CAP - Exercises: (3-address) Code Generation (chapter 5)

Laure Gonnord

Oct. 2016

1 Rules for three address code generation

The code we generate will have an unbounded number of temporaries (`tmp0`, `tmp1`, ...) but actual LC-3 instructions (`ADD`, `AND`, `BR` ...).

The code generation functions have the following signatures (pseudo-code is given in Figures 1 and 2.) :

`GenCodeExpr` : AExp $\rightarrow$ Code$^*\times \mathbb{N}$
`GenCodeSmt` : Inst $\rightarrow$ 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 $Var \rightarrow \mathbb{N}$ (program variable to a temporary where to find its current value).

Auxiliary functions :

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

2 Exercises

EXERCISE ▶ **By hand !**
Using the code generation rules, generate the three-address code for the following (mini-while) program :

```
x1 = 3;
x2 = 7 + x1;
while not (x2<x1) do
   x2 = x2-1;
```

3 Language expansions

EXERCISE ▶ **Rules for boolean expressions**
Write a code generation rule for the `xor` boolean operator.

EXERCISE ▶ **Rules for statements**
Write a code generation rule for the `repeat S until e` statement.

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(InstructionAND(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 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` $S1$ `else` $S2$	`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`	`TODO!!`

FIGURE 2 – Code generation for Statements