

CS302 Languages and Compiler Design II

Lecture 6

Intermediate Languages

Why have one?

- Simplify compilation task by dividing into stages.
- Ease porting to new source or target languages.
- Ease implementation of code transformations.

Must bridge source and object code styles.

Source code is primarily **hierarchical**.

- expressions
- structured control statements
- (perhaps) local scoping

Object code is primarily **linear**.

- explicit intermediate values
- explicit labels and jumps
- flat name space

Source Code -> ?? -> Object Code

Intermediate language is a compromise

- maintain some hierarchy for ease of generating I.L.
- linearize somewhat for ease of generating object code.
- many possibilities.

Linear machine-like code

- expressions are linearized, with intermediate results in temporaries.
- use explicit labels & jumps
- flat address space
- for PCAT project, we'll use enriched SPARC assembler code.

I.R. trees

- expressions remain in tree form.
- use explicit labels & jumps
- locally-scoped temporaries.

Stack machine code

- expressions are linearized with intermediate results on stack
- use explicit labels & jumps

Specifying 3-address code

General form: three operands, one operator

X := Y op Z

Typical operators:

A := B

A := B op C

goto L

if0 A goto L

A := addr B

A := *B

Operands: named variables, temporaries, labels.

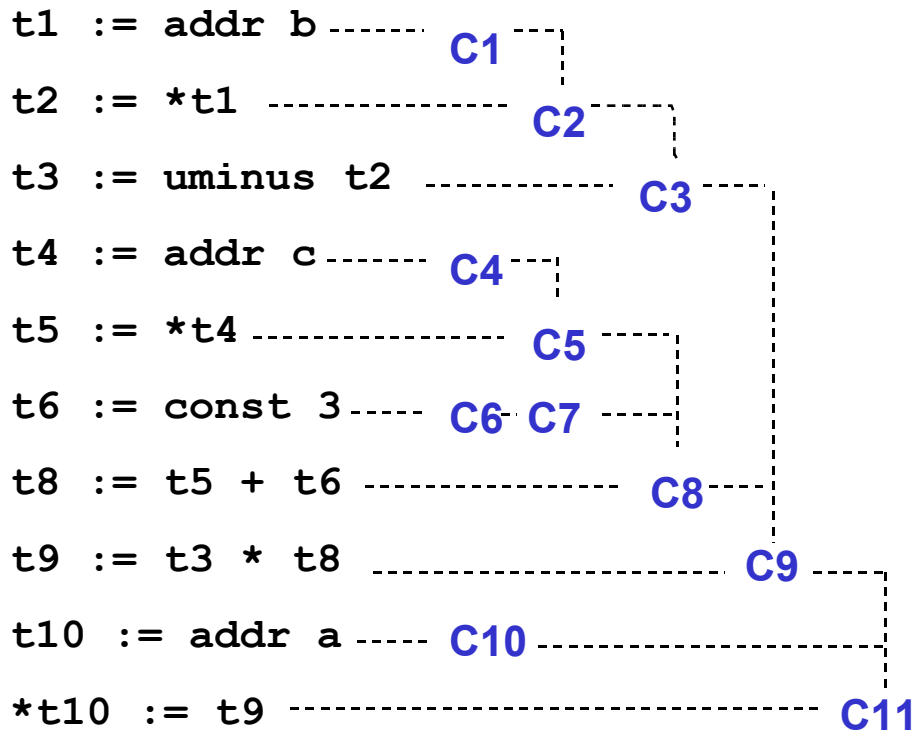
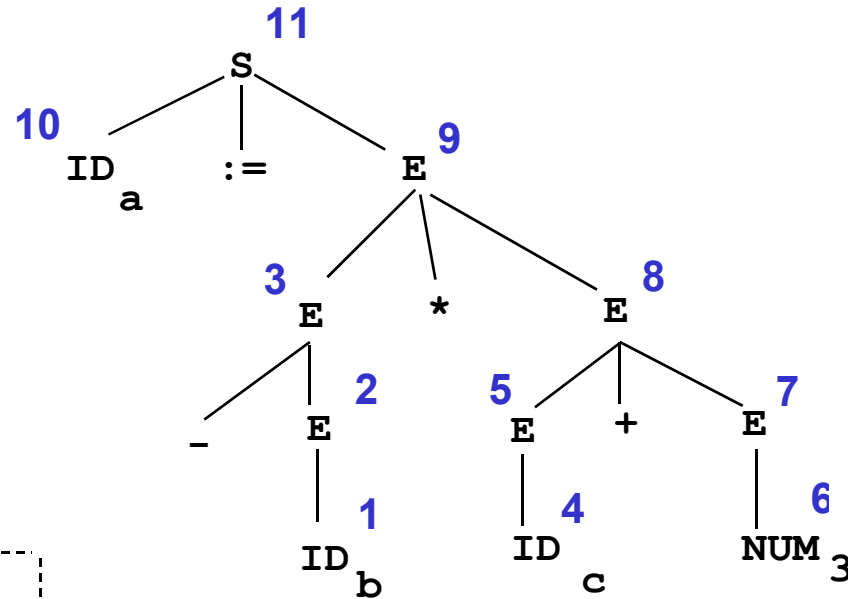
Assume an abstract instruction-generation function:

gen(result, operator, arg1, arg2)

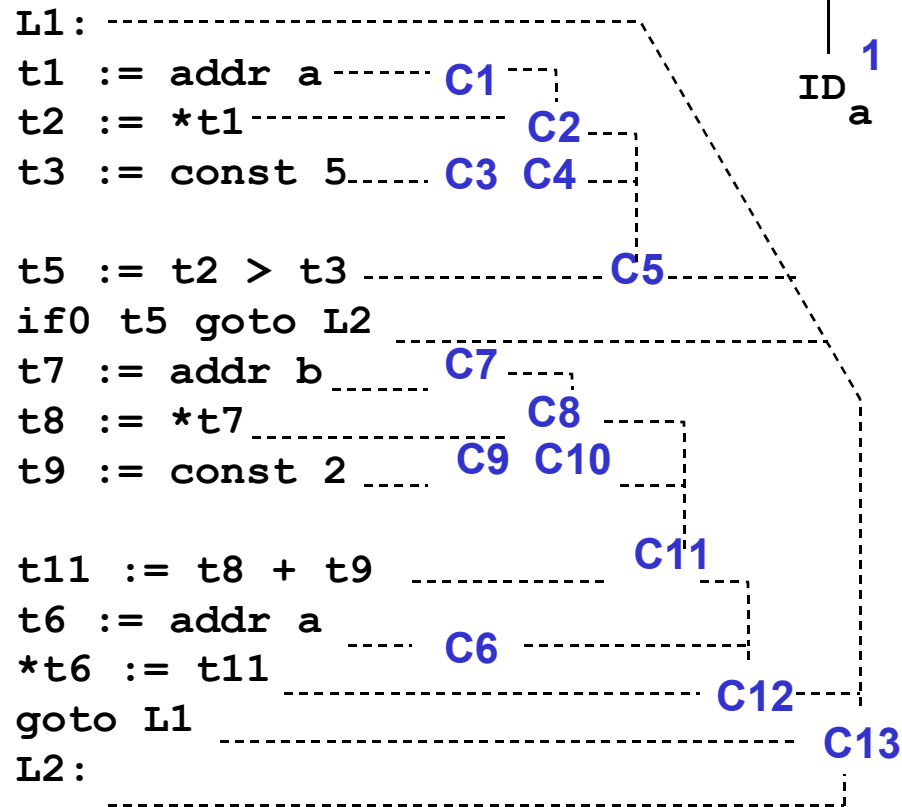
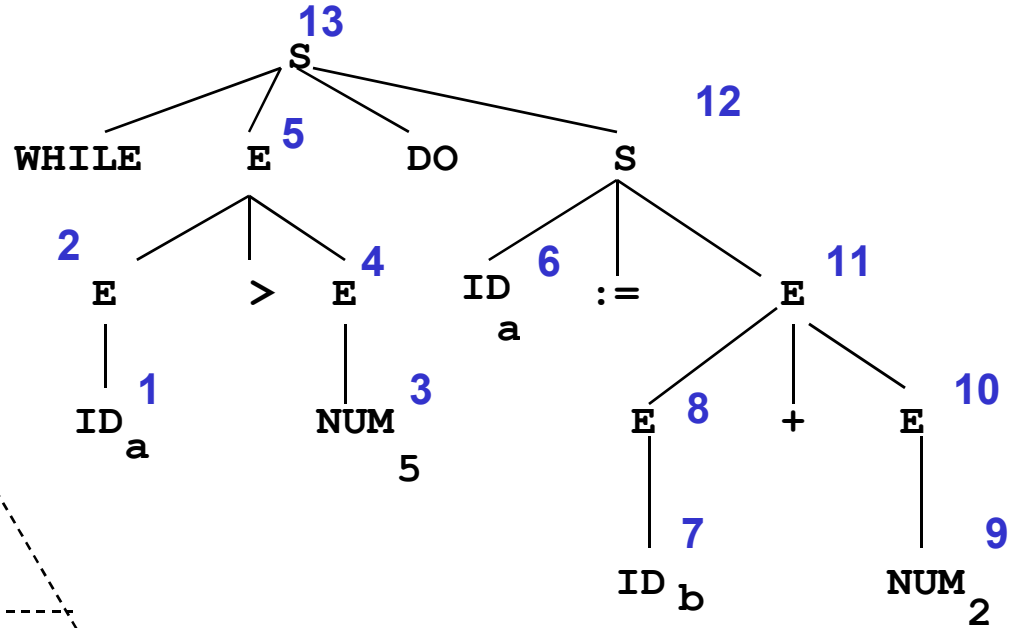
– can produce strings, “quads”, or whatever.

Quite ad-hoc: we’ll add new instructions when we need to.

Example: 3-Addr Code for $a := -b * (c + 3)$



Example: 3-Addr Code for while a > 5 do a := b + 2



Syntax-directed translation of Expressions

E := V	E.place = newtemp(); E.code = V.code @ [gen(E.place,*,V.place,_)]
E := N	E.place = N.place; E.code = N.code
E := E '+' E	E.place = newtemp(); E.code = E1.code @ E2.code @ [gen(E.place,+,E1.place,E2.place)]
E := '-' E	E.place = newtemp(); E.code = E1.code @ [gen(E.place,uminus,E1.place)]
E := E '>' E	E.place = newtemp(); E.code = E1.code @ E2.code @ [gen(E.place,>,E1.place,E2.place)]
V := VAR	V.place = newtemp(); V.code = [gen(V.place,addr,VAR.var,_)]
N := NUM	N.place = newtemp(); N.code = [gen(N.place,const,NUM.num,_)]

Contrast Syntax-directed Evaluation of Expressions

E := V E.val = *V.loc

E := N E.val = N.val

E := E '+' E E.val = E1.val + E2.val

E := '-' E E.val = - E1.val

E := E '>' E E.val = (E1.val > E2.val) ? 1 : 0

V := VAR V.loc = lookup(VAR.var)

N := NUM N.val = NUM.num

