Code Generation for Arithmetical Expressions

GCodeAExp(x)	=		$(\varepsilon$,x)
GCodeAExp(n)	=		$(\varepsilon$,n)
GCodeAExp(tab[i])	=	Let	
			t1=newTemp, t2=newTemp
		in	(t_1 := T*i ∥
			t_2 := tab[t_1],t_2)
GCodeAExp(tab[i,j])	=	Let	
			t1=newTemp(), t2=newTemp()
			t3=newTemp(), t4=newTemp()
			t5=newTemp()
		in	(t_1 := T*i ∥
			t_2 := N× T∥
			t_3 := t_2× j ∥
			t_4 := t_1+t_3 ∥
			t_5 := tab[t_4],t_5)
GCodeAExp(a_1 + a_2)	=	Let	(C_1,t_1)=GCodeAExp(a_1),
			(C_2,t_2)=GCodeAExp(a_2),
			t=newTemp
		in	(C_1∥C_2∥ t := t_1+t_2,t)

Code Generation for Boolean Expressions

GCodeBExp(a_1 < a_2, ltrue, lfalse)	=	Let	(C_1,t_1)=GCodeAExp(a_1),
			(C_2,t_2)=GCodeAExp(a_2),
		in	C_1∥C_2∥
			if t_1 < t_2
			goto ltrue ∥
			goto lfalse
GCodeBExp(b_1 ∧ b_2, ltrue, lfalse)	=	Let	l=newLabel()
		in	GCodeBExp(b_1, l, lfalse)∥
			l:∥
			GCodeBExp(b_2, ltrue, lfalse)
GCodeBExp(¬ b, ltrue, lfalse)	=		GCodeBExp(b, lfalse, ltrue)

Code Generation for Statements

Assignment and sequential composition

To each node of the abstract syntax tree, we associate code.

GCodeStm (x := a)	=	Let	(C,t)=GCodeAExp(a)
		in	C∥ x := t
GCodeStm(tab[i] := a)	=	Let	t1=newTemp,
			(C,t)=GCodeAExp(a)
		in	(t_1 := T*i ∥
			C ∥ tab[t_1] :=t)
GCodeStm (S_1 ; S_2)	=	Let	C_1 = GCodeStm(S_1),
			C_2 = GCodeStm(S_2)
		in	C_1 ∥ C_2

Code Generation for Statements

Iterative statement

GCodeStm (while b do S od)	=	Let	lbegin=newLabel(),
			ltrue=newLabel(),
			lfalse=newLabel()
		in	lbegin:∥
			GCodeBExp(b, ltrue, lfalse)∥
			ltrue:∥
			GCodeStm(S)∥
			goto lbegin∥
			lfalse:

Code Generation for Statements

Conditional statement

```
GCodeStm (if b then S₁ else S₂)  =  Let  lnext=newLabel(),
                                          ltrue=newLabel(),
                                          lfalse=newLabel()
                                     in   GCodeBExp(b,ltrue,lfalse)||
                                          ltrue:
                                          GCodeStm(S₁)||
                                          goto lnext ||
                                          lfalse:||
                                          GCodeStm(S₂)||
                                          lnext:
```

Summary - Intermediate-Code Generation

Intermediate-Code Generation

- From `While` to 3-address code.
- 3-address code = general-purpose representation of code:
  - easy to generate,
  - suitable for optimization,
  - easy to generate to assembly code.
- Ready for optimization!