Code Generation

- **Syntax Directed translation: Attribute *code* vs Side Effects**
- **The intermediate language: 3AC**
- **Code Generation for non control transfer code**

An Intermediate Language
3-address code - 3AC

1. (assignment)

x:= y op z

x:= op z

$$\frac{S \equiv <S\rho, S_M> \vdash loc(x) \rightarrow lx = S\rho(x) \quad S \vdash r(y) \rightarrow ry = S_M(S\rho(y)) \quad S \vdash r(z) \rightarrow rz \quad \vdash [op](ry,rz)=v}{S \vdash x:=y\ op\ z \rightarrow S[lx/v] = S_M(lx) <- v}$$

2. (copy)

x:=y

$$\frac{S \vdash loc(x) \rightarrow lx \quad S \vdash r(z) \rightarrow rz}{S \vdash x:=y \rightarrow S[lx/ry]}$$

3. (location names - values)

newtemp- a meta operator for fresh location names, e.g. newtemp:=-newtemp

values - scalar values of the meta prefixed by #, e.g. newtemp:=#3+newtemp

3-address code/2

4. (uncoditioned jump)

goto l

$$\frac{S \vdash code(l) \rightarrow P = S\rho(l) \qquad S \vdash P \rightarrow S'}{S \vdash goto(l) \,\|\, Ps \rightarrow S'}$$

5. (conditioned jump)

If x opr y goto l

$$\frac{S \vdash code(l) \rightarrow P \qquad S \vdash r(x) \rightarrow rx \qquad S \vdash r(y) \rightarrow ry \qquad \vdash [opr](rx,ry)=false \qquad S \vdash Ps \rightarrow S'}{S \vdash \text{if x opr y goto l} \,\|\, Ps \rightarrow S'}$$

?

3-address code/3

6. (i-structure)

x:=y[i]

x[i]:=y

$$\frac{\begin{array}{c} S \vdash loc(x) \rightarrow lx \\ S \vdash loc(y) \rightarrow ly \\ S \vdash r(i) \rightarrow ri \\ \vdash ly+ri = add \\ S \vdash S_M(add) \rightarrow lv \end{array}}{S \vdash x:=y[i] \rightarrow S_M(lx) \leftarrow lv}$$

complete with the other statement

3-address code/4

7. (pointer)

x:=&y
x:=*y
*x:=y

$$\frac{S \vdash loc(x) \rightarrow lx \quad S \vdash loc(y) \rightarrow ly}{S \vdash x:=\&y \rightarrow S_M(lx) \leftarrow ly}$$

$$\frac{S \vdash loc(x) \rightarrow lx \quad S \vdash r(y) \rightarrow ry \quad S \vdash S_M(ry) \rightarrow v}{S \vdash x:=*y \rightarrow S_M(lx) \leftarrow v}$$

$$\frac{S \vdash r(x) \rightarrow rx \quad S \vdash r(y) \rightarrow ry}{S \vdash *x:=y \rightarrow S_M(rx) \leftarrow rv}$$

where:

- **$S_M(l/r) = S_M(l) \leftarrow r$ = updade of cell l with value r**
- **. = location of the current statement of the program**
- **|| = code concatenation (sequencing)**

3-address code/5

8. (P-call)

param x1
param x2
…
param xn
call p n

$$\frac{S \vdash r(x) \rightarrow rx}{S \vdash \text{param } x \rightarrow S_M(.) \leftarrow rx}$$

$$\frac{S \vdash code(p) \rightarrow P \quad S \vdash P \rightarrow S',v \quad \langle S\rho, S_M'|_v \rangle \vdash Ps \rightarrow S''}{S \vdash \text{call } p\ n \parallel Ps \rightarrow S''}$$

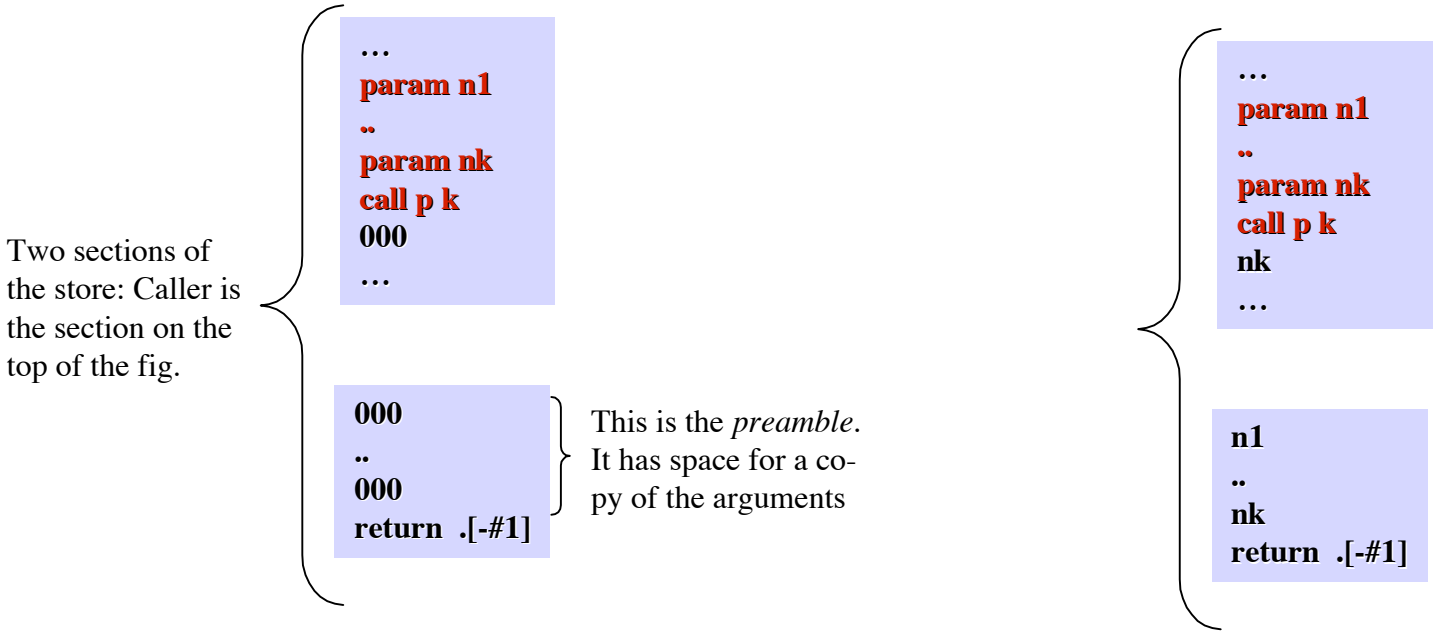
9. (call-ret)

return v

$$\frac{}{S \vdash \text{return } v \parallel ps \rightarrow S,v}$$

-The caller puts the arguments before the invocation and waits for a result in the word that is located immediately, following the invocation
-The callee has a copy of the arguments immediately before its first statement
-The return from the callee, puts the result immediately below the caller invocation statement
-The store is updated accordingly: $S_M'|_v$ is S_M' where the word following invocation is set to v

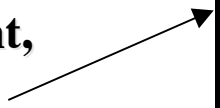
3-address code/6 caller-callee



Complete by giving a text that says what procedured p is supposed to compute.

3-address code
Defining a procedure for Factorial

procedure entry point,
i.e. address p is here



```
000
n = .[#-1]
rec = .+#3
if n≠ 0 goto rec
return #1
n1 = n-#1
param n1
call p #1
000
arg = .[-#3]
arg = arg +#1
val = .[-#3]
r = arg * val
return r
```

+ Use it in the computation of:
  call p 3
  where p is the address of the first statement;
+ Show the "activation records" (that the machine executor of 3AC is supposed to use) generated by the computation

Translation of Expressions in 3-address code (compositional)

x+y*3 *is translated into*: t1:= y.loc [*] #3
t2:= x.loc [+] t1

where:
t1 and t2 = 3AC locations
.loc = attribute for 3AC locations
[op] is the 3AC operation that corresponds to p

How does it to do it ?

$$\frac{S \vdash e1 \rightarrow (S,v1) \qquad S \vdash e2 \rightarrow (S,v2)}{S \vdash e1\ op\ e2 \rightarrow (S, op(v1,v2))}$$

(Semantics of expressions without side-effects)

$$\frac{\vdash e1 \Rightarrow ([\![e1]\!],l1) \qquad \vdash e2 \Rightarrow ([\![e2]\!],l2) \qquad l=newtemp()}{\vdash\ e1\ op\ e2 \Rightarrow ([\![e1]\!]\,||\,[\![e2]\!]\,||\,emit(l:=l1\ [op]\ l2), l)}$$

(Code Translation of exps without side-effects)

Meta:
newtemp: -> **loc** **-It is executed at compile time and furnishes a fresh 3AC location**
emit: string -> void **-It is executed at compile time and updates the output code file (called emit-file) by inserting, as the last line, the 3AC command, if any, whose textual represen tation is the argument of emit.**

Translation of Expressions in 3-address code - 2

Attributes:

loc:- location where the execution of the translated code, in the given state, will put the value of expression, in suc a state

-synthesized of any grammatical deriving expressions, or parts of them {E,E',F,F',T,num,ide}

Side-effect: the translated code is put in the emit-file

[15]E::= F {E'.in = F.loc;}
E' {E.loc:= E'.loc;}
[16]E'_1::= op-l F {l:=newtemp; emit(l":="E'_1.in [op-l] F.loc); E'_2.in:=l;}
E'_2 {E'_1.loc:= E'_2.loc;}
[17]E'::= ε {E'.loc = E'.in}
[18]F::= T {…}
F' {…}
[19]F'::= op-h T {…}
F' {…}
[20]F'::= ε {…}
[21]T::= num {T.loc = num.loc;}
[22]T::= ide {T.loc = ide.loc;}
[23]T::= (E) {T.loc = E.loc;}