  #else (execute S1`
	`code.add(InstructionBR(lendif)) #and jump to end)`
	`code.addLabel(lfalse)`
	`GenCodeSmt(S2)`
	`code.addLabel(lendif)`

`while` *b* `do` *S* `done`	`ltest,lendwhile=newLabels()`
	`code.addLabel(ltest)`
	`dr <-GenCodeExpr(b)  #dr is the last written register`
	`code.add(InstructionBRz(lendwhile) #if 0 jump to end`
	`GenCodeSmt(S)  #else (execute S`
	`code.add(InstructionBR(ltest))  #and jump to the test)`
	`code.addLabel(lendwhile)`

Figure 3.2: Code generation for Statements

EXERCISE #2 ▶ **By hand!**
Using the code generation rules of Figures 3.1 and 3.2, generate the three-address LC-3 code for the following (mini-while) program:

```
x1 := 3;
x2 := 7 + x1;
while (x2<x1) do
   x2 := x2-1;
done
```

EXERCISE #3 ▶ **A new operator for expressions**
Write a code generation rule for the `xor` boolean operator.

EXERCISE #4 ▶ **A new langage construction**
Write a code generation rule for the `repeat S until e` statement.