

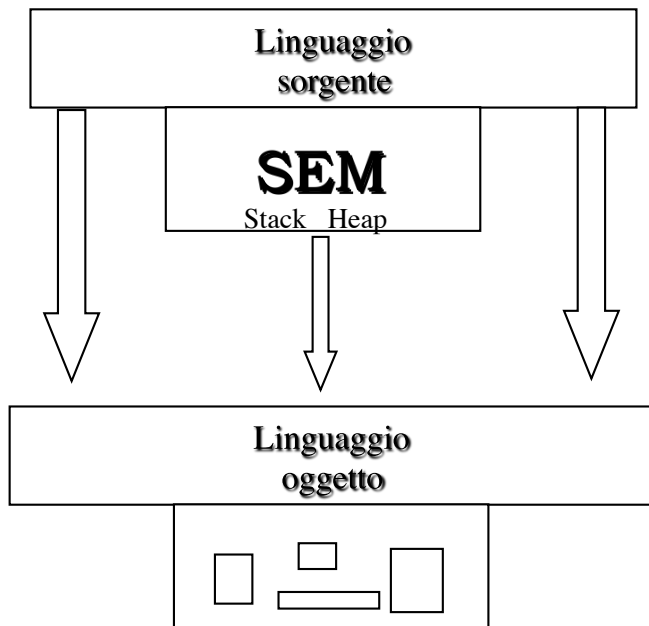
Generazione di codice

- Gerarchia di macchine: sorgente, target
- Traduzioni guidate dalla sintassi: l'attributo codice
- Generiamo codice per le espressioni su macchine a stack: codice postfixo {NO - 2000}
- Un linguaggio intermedio: codice a tre indirizzi
- Generazione di codice su L.I. - *parte 1*

Da strutture di una **Macchina Astratta** a
strutture equivalenti (semantica)
di una macchina (concreta)

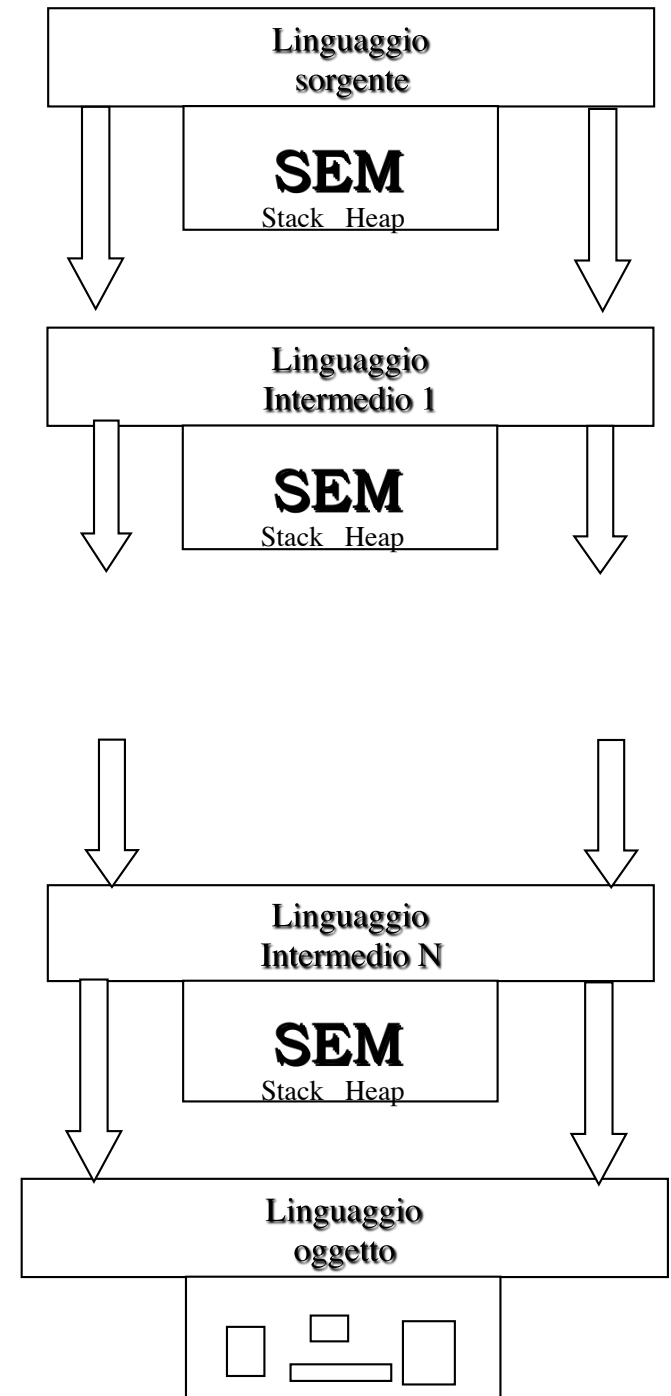
generazione di codice corrispondente

mediante una traduzione di strutture



Metodologia:

gerarchia di macchine alla cui base e' una macchina concreta dotata di un esecutore



TECNICA

ancora grammatiche ad attributi o SDD

codice generato = attributo

codice = valore di un'opportuna algebra

i.e.

opportune operazioni per:

manipolare e

comporre codice

un semplice e vantaggioso *codice* per le espressioni.

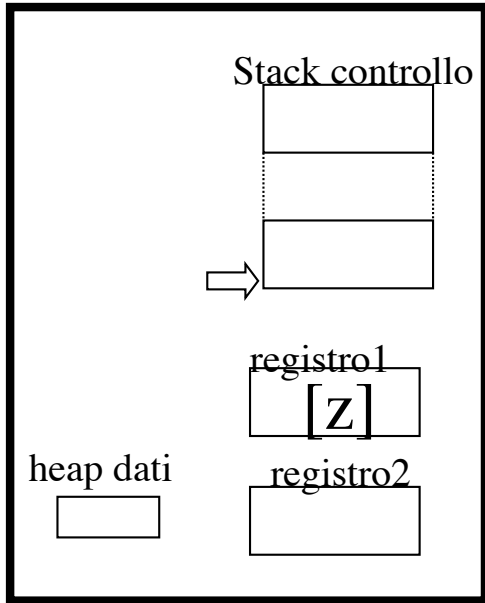
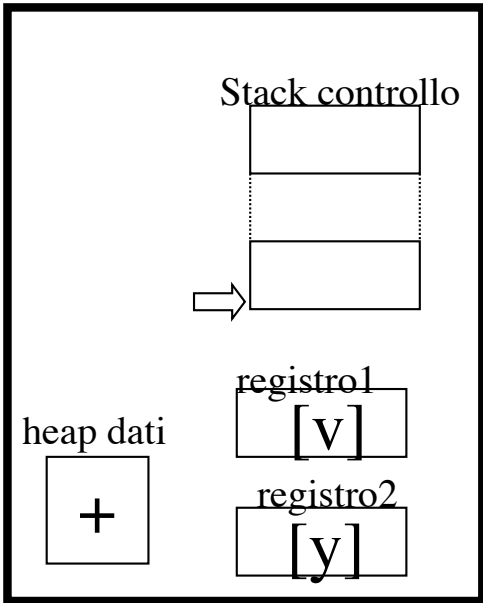
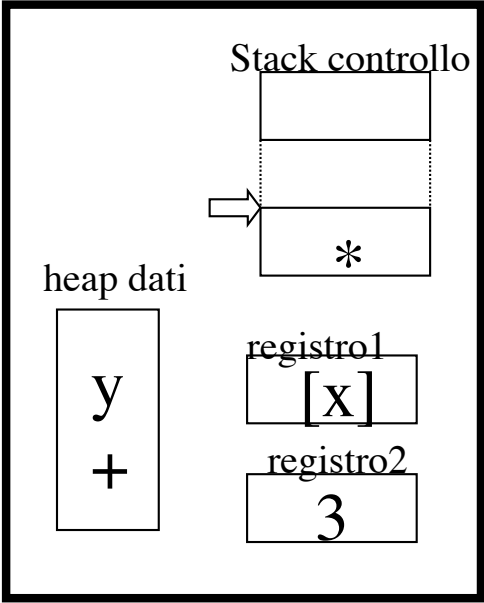
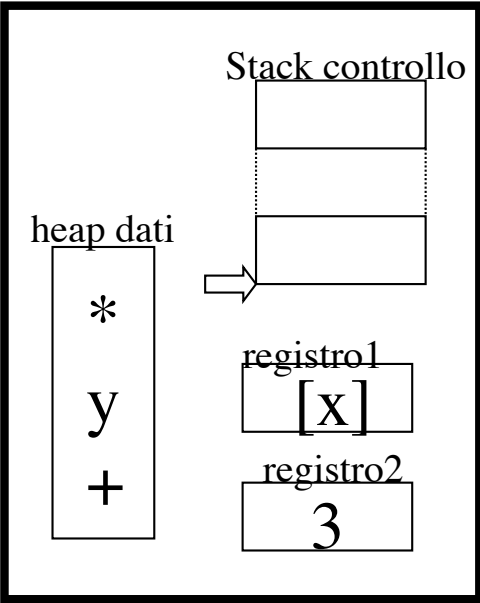
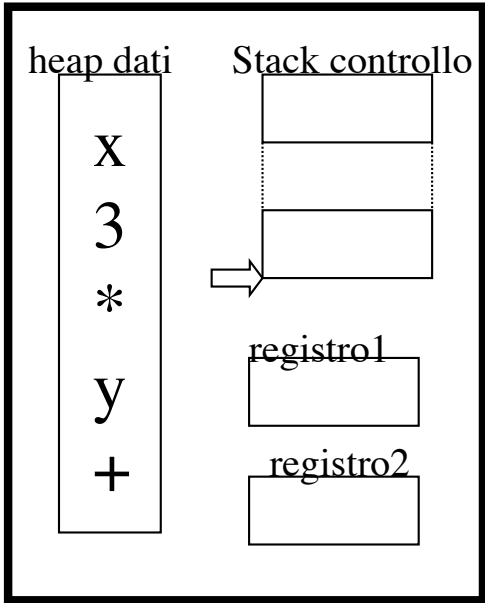
postfissa: opd1 opd2 opt

regole di inferenza:

$$\text{operando: } \frac{I=\text{opd seq} \quad S=T}{I:=\text{seq} \quad S:=\text{push}(\text{opd},T)}$$

$$\text{operatore:} \quad \frac{I=\text{opt seq} \quad S=T}{\text{binario} \quad \frac{[\text{opt}](\text{val}(\text{top}(S)), \text{val}(\text{top}(S)-1))=v}{\text{val}(\text{top}(S)-1):=v \quad S:=\text{pop}(T,1)}}$$

$(x*3)+y$ diventa $x\ 3\ * y\ +$



osserva:

- $[x]=\text{valore di } x$
- $[x]*[3]=v$
- $[v]+[y]=z$

Piano di traduzione

attributi: in= codice sottoespressione sin. (ereditato di E', F')
 code= codice espressione (sintetizzato di E, E', F, F', T)
 loc= locazione di memoria per variabili (sintetiz)
 operazioni: *= concatena (giustappone) codice postfisso

[0] P ::= Ds Cs	
[1] P ::= Cs	
[2] D ::= var ide O	?
[3] O₁ ::= , ide O₂	
[4] O ::= ε	
[5] Cs₁ ::= ; C Cs₂	
[6] Cs ::= ε	
[7] C ::= A	
[8] C ::= W	
[9] A ::= ide := E	
[10] W ::= while E do C endw	

Questa parte della grammatica non e' coinvolta nella traduzione

[11] $E ::= F E'$	$E'.in := F.code,$ $E.code := E'.code$
[12] $E'_1 ::= op-l F E'_2$	$E'_2.in := E'_1.in * F.code * op-l.value,$ $E'_1.code := E'_2.code$
[13] $E' ::= \epsilon$	$E'.code := E'.in$
[14] $F ::= T F'$	$F'.in := T.code,$ $F.code := F'.code$
[15] $F'_1 ::= op-h T F'_2$?
[16] $F' ::= \epsilon$	$F'.code := F'.in$
[17] $T ::= num$	$T.code := num.value$
[18] $T ::= ide$	$T.code := ide.loc$
[19] $T ::= (E)$	$T.code := E.code$

UN LINGUAGGIO INTERMEDIO

1. (assegnamento)

$$\begin{array}{l} x := y \text{ op } z \\ x := \text{op } y \end{array}$$

$$\begin{array}{l} S \vdash \langle S\rho, SM \rangle \vdash \text{loc}(x) \rightarrow lx = S\rho(x) \\ S \vdash r(y) \rightarrow ry = SM(S\rho(y)) \\ S \vdash r(z) \rightarrow rz \\ \hline \vdash [\text{op}](ry, rz) = v \\ \hline S \vdash x := y \text{ op } z \rightarrow S[lx/v] = SM(lx) \langle -v \end{array}$$

2. (copy)

$$x := y$$

$$\begin{array}{l} S \vdash \text{loc}(x) \rightarrow lx \\ S \vdash r(y) \rightarrow ry \\ \hline S \vdash x := y \rightarrow S[lx/ry] \end{array}$$

(indirizzamento immediato)

$$\dots \#y$$

$$S \vdash \text{Loc}(\#y) \rightarrow \text{undef}$$

$$S \vdash r(\#y) \rightarrow y$$

3. (u-jump)

GoTo l

$$S \vdash \text{code}(l) \rightarrow P = S\rho(l)$$
$$S \vdash P \rightarrow S'$$

$$S \vdash \text{goto } l \mid P_s \rightarrow S'$$

4. (c-jump)

if x rel y GoTo l

$$S \vdash \text{code}(l) \rightarrow P$$
$$S \vdash r(x) \rightarrow rx$$
$$S \vdash r(y) \rightarrow ry$$
$$\vdash [\text{rel}](rx, ry) = \text{false}$$
$$S \vdash P_s \rightarrow S'$$

$$S \vdash \text{if } x \text{ rel } y \text{ goto } l \mid P_s \rightarrow S'$$

?

5. (P-call)

param x1
 param x2
 ...
 param xn
 call p,n

$$\frac{S \vdash r(x) \rightarrow rx}{S \vdash \text{param } x \rightarrow S[./rx]}$$

$$\frac{S \vdash \text{code}(p) \rightarrow P \quad S \vdash P \rightarrow S'}{S \vdash \text{call } p, n \rightarrow S|_M S' = \langle S\rho, S'M(S\rho) \rangle}$$

7. (i-structure)

x := y[i]
 x[i] := y

$$\frac{\begin{array}{l} S \vdash \text{loc}(x) \rightarrow lx \\ S \vdash \text{loc}(y) \rightarrow ly \\ S \vdash r(i) \rightarrow ri \\ \quad \vdash ly + ri = \text{add} \\ S \vdash SM(\text{add}) = v \end{array}}{S \vdash x := y[i] \rightarrow S[lx/v]}$$

?

8. (pointer)

$x := \&y$ $x := *y$ $*x := y$

$\frac{S \vdash \text{loc}(x) \rightarrow lx \quad S \vdash \text{loc}(y) \rightarrow ly}{S \vdash x := \&y \rightarrow S[lx/ly]}$
$\frac{S \vdash \text{loc}(x) \rightarrow lx \quad S \vdash r(y) \rightarrow ry \quad S \vdash SM(ry) \rightarrow v}{S \vdash x := *y \rightarrow S[lx/v]}$
$\frac{S \vdash r(x) \rightarrow rx \quad S \vdash r(y) \rightarrow ry}{S \vdash *x := y \rightarrow S[rx/ry]}$

significato dei simboli:

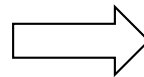
- l/r = update di contenuto di l con r
- . = locazione corrente stm nel prog.
- || = concatenazione di codice

Traduzione delle espressioni di Semplice nel linguaggio intermedio

$x+y*3$ sarà tradotto in: $t1:= y.loc * \#3$
 $t2:= x.loc + t1$

Dove:

$t1$ e $t2$ = locazioni del
linguaggio intermedio
 $.loc$ = attributo il cui valore
è una locazione di L.I.
 $:=$ è un comando di L.I.



Operazioni nel meta:

newtemp: -> **loc**
gen: *expr* -> **cmd**
//: **cmd cmd** -> **cmd**

Piano di traduzione

code= sintetizzato per il codice prodotto

loc= sintetizzato per la *locazione*

[ogni sottoespressione calcola la
locazione ove è posto il risultato]

in= ereditato per la *locazione* della
sottoespressione sinistra

Aggiungiamo la traduzione del comando di assegnamento

[0] P::= Ds Cs	
[1] P::= Cs	
[2] D::= var ide O [3] O₁::= , ide O₂ [4] O::=ε	?
[5] Cs₁::= ; C Cs₂	
[6] Cs::= ε	
[7] C::= A	
[8] C::= W	
[9] A::= ide := E	A.code:=E.codellgen(ide.loc ':=' E.loc)
[10] W::= while E do C endw	

Questa parte della grammatica
non è coinvolta nella traduzione

[11] $\mathbf{E} ::= \mathbf{F E}'$	
[12] $\mathbf{E}'_1 ::= \mathbf{op-l F E}'_2$ $\mathbf{E}'_1.\text{code} := \mathbf{F}.\text{code} \parallel \mathbf{g}$	
[13] $\mathbf{E}' ::= \epsilon$	
[14] $\mathbf{F} ::= \mathbf{T F}'$	
[15] $\mathbf{F}'_1 ::= \mathbf{op-h T F}'_2$ $\mathbf{F}'_1.\text{code} := \mathbf{T}.\text{code} \parallel \mathbf{g}$	
[16] $\mathbf{F}' ::= \epsilon$	
[17] $\mathbf{T} ::= \mathbf{num}$	
[18] $\mathbf{T} ::= \mathbf{ide}$	
[19] $\mathbf{T} ::= (\mathbf{E})$	

[11] $\mathbf{E} ::= \mathbf{F E}'$	$\mathbf{E}.loc := \mathbf{E}'.loc; \mathbf{E}'.in := \mathbf{F}.loc,$ $\mathbf{E}.code := (\mathbf{F}.code \parallel \mathbf{E}'.code)$
[12] $\mathbf{E}'_1 ::= \mathbf{op-l F E}'_2$ $\mathbf{E}'_1.code := \mathbf{F}.code \parallel g$	
[13] $\mathbf{E}' ::= \epsilon$	
[14] $\mathbf{F} ::= \mathbf{T F}'$	
[15] $\mathbf{F}'_1 ::= \mathbf{op-h T F}'_2$ $\mathbf{F}'_1.code := \mathbf{T}.code \parallel g$	
[16] $\mathbf{F}' ::= \epsilon$	
[17] $\mathbf{T} ::= \mathbf{num}$	
[18] $\mathbf{T} ::= \mathbf{ide}$	
[19] $\mathbf{T} ::= (\mathbf{E})$	

[11] $\mathbf{E} ::= \mathbf{F E}'$	$\mathbf{E}.loc := \mathbf{E}'.loc; \mathbf{E}'.in := \mathbf{F}.loc,$ $\mathbf{E}.code := (\mathbf{F}.code \parallel \mathbf{E}'.code)$
[12] $\mathbf{E}'_1 ::= \mathbf{op-l F E}'_2$	$t := \text{newtemp}, \mathbf{E}'_2.in := t, \mathbf{E}'_1.loc := \mathbf{E}'_2.loc$ $\mathbf{E}'_1.code := \mathbf{F}.code \parallel \text{gen}(t := \mathbf{E}'_1.in \mathbf{op-l.value F}.loc) \parallel \mathbf{E}'_2.code$
[13] $\mathbf{E}' ::= \epsilon$	
[14] $\mathbf{F} ::= \mathbf{T F}'$	
[15] $\mathbf{F}'_1 ::= \mathbf{op-h T F}'_2$	
$\mathbf{F}'_1.code := \mathbf{T}.code \parallel g$	
[16] $\mathbf{F}' ::= \epsilon$	
[17] $\mathbf{T} ::= \mathbf{num}$	
[18] $\mathbf{T} ::= \mathbf{ide}$	
[19] $\mathbf{T} ::= (\mathbf{E})$	

[11] $\mathbf{E} ::= \mathbf{F E}'$	$\mathbf{E}.loc := \mathbf{E}'.loc; \mathbf{E}'.in := \mathbf{F}.loc,$ $\mathbf{E}.code := (\mathbf{F}.code \parallel \mathbf{E}'.code)$
[12] $\mathbf{E}'_1 ::= \mathbf{op-l F E}'_2$ $\mathbf{E}'_1.code := \mathbf{F}.code \parallel \mathbf{gen}(t := \mathbf{E}'_1.in \mathbf{op-l.value F}.loc) \parallel \mathbf{E}'_2.code$	$t := \text{newtemp}, \mathbf{E}'_2.in := t, \mathbf{E}'_1.loc := \mathbf{E}'_2.loc$
[13] $\mathbf{E}' ::= \epsilon$	$\mathbf{E}.code := \text{nop}, \mathbf{E}'.loc := \mathbf{E}'.in$
[14] $\mathbf{F} ::= \mathbf{T F}'$	
[15] $\mathbf{F}'_1 ::= \mathbf{op-h T F}'_2$ $\mathbf{F}'_1.code := \mathbf{T}.code \parallel \mathbf{g}$	
[16] $\mathbf{F}' ::= \epsilon$	
[17] $\mathbf{T} ::= \mathbf{num}$	
[18] $\mathbf{T} ::= \mathbf{ide}$	
[19] $\mathbf{T} ::= (\mathbf{E})$	

[11] $\mathbf{E} ::= \mathbf{F E}'$	$\mathbf{E}.loc := \mathbf{E}'.loc; \mathbf{E}'.in := \mathbf{F}.loc,$ $\mathbf{E}.code := (\mathbf{F}.code \parallel \mathbf{E}'.code)$
[12] $\mathbf{E}'_1 ::= \mathbf{op-l F E}'_2$ $\mathbf{E}'_1.code := \mathbf{F}.code \parallel \mathbf{gen}(t := \mathbf{E}'_1.in \mathbf{op-l}.value \mathbf{F}.loc) \parallel \mathbf{E}'_2.code$	$t := \text{newtemp}, \mathbf{E}'_2.in := t, \mathbf{E}'_1.loc := \mathbf{E}'_2.loc$
[13] $\mathbf{E}' ::= \varepsilon$	$\mathbf{E}.code := \text{nop}, \mathbf{E}'.loc := \mathbf{E}'.in$
[14] $\mathbf{F} ::= \mathbf{T F}'$	$\mathbf{F}.loc := \mathbf{F}'.loc; \mathbf{F}'.in := \mathbf{T}.loc,$ $\mathbf{F}.code := (\mathbf{T}.code \parallel \mathbf{F}'.code)$
[15] $\mathbf{F}'_1 ::= \mathbf{op-h T F}'_2$ $\mathbf{F}'_1.code := \mathbf{T}.code \parallel \mathbf{g}$	
[16] $\mathbf{F}' ::= \varepsilon$	
[17] $\mathbf{T} ::= \mathbf{num}$	
[18] $\mathbf{T} ::= \mathbf{ide}$	
[19] $\mathbf{T} ::= (\mathbf{E})$	

[11] $\mathbf{E} ::= \mathbf{F E}'$	$\mathbf{E}.loc := \mathbf{E}'.loc; \mathbf{E}'.in := \mathbf{F}.loc,$ $\mathbf{E}.code := (\mathbf{F}.code \parallel \mathbf{E}'.code)$
[12] $\mathbf{E}'_1 ::= \mathbf{op-l F E}'_2$	$t := \text{newtemp}, \mathbf{E}'_2.in := t, \mathbf{E}'_1.loc := \mathbf{E}'_2.loc$ $\mathbf{E}'_1.code := \mathbf{F}.code \parallel \text{gen}(t := \mathbf{E}'_1.in \mathbf{op-l.value F}.loc) \parallel \mathbf{E}'_2.code$
[13] $\mathbf{E}' ::= \epsilon$	$\mathbf{E}.code := \text{nop}, \mathbf{E}'.loc := \mathbf{E}'.in$
[14] $\mathbf{F} ::= \mathbf{T F}'$	$\mathbf{F}.loc := \mathbf{F}'.loc; \mathbf{F}'.in := \mathbf{T}.loc,$ $\mathbf{F}.code := (\mathbf{T}.code \parallel \mathbf{F}'.code)$
[15] $\mathbf{F}'_1 ::= \mathbf{op-h T F}'_2$	$t := \text{newtemp}, \mathbf{F}'_2.in := t, \mathbf{F}'_1.loc := \mathbf{F}'_2.loc$ $\mathbf{F}'_1.code := \mathbf{T}.code \parallel \text{gen}(t := \mathbf{F}'_1.in \mathbf{op-l.value T}.loc) \parallel \mathbf{F}'_2.code$
[16] $\mathbf{F}' ::= \epsilon$	$\mathbf{F}'.code := \text{nop}, \mathbf{F}'.loc := \mathbf{F}'.in$
[17] $\mathbf{T} ::= \mathbf{num}$	
[18] $\mathbf{T} ::= \mathbf{ide}$	
[19] $\mathbf{T} ::= (\mathbf{E})$	

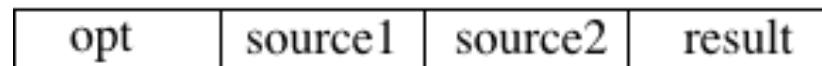
[11] $\mathbf{E} ::= \mathbf{F E}'$	$\mathbf{E}.loc := \mathbf{E}'.loc; \mathbf{E}'.in := \mathbf{F}.loc,$ $\mathbf{E}.code := (\mathbf{F}.code \parallel \mathbf{E}'.code)$
[12] $\mathbf{E}'_1 ::= \mathbf{op-l F E}'_2$ $\mathbf{E}'_1.code := \mathbf{F}.code \parallel \text{gen}(t := \mathbf{E}'_1.in \mathbf{op-l.value F}.loc) \parallel \mathbf{E}'_2.code$	$t := \text{newtemp}, \mathbf{E}'_2.in := t, \mathbf{E}'_1.loc := \mathbf{E}'_2.loc$
[13] $\mathbf{E}' ::= \epsilon$	$\mathbf{E}.code := \text{nop}, \mathbf{E}'.loc := \mathbf{E}'.in$
[14] $\mathbf{F} ::= \mathbf{T F}'$	$\mathbf{F}.loc := \mathbf{F}'.loc; \mathbf{F}'.in := \mathbf{T}.loc,$ $\mathbf{F}.code := (\mathbf{T}.code \parallel \mathbf{F}'.code)$
[15] $\mathbf{F}'_1 ::= \mathbf{op-h T F}'_2$ $\mathbf{F}'_1.code := \mathbf{T}.code \parallel \text{gen}(t := \mathbf{F}'_1.in \mathbf{op-l.value T}.loc) \parallel \mathbf{F}'_2.code$	$t := \text{newtemp}, \mathbf{F}'_2.in := t, \mathbf{F}'_1.loc := \mathbf{F}'_2.loc$
[16] $\mathbf{F}' ::= \epsilon$	$\mathbf{F}'.code := \text{nop}, \mathbf{F}'.loc := \mathbf{F}'.in$
[17] $\mathbf{T} ::= \mathbf{num}$	$\mathbf{T}.code := \text{nop}, \mathbf{T}.loc := \mathbf{num.value}$
[18] $\mathbf{T} ::= \mathbf{ide}$	
[19] $\mathbf{T} ::= (\mathbf{E})$	

[11] $E ::= F E'$	$E.loc := E'.loc; E'.in := F.loc,$ $E.code := (F.code \parallel E'.code)$
[12] $E'1 ::= op-l F E'2$	$t := newtemp, E'2.in := t, E'1.loc := E'2.loc$ $E'1.code := F.code \parallel gen(t := E'1.in \text{ op-l.value } F.loc) \parallel E'2.code$
[13] $E' ::= \epsilon$	$E.code := nop, E'.loc := E'.in$
[14] $F ::= T F'$	$F.loc := F'.loc; F'.in := T.loc,$ $F.code := (T.code \parallel F'.code)$
[15] $F'1 ::= op-h T F'2$	$t := newtemp, F'2.in := t, F'1.loc := F'2.loc$ $F'1.code := T.code \parallel gen(t := F'1.in \text{ op-l.value } T.loc) \parallel F'2.code$
[16] $F' ::= \epsilon$	$F'.code := nop, F'.loc := F'.in$
[17] $T ::= num$	$T.code := nop, T.loc := num.value$
[18] $T ::= ide$	$T.code := nop, T.loc := ide.loc$
[19] $T ::= (E)$	

[11] $E ::= F E'$	$E.loc := E'.loc; E'.in := F.loc,$ $E.code := (F.code \parallel E'.code)$
[12] $E'1 ::= op-l F E'2$	$t := newtemp, E'2.in := t, E'1.loc := E'2.loc$ $E'1.code := F.code \parallel gen(t := E'1.in \text{ op-l.value } F.loc) \parallel E'2.code$
[13] $E' ::= \epsilon$	$E.code := nop, E'.loc := E'.in$
[14] $F ::= T F'$	$F.loc := F'.loc; F'.in := T.loc,$ $F.code := (T.code \parallel F'.code)$
[15] $F'1 ::= op-h T F'2$	$t := newtemp, F'2.in := t, F'1.loc := F'2.loc$ $F'1.code := T.code \parallel gen(t := F'1.in \text{ op-l.value } T.loc) \parallel F'2.code$
[16] $F' ::= \epsilon$	$F'.code := nop, F'.loc := F'.in$
[17] $T ::= num$	$T.code := nop, T.loc := num.value$
[18] $T ::= ide$	$T.code := nop, T.loc := ide.loc$
[19] $T ::= (E)$	$T.code := E.code, T.loc := E.loc$

come risulta il codice generato per
E.code allorche'
E derivi la frase $(x+2)*(y-x)$

il linguaggio target e' a 3 indirizzi
perche' ogni stm
e' rappresentabile con una word=32
bit secondo
il formato:



operandi=locazione
+
mod. indiriz.

:=/+	x.loc	2	t1
:=/-	y.loc	x.loc	t2
:=/*	t1	t2	t3

Osserva:

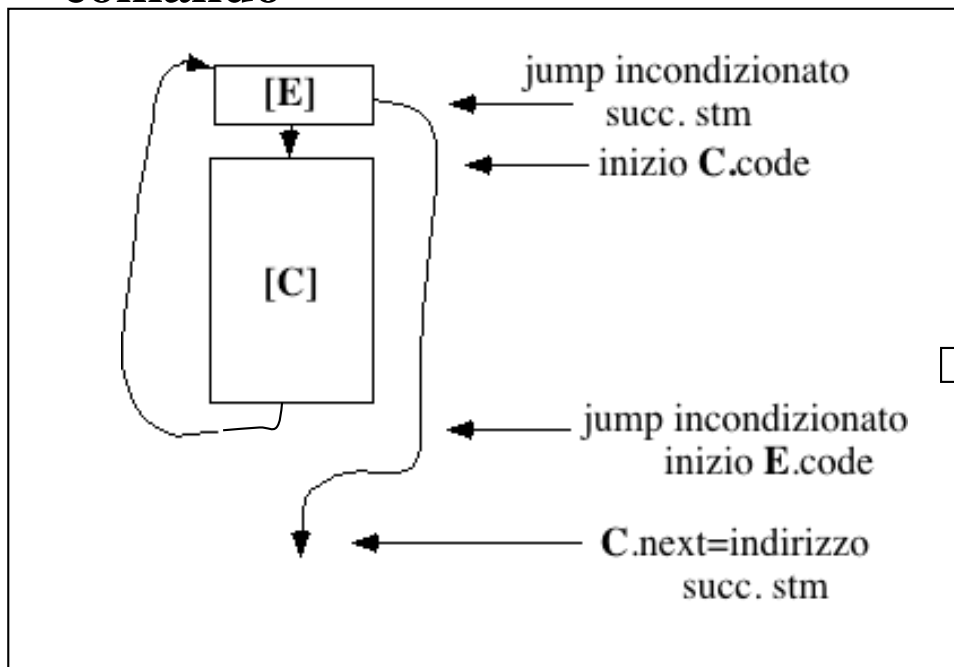
il linguaggio a tre indirizzi prevede
assegnamenti differenti per differenti
operatori di espressioni

Traduciamo l'iterazione di Semplice

[10]W ::= while E do C endw

Schema di
Generazione di Codice

Schema del
Controllo del
comando



[E]
If E.loc=false goto ?
[C]
Goto ??

Operazioni nel meta
indirizzo del codice
cui trasferire controllo

Piano di traduzione

Code = sintetizzato per il codice prodotto

Loc = sintetizzato per la locazione

[ogni sottoespressione calcola la
locazione ove e' posto il risultato]

begin,next = sintetizzati per (etichetta) locazione
inizio codice
trasferimento controllo finale

[10]**W::= while E do C endw**

```
W.begin:=newtemp,W.next:=newtemp,  
W.code:= gen(W.begin ':')||  
    E.codell  
    gen('if E.loc '=' 'false' 'goto' W.next)||  
    C.codellgen('goto'W.begin)  
gen(W.next ':')
```

Traduciamo la sequenzializzazione di Semplice

[5] $Cs1 ::= ; C Cs2$

$Cs1.code := C.code \parallel Cs2.code$

Come risulta il codice generato per C.code allorché C derivi la frase:

```
while x=0
  do y:=(x+2)*(y-x);
  x:=x-y endw
```

L1:	:=/=	x.loc	0	t1
	if/=	t1	false	L2
	:=/+	x.loc	2	t2
	:=/-	y.loc	x.loc	t3
	:=/*	t2	t3	y.loc
	:=/-	x.loc	y.loc	x.loc
	goto	L1		
L2:	?	?	?	?

Questo codice e' codice del nostro linguaggio
intermedio???

sono previsti stm con label???