



UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche,
Informatiche e Matematiche

12. Terzo Assignment: LICM

Compilatori – Middle end [I215-014]

Corso di Laurea in INFORMATICA
(D.M.270/04) [16-215]
Anno accademico 2024/2025

Prof. Andrea Marongiu
andrea.marongiu@unimore.it

Copyright note

È vietata la copia e la riproduzione dei contenuti e immagini in qualsiasi forma.

È inoltre vietata la redistribuzione e la pubblicazione dei contenuti e immagini non autorizzata espressamente dall'autore o dall'Università di Modena e Reggio Emilia.

Credits

- Cooper, Torczon, “Engineering a Compiler”, Elsevier
- Sampson, Cornell University, “Advanced Compilers”
- Gibbons, Carnegie Mellon University, “Optimizing Compilers”
- Pekhimenko, University of Toronto, “Compiler Optimization”



UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche,
Informatiche e Matematiche

Loop-Invariant Code Motion

Loop-Invariant Code Motion

```
a = ...;  
b = ...;  
for (i = 0; i < 100; i++)  
    f(a * b);
```

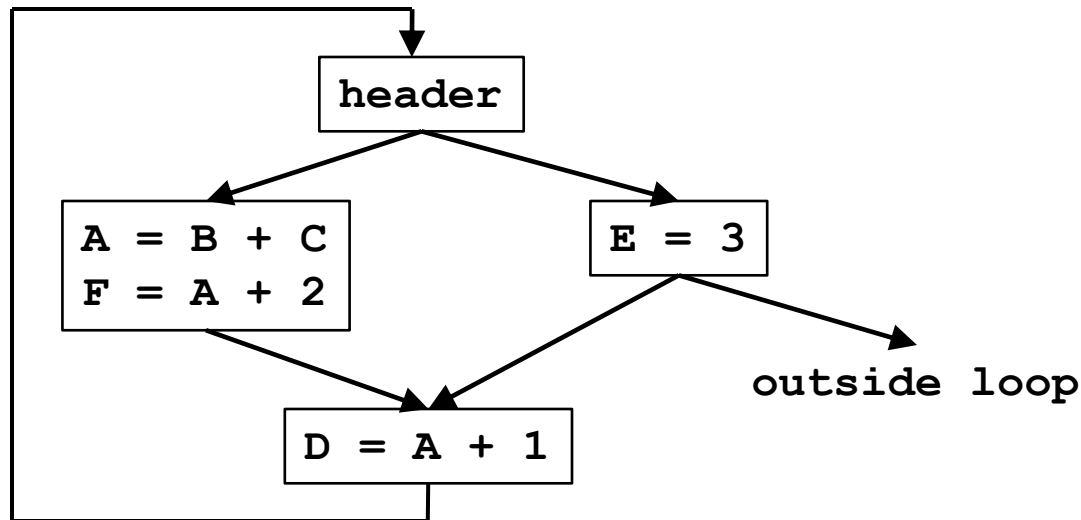


```
a = ...;  
b = ...;  
c = a * b;  
for (i = 0; i < 100; i++)  
    f(c);
```

- Sposta le istruzioni che non dipendono dal control flow del loop fuori dal loop stesso
- Evita di ricalcolare in maniera ridondante la stessa cosa
- Essendo il grosso della computazione di un programma contenuta nei loop c'è enorme potenziale di miglioramento della performance

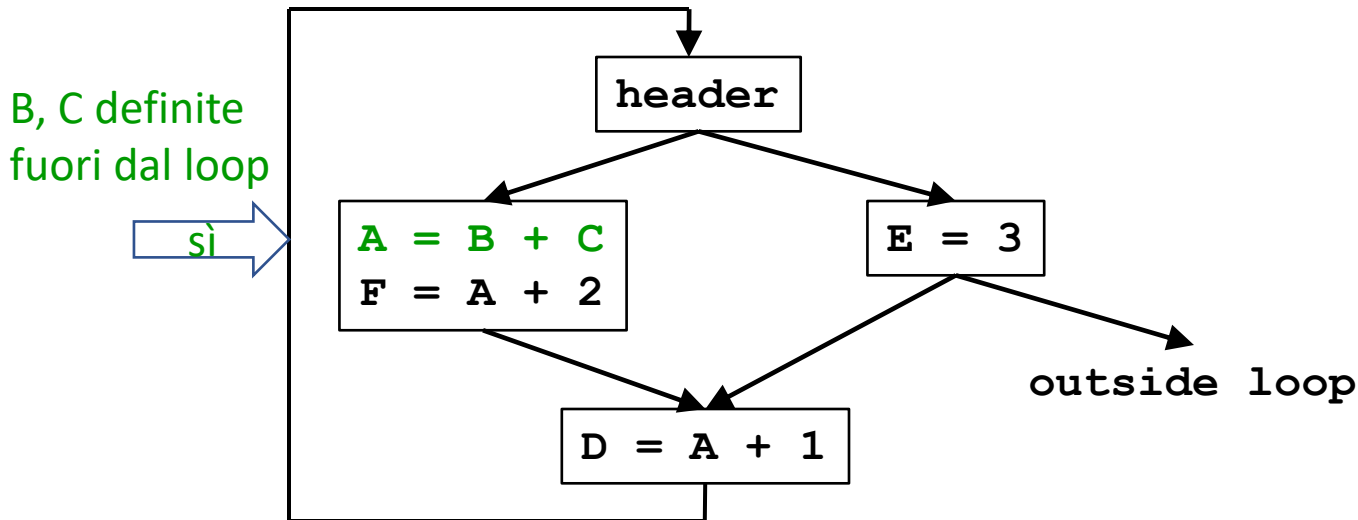
Istruzioni Loop-Invariant

- Quali istruzioni sono *loop-invariant* in questo esempio?
 - istruzioni il cui valore non cambia fintanto che il controllo rimane dentro il loop



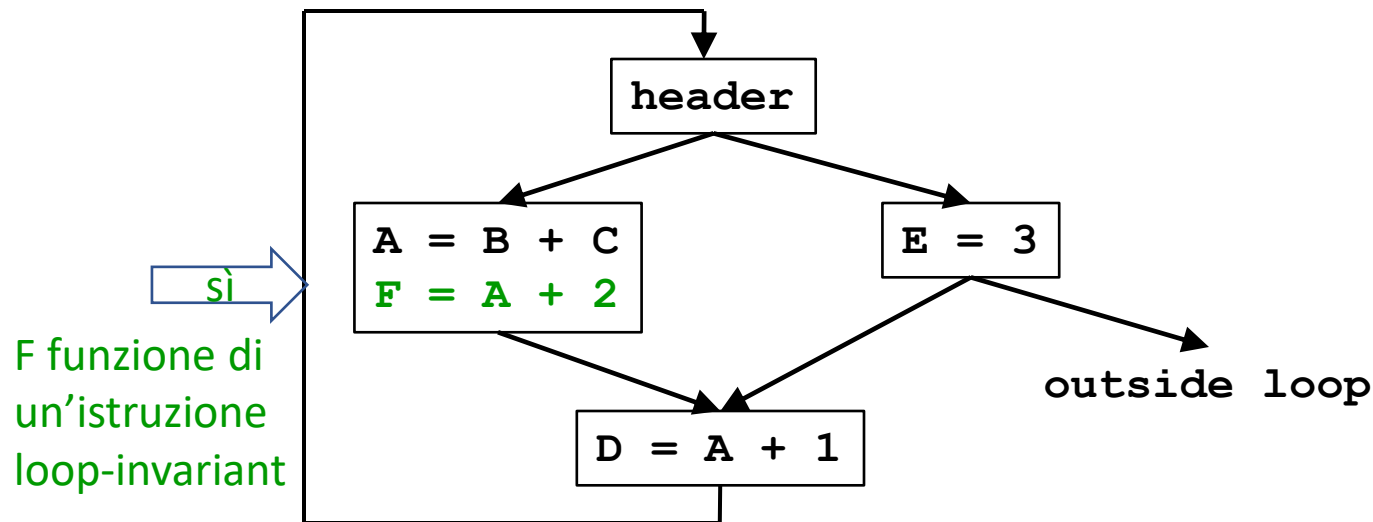
Istruzioni Loop-Invariant

- Quali istruzioni sono *loop-invariant* in questo esempio?
 - istruzioni il cui valore non cambia fintanto che il controllo rimane dentro il loop



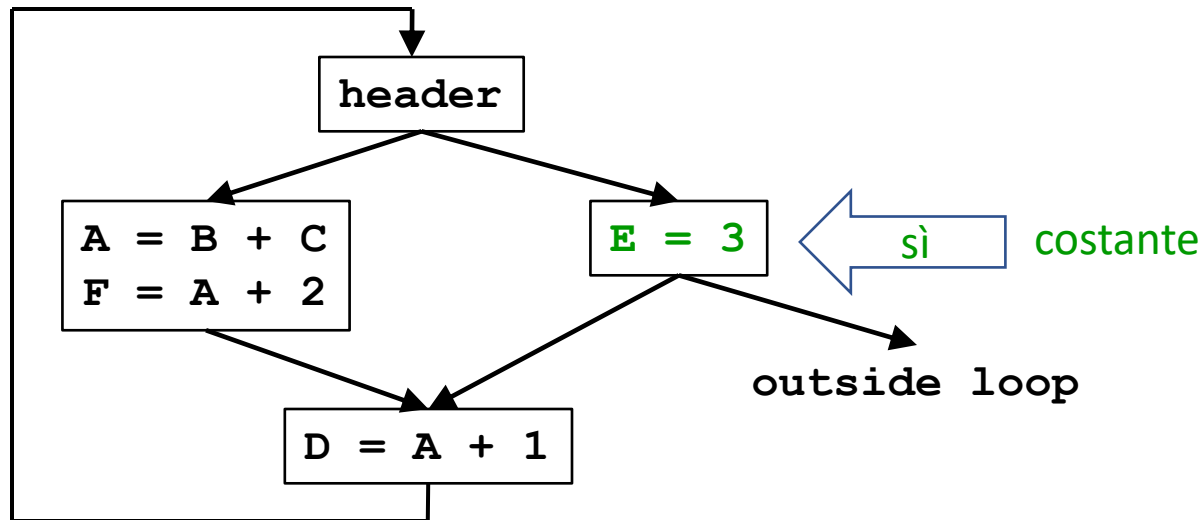
Istruzioni Loop-Invariant

- Quali istruzioni sono *loop-invariant* in questo esempio?
 - istruzioni il cui valore non cambia fintanto che il controllo rimane dentro il loop



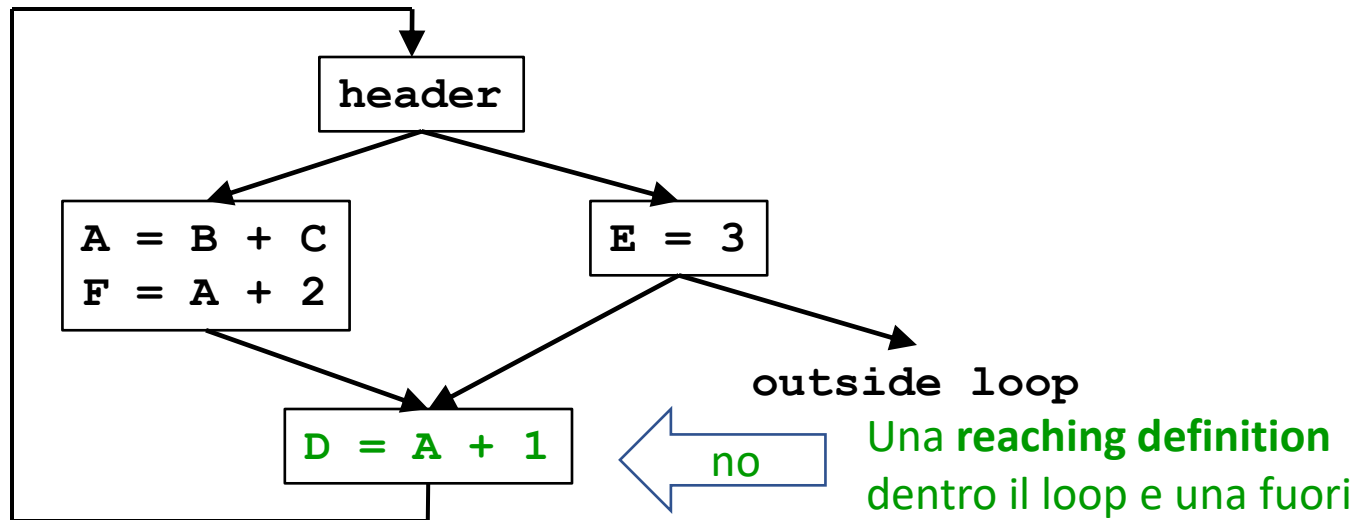
Istruzioni Loop-Invariant

- Quali istruzioni sono *loop-invariant* in questo esempio?
 - istruzioni il cui valore non cambia fintanto che il controllo rimane dentro il loop



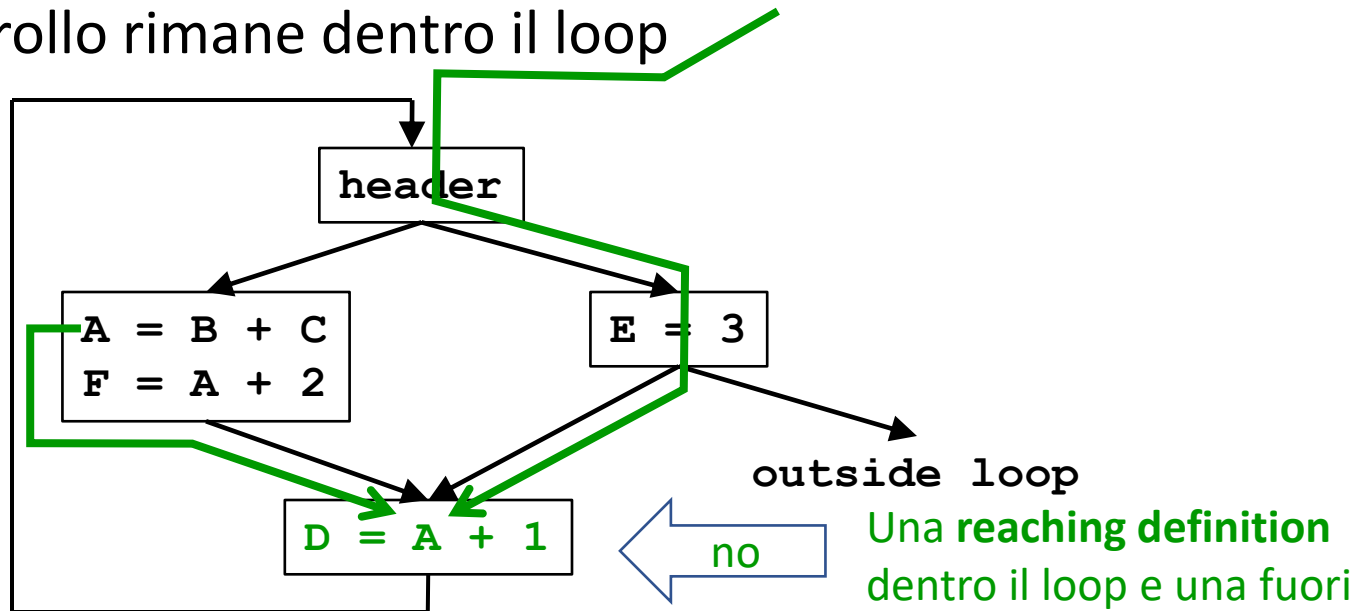
Istruzioni Loop-Invariant

- Quali istruzioni sono *loop-invariant* in questo esempio?
 - istruzioni il cui valore non cambia fintanto che il controllo rimane dentro il loop



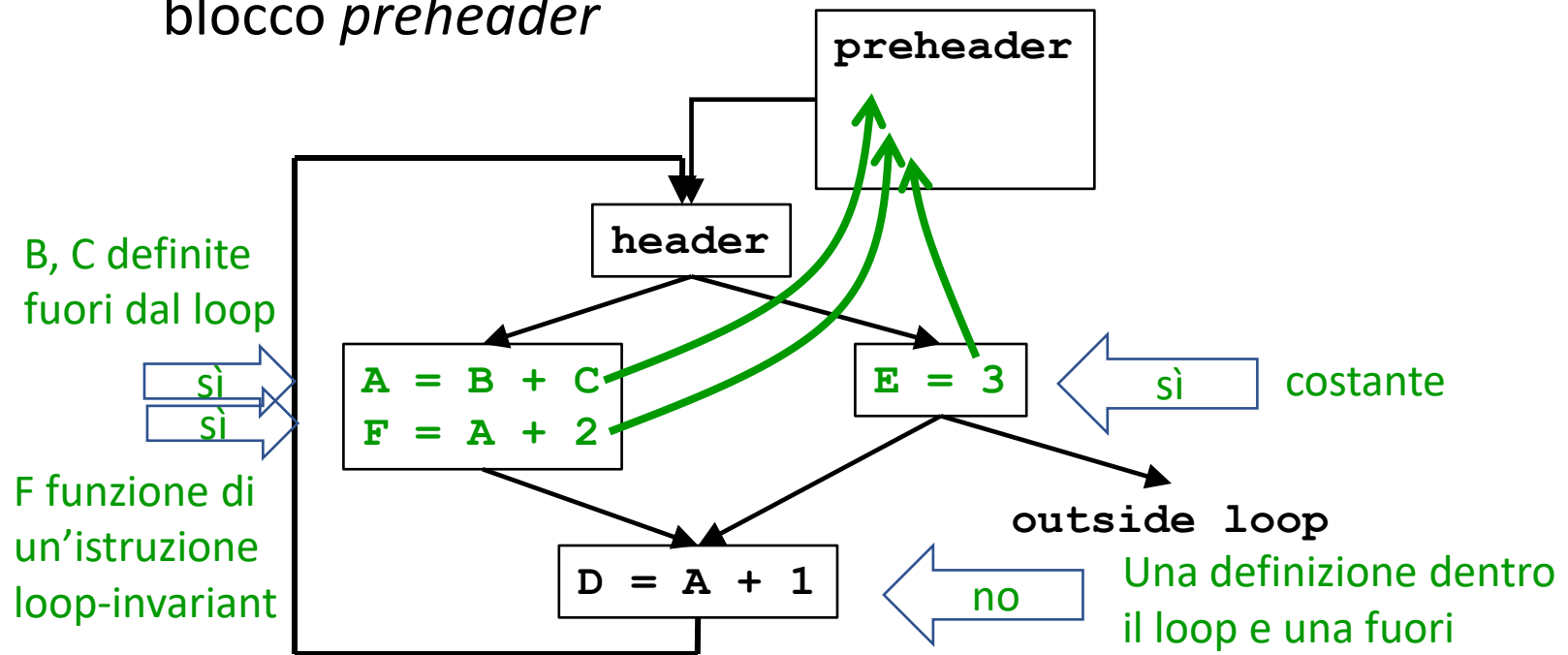
Istruzioni Loop-Invariant

- Quali istruzioni sono *loop-invariant* in questo esempio?
 - istruzioni il cui valore non cambia fintanto che il controllo rimane dentro il loop



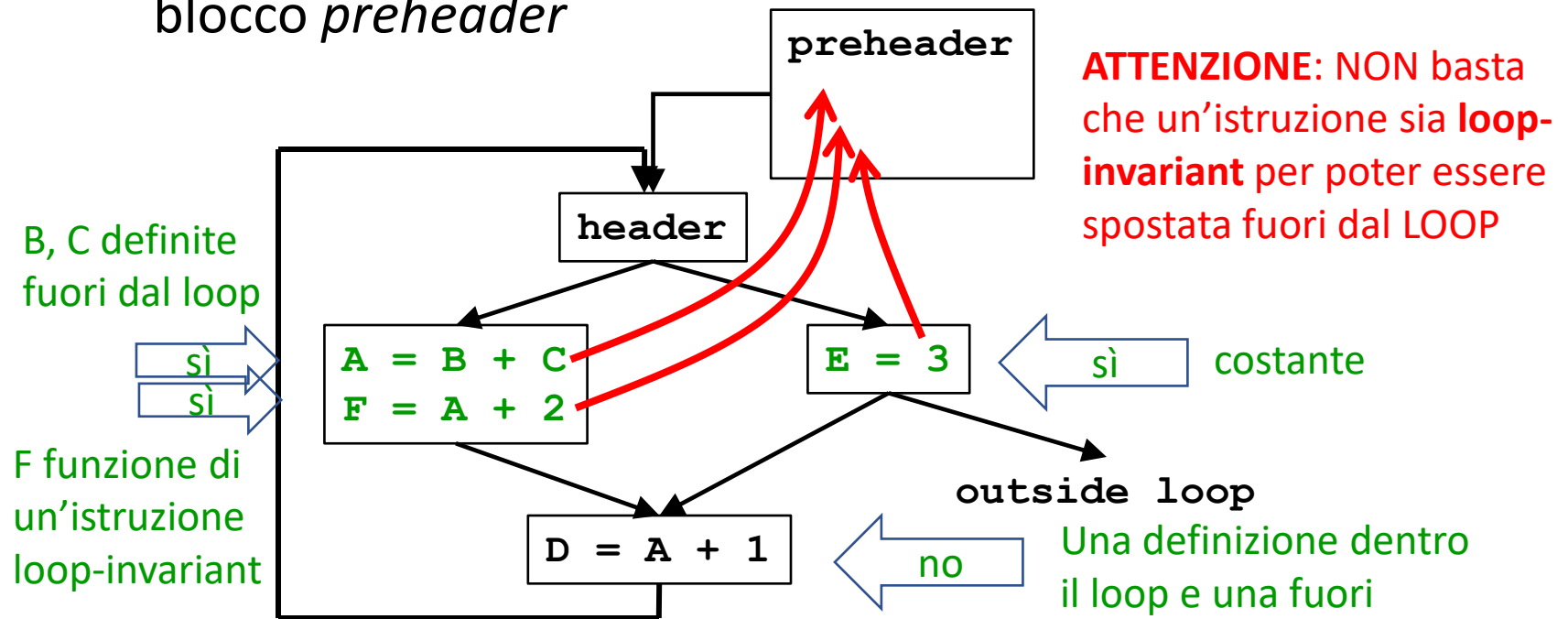
Code Motion

- Code motion:
 - Lo spostamento delle istruzione *loop-invariant* dentro il blocco *preheader*



Code Motion

- Code motion:
 - Lo spostamento delle istruzione *loop-invariant* dentro il blocco *preheader*



Algoritmo

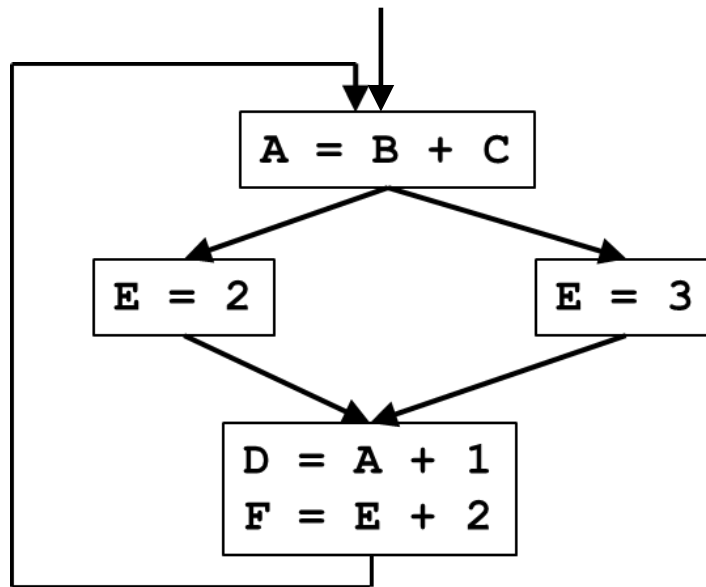
- Osservazioni
 - Istruzioni *loop-invariant*
 - Gli operandi sono definiti fuori dal loop o tramite istruzioni che sono *loop-invariant*
 - *Code motion*
 - Non tutte le istruzioni *loop-invariant* possono essere spostate nel *preheader*
- Algoritmo (tre macro fasi)
 - Troviamo le istruzioni *loop-invariant*
 - Verifichiamo che le condizioni per la *code motion* siano soddisfatte
 - Spostiamo le istruzioni

Identificare le istruzioni Loop Invariant

1. Calcolare le *reaching definitions*
2. Per ogni istruzione del tipo $A=B+C$ marcare l'istruzione come Loop-INVARIANT se:
 - Tutte le definizioni di B e C che raggiungono l'istruzione $A=B+C$ si trovano fuori dal loop
 - Se B e/o C fossero costanti?
 - **Oppure** c'è esattamente una *reaching definition* per B (e C), e si tratta di un'istruzione del loop che è stata marcata *loop-invariant*

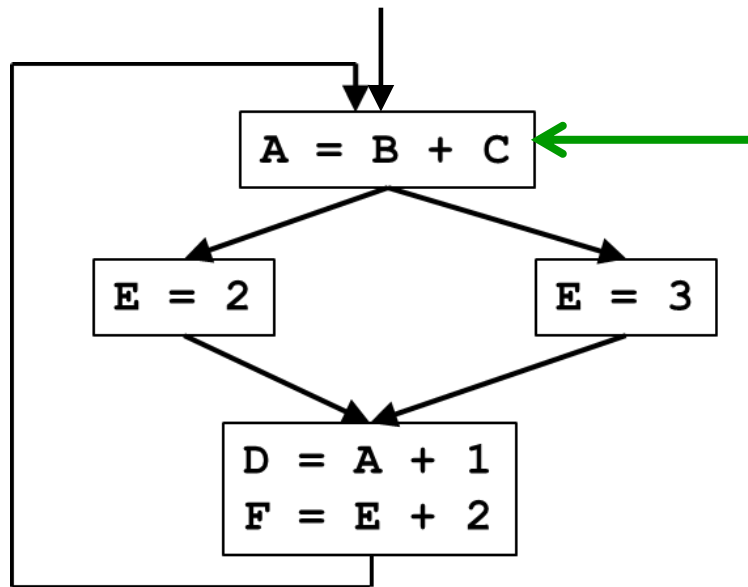
Esempio

- Quali istruzioni sono *loop-invariant*?



Esempio

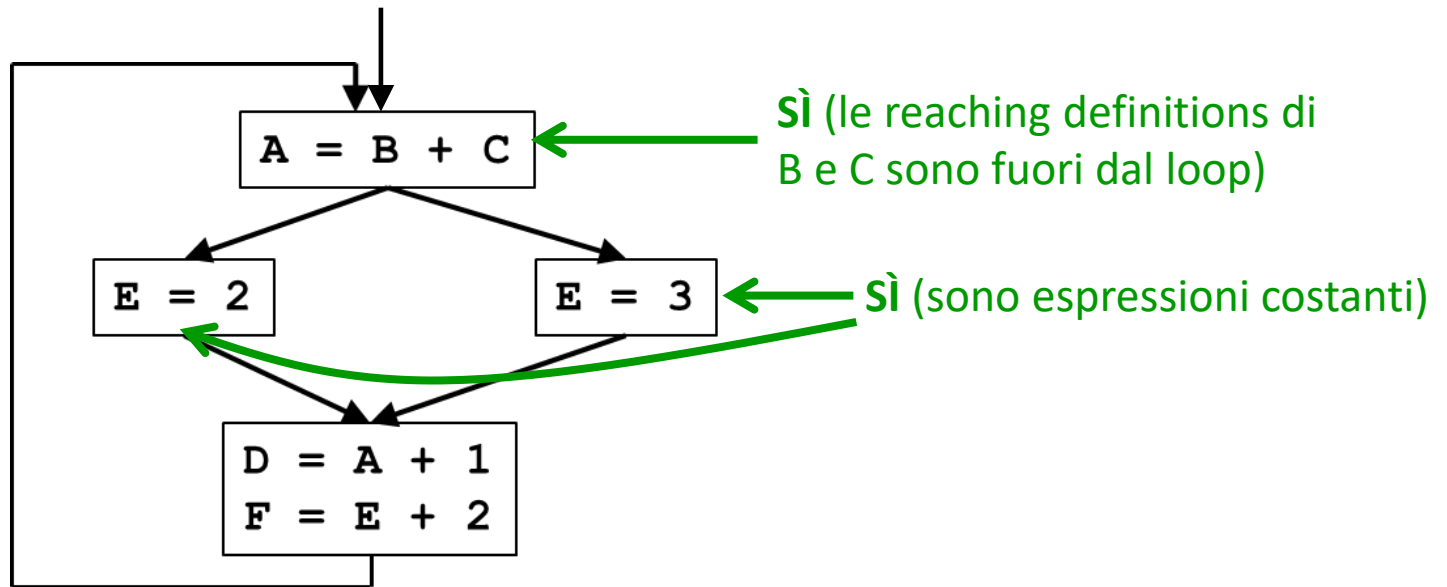
- Quali istruzioni sono *loop-invariant*?



Sì (le reaching definitions di B e C sono fuori dal loop)

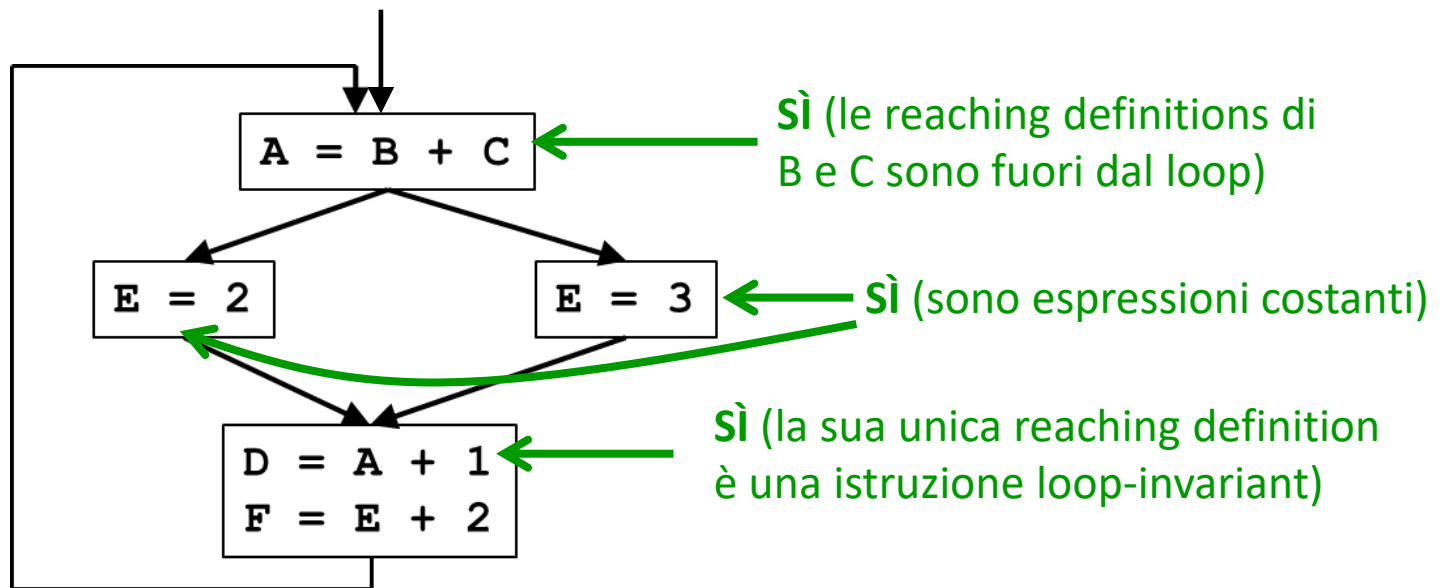
Esempio

- Quali istruzioni sono *loop-invariant*?



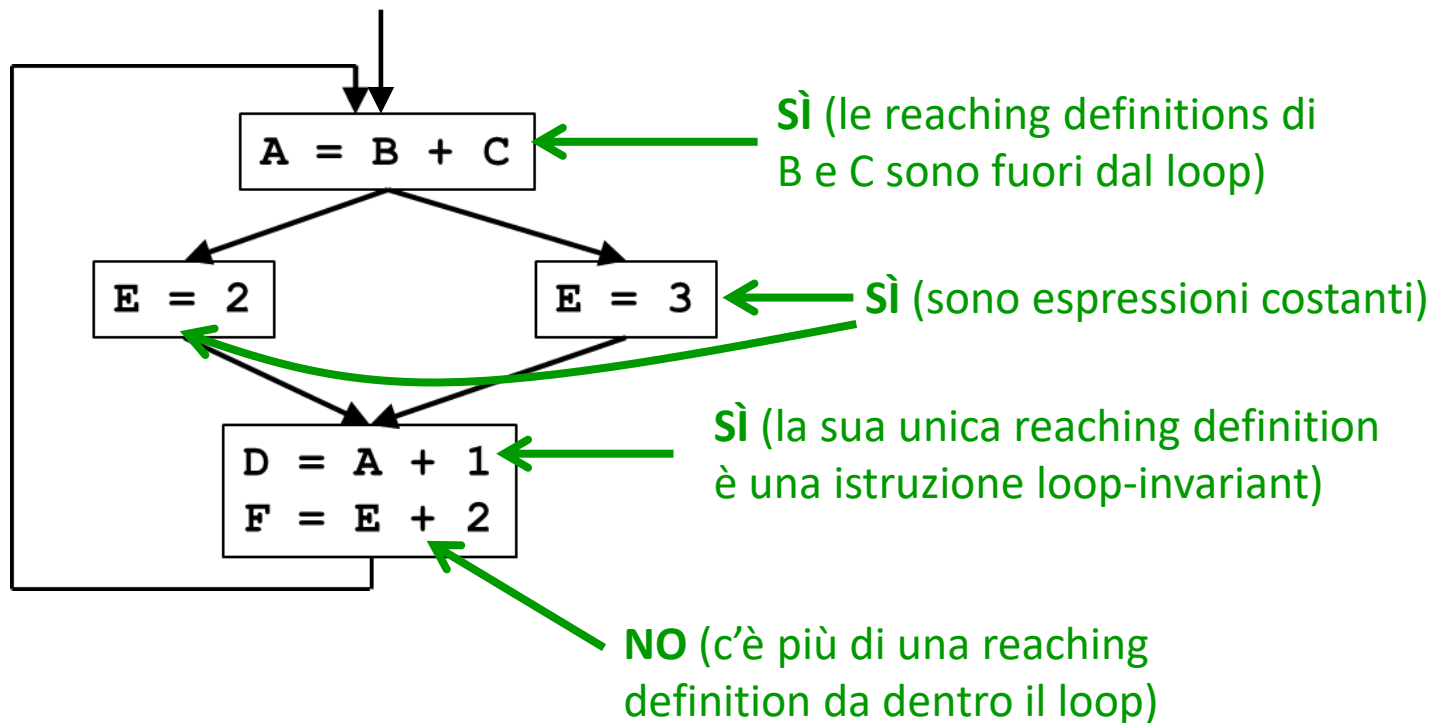
Esempio

- Quali istruzioni sono *loop-invariant*?



Esempio

- Quali istruzioni sono *loop-invariant*?



Condizioni per la Code Motion

- Condizioni generali (valide per tutte le trasformazioni):
 - **Correttezza:** Lo spostamento del codice non altera la semantica del programma
 - **Performance:** l'esecuzione del codice non è rallentata

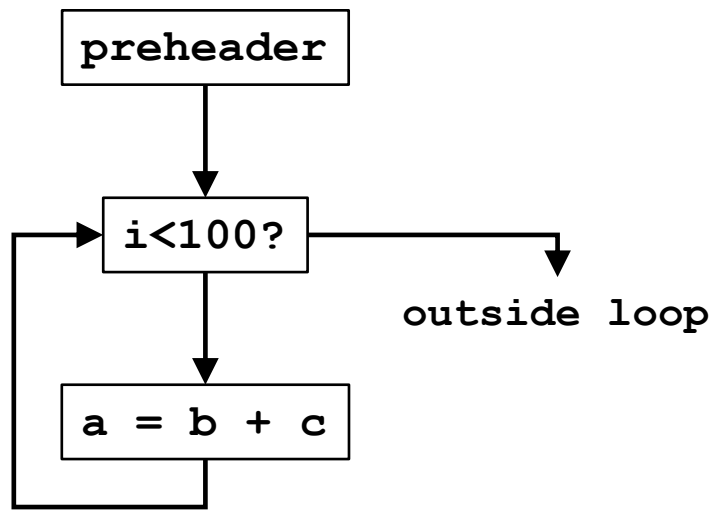
Condizioni per la Code Motion

- **Idea di base:** L'istruzione candidata per la *code motion* definisce la variabile nel suo LHS **una volta** e **per tutte** nel loop:
 - **control flow:** **una volta**?
L'istruzione (il blocco) domina tutte le uscite del loop
 - **Altre definizioni:** **per tutte**?
Non ci sono altre definizioni della variabile nel loop
 - **Altri usi:** **per tutte**?
L'istruzione (la definizione) domina tutti gli usi o non ci sono altre *reaching definitions*

Esempio

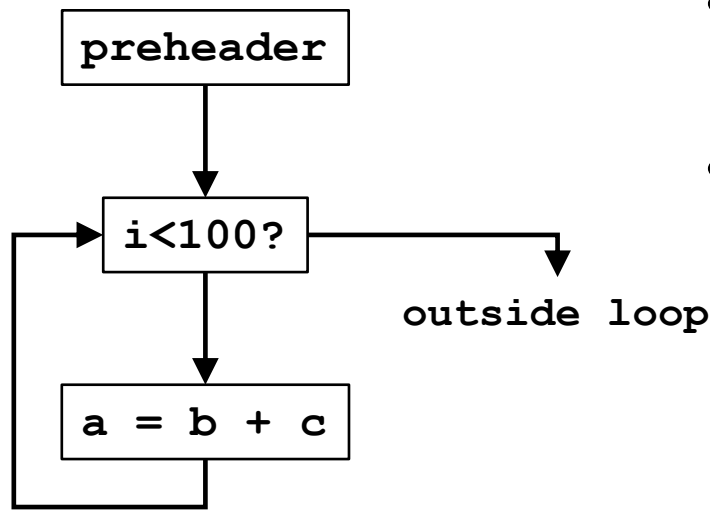
```
a = ...;  
x = ...;  
for (i = x; i < 100; i++)  
    a = b + c;
```

- L'istruzione $a = b + c$ è *loop-invariant*
- Posso procedere alla *code motion*?



Esempio

```
a = ...;  
x = ...;  
for (i = x; i < 100; i++)  
    a = b + c;
```

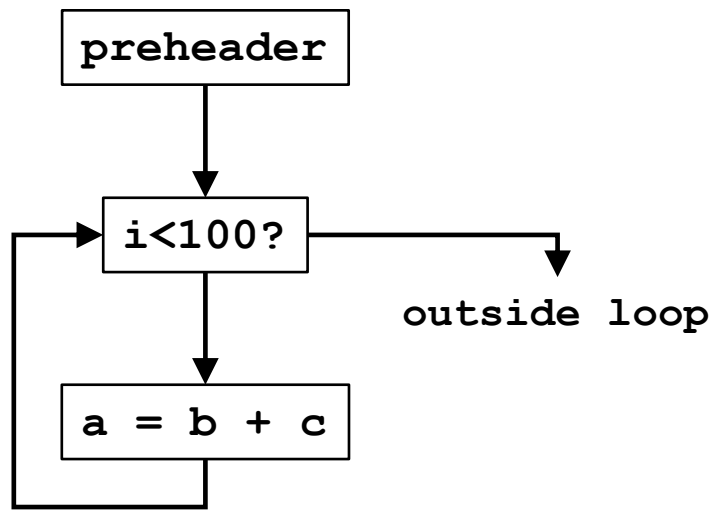


- L'istruzione $a = b + c$ è *loop-invariant*
- Ci sono altre definizioni della variabile nel loop?
- La definizione domina tutti gli usi?
- Il blocco domina tutte le uscite?

Esempio

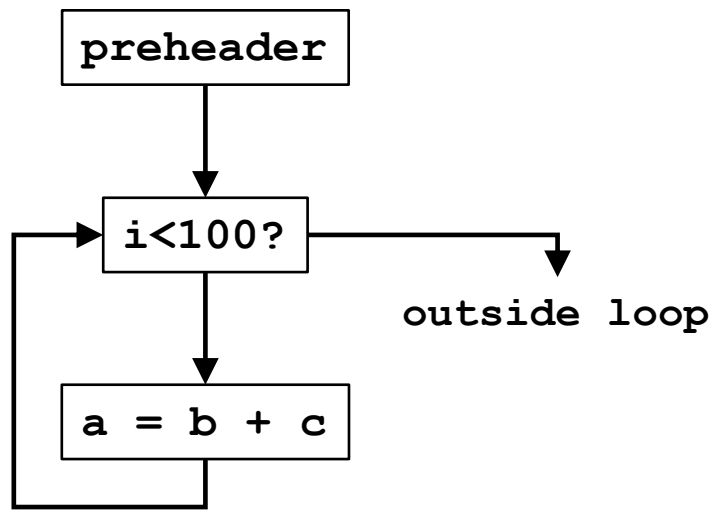
```
a = ...;  
x = ...;  
for (i = x; i < 100; i++)  
    a = b + c;
```

- Cosa succede se $x \geq 100$?



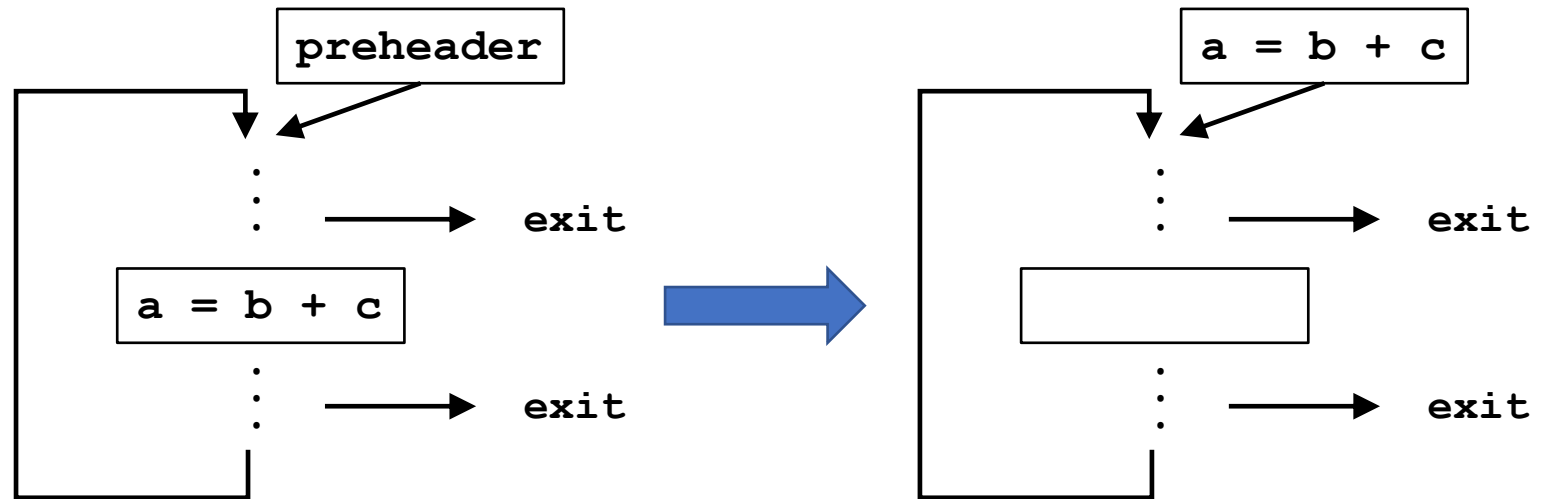
Esempio

```
a = ...;  
x = ...;  
for (i = x; i < 100; i++)  
    a = b + c;
```



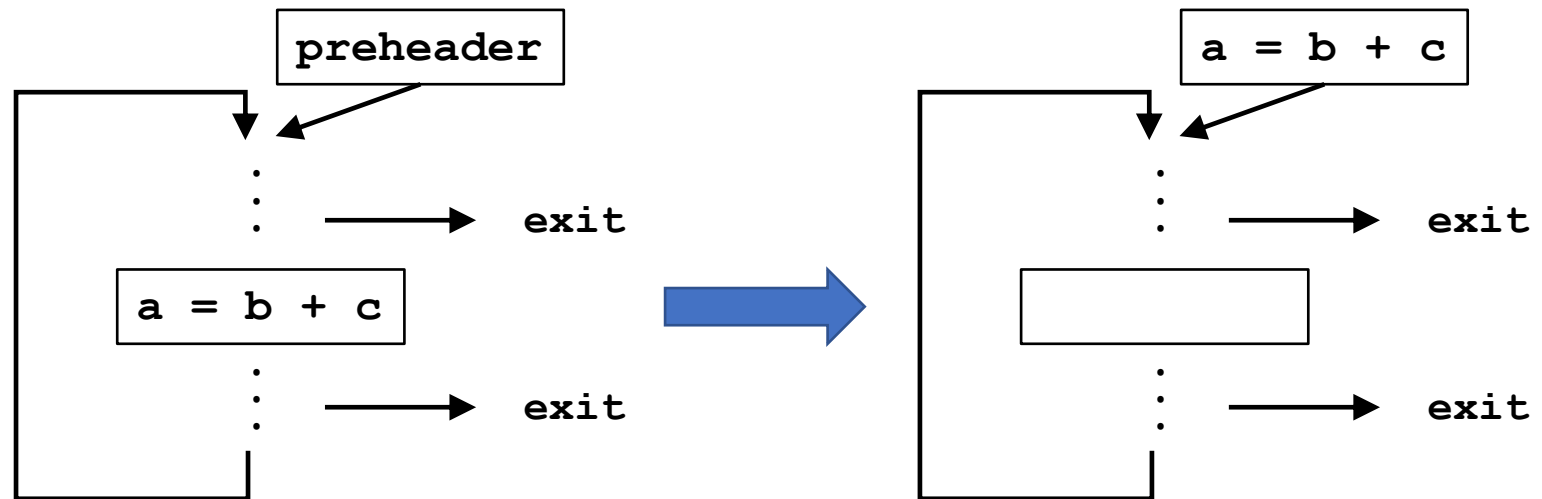
- Cosa succede se $x \geq 100$?
- Il loop non viene eseguito
- Se spostassi l'istruzione $a = b + c$ nel *preheader* cambierei la semantica del programma
 - Che porterebbe in uscita al loop la definizione di a precedente al loop

Esempio (2)



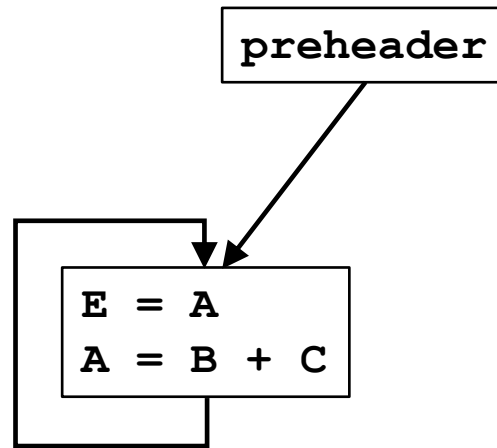
- È corretto effettuare lo spostamento del codice al *preheader*?

Esempio (2)



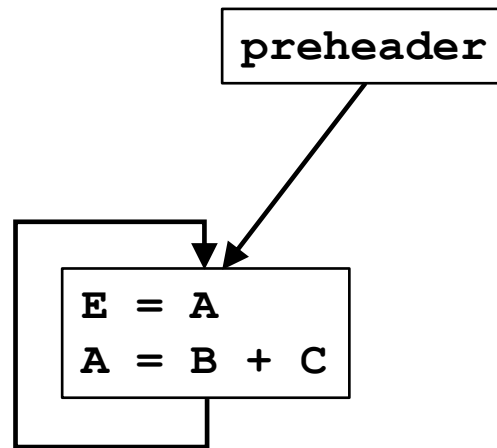
- È corretto effettuare lo spostamento del codice al *preheader*?
 - **NO**: stiamo modificando le proprietà di dominanza spostando l'istruzione prima della prima uscita

Esempio (3)



- È corretto effettuare lo spostamento del codice al *preheader*?

Esempio (3)



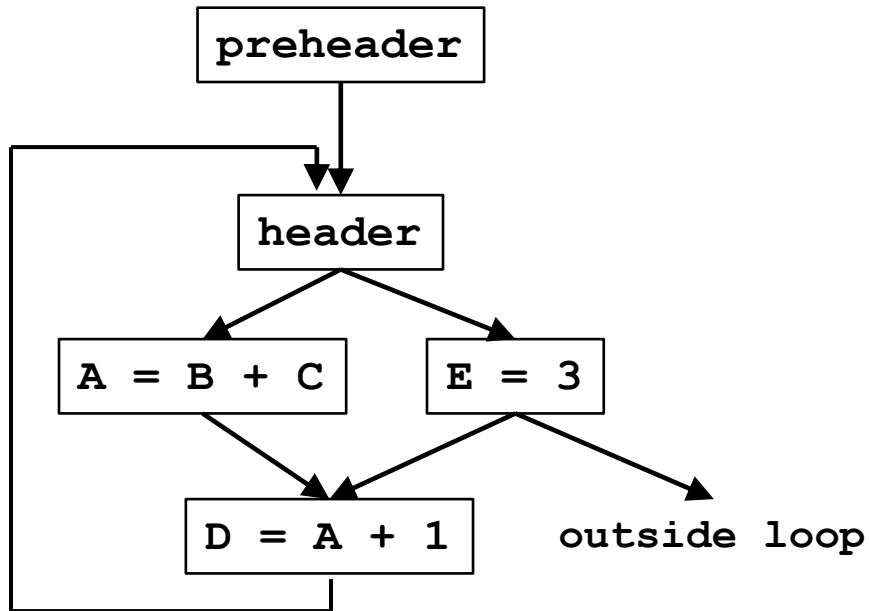
- È corretto effettuare lo spostamento del codice al *preheader*?
 - **NO**: La prima istruzione non è *loop-invariant*
 - **NO**: La seconda istruzione non domina i suoi usi

Algoritmo per la Code Motion

Dato un insieme di nodi in un loop

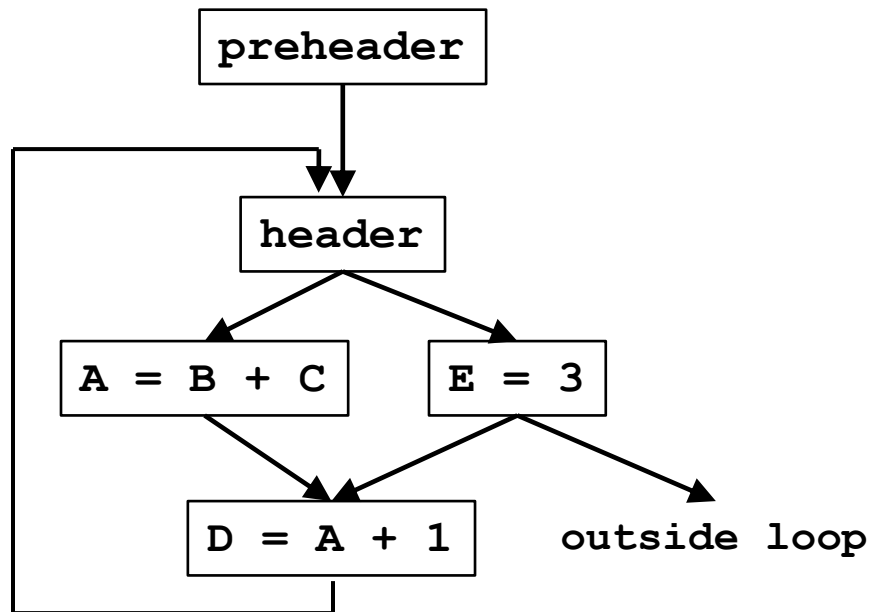
- Calcolare le *reaching definitions*
- Trovare le istruzioni *loop-invariant*
- Calcolare i dominatori (*dominance tree*)
- Trovare le uscite del loop (i successori fuori dal loop)
- Le istruzioni candidate alla *code motion*:
 - Sono *loop invariant*
 - Si trovano in blocchi che dominano tutte le uscite del loop
 - Assegnano un valore a variabili non assegnate altrove nel loop
 - Si trovano in blocchi che dominano tutti i blocchi nel loop che usano la variabile a cui si sta assegnando un valore
- Eseguire una ricerca *depth-first* dei blocchi
 - Spostare l'istruzione candidata nel *preheader* se tutte le istruzioni invarianti da cui questa dipende sono state spostate

Esempio



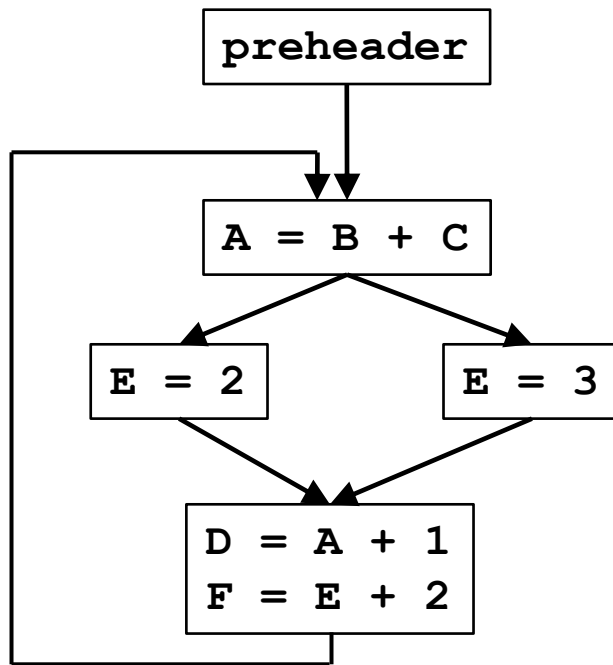
- Quali istruzioni possono essere spostate nel *preheader*?

Esempio



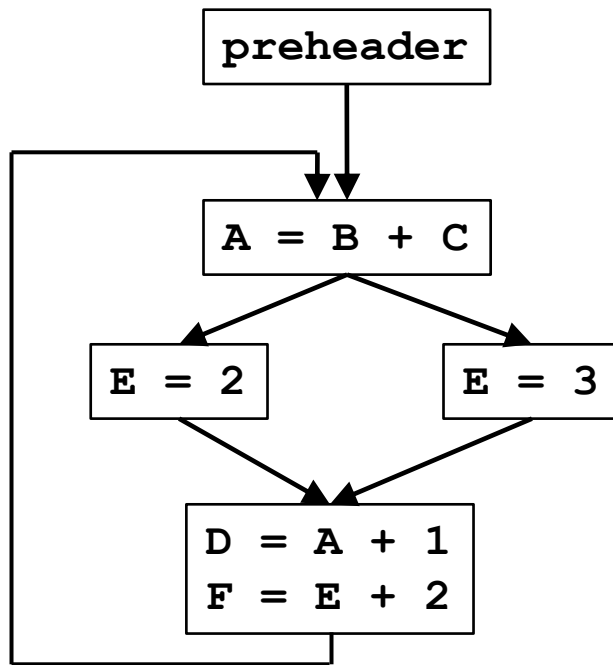
- Quali istruzioni possono essere spostate nel *preheader*?
- Solo $E = 3$
 - Unica istruzione che domina tutte le uscite

Esempio (2)



- Quali istruzioni possono essere spostate nel *preheader*?

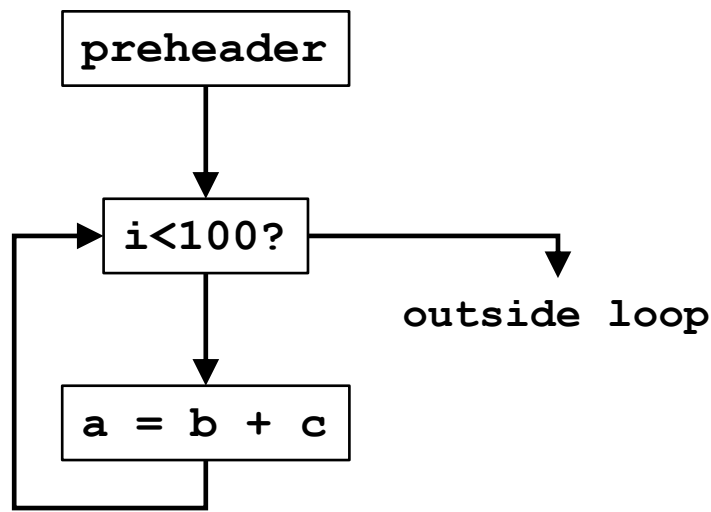
Esempio (2)



- Quali istruzioni possono essere spostate nel *preheader*?
- $A = B + C$
- $D = A + 1$
- Anche se $E=2$ e $E=3$ sono istruzioni *loop-invariant* nessuna è l'unica definizione di E
- **Ricorda:** le istruzioni devono definire la variabile **una volta e per tutte**

Possiamo essere più aggressivi?

```
a = ...;  
x = ...;  
for (i = x; i < 100; i++)  
    a = b + c;
```



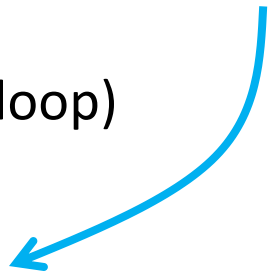
- Torniamo all'esempio di prima ($x < 100$)
- Anche se l'istruzione $a = b + c$ è *loop-invariant* non posso procedere alla *code motion*
 - Perché il blocco che ospita l'istruzione non domina tutte le uscite
- Ma cosa succederebbe se la variabile a non fosse mai usata fuori dal loop?

Algoritmo per la Code Motion

Dato un insieme di nodi in un loop

- Calcolare le *reaching definitions*
- Trovare le istruzioni *loop-invariant*
- Calcolare i dominatori (*dominance tree*)
- Trovare le uscite del loop (i successori fuori dal loop)
- Le istruzioni candidate alla *code motion*:
 - Sono *loop invariant*
 - **Si trovano in blocchi che dominano tutte le uscite del loop**
 - Assegnano un valore a variabili non assegnate altrove nel loop
 - Si trovano in blocchi che dominano tutti i blocchi nel loop che usano la variabile a cui si sta assegnando un valore
- Eseguire una ricerca *depth-first* dei blocchi
 - Spostare l'istruzione candidata nel *preheader* se tutte le istruzioni invarianti da cui questa dipende sono state spostate

Oppure la variabile definita dall'istruzione è **dead** all'uscita del loop





UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche,
Informatiche e Matematiche

Assignment