Traduzione di espressioni in assembly Motorola 68000

Daniele Paolo Scarpazza
daniele.scarpazza@elet.polimi.it

Politecnico di Milano
Ultimo aggiornamento: 10 Maggio 2005

Traduzione e assemblaggio

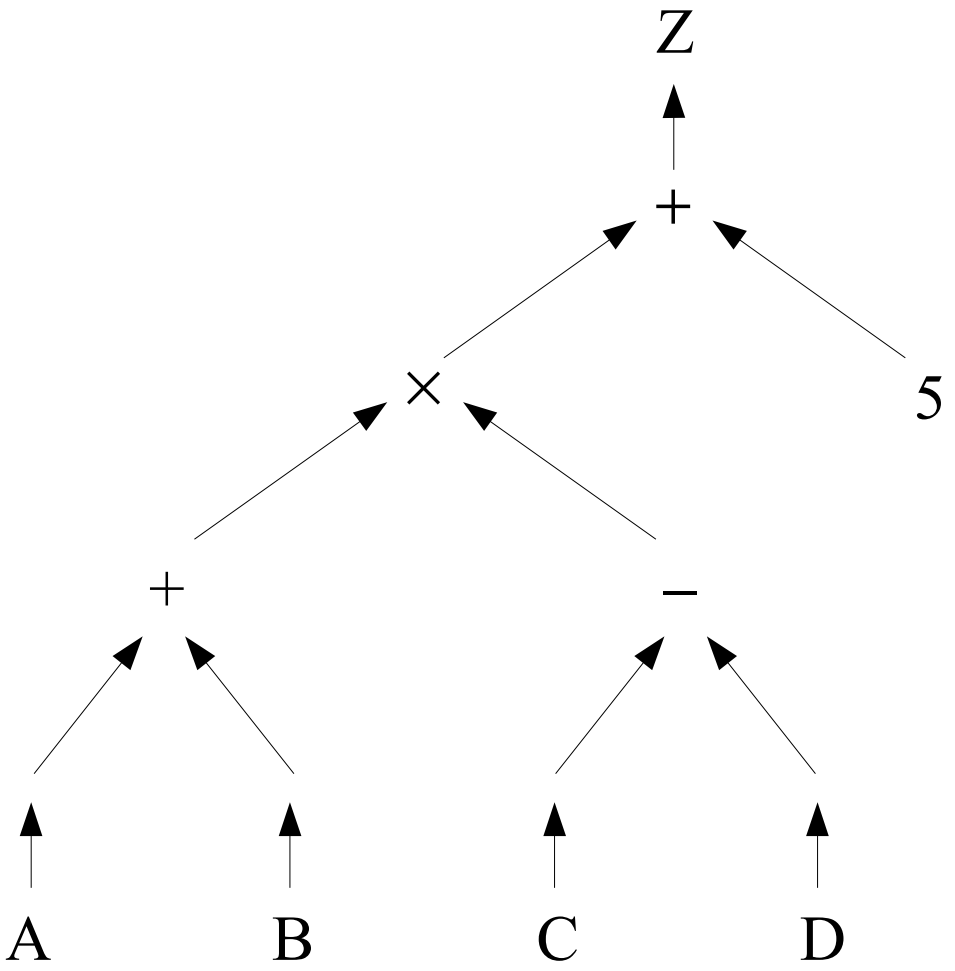
- Traduzione:
  - dato un programma, espresso in un linguaggio ad alto livello (per esempio C)
  - preparare il programma corrispondente, in linguaggio assembly;
- Assemblaggio:
  - tradurre il programma assembly in codice oggetto, direttamente eseguibile dal processore;

Traduzione

- Svolgere l'operazione nelle seguenti fasi:
  1. ricavare l'albero sintattico;
  2. numerare i nodi interni dell'albero
     secondo una visita *postordine*;
  3. coprire l'albero con le tegole prefissate;
  4. allocare i registri
     (nell'ordine di numerazione, senza ottimizzazioni,
     rispettando i vincoli);
  5. codificare in assembly le tegole
     (nell'ordine di numerazione, con i registri allocati);

1. Determinare l'albero sintattico

- Espressione di esempio: $Z = (A + B) \times (C - D) + 5$
- Determinazione dell'albero sintattico:



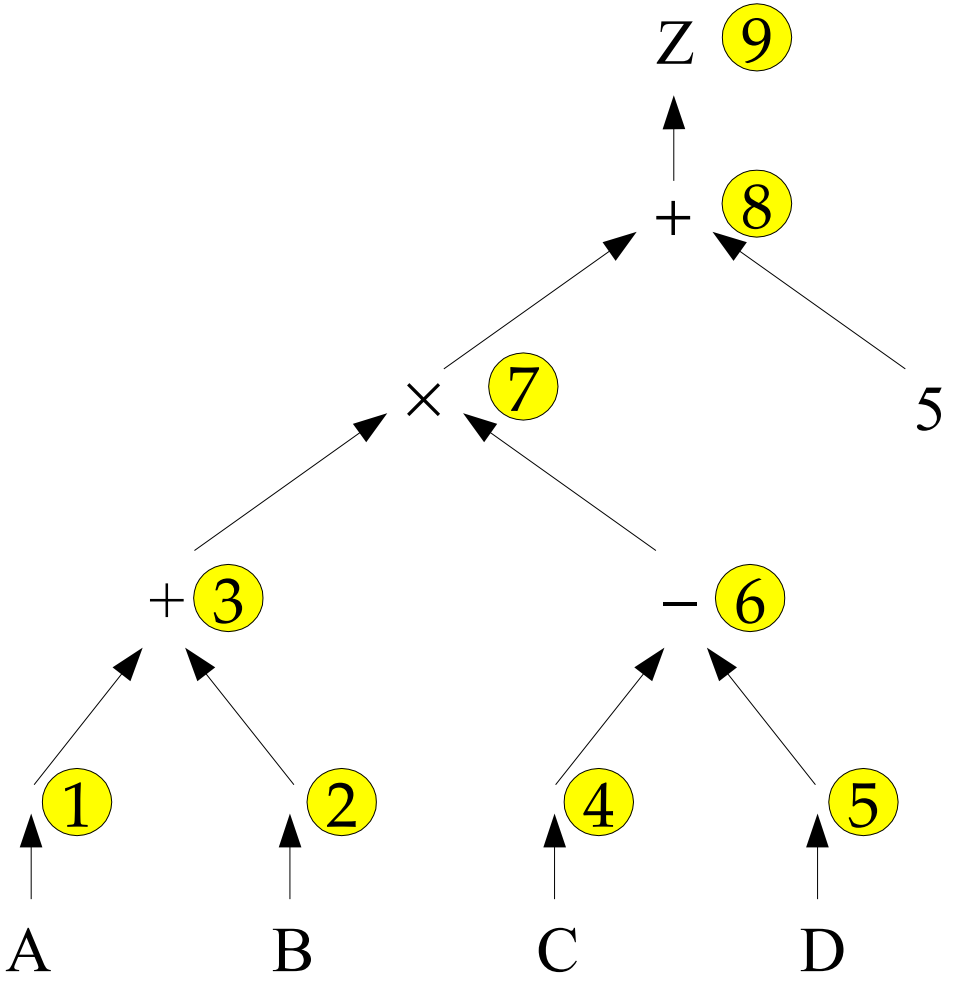
2. Numerazione dei nodi dell'albero

- Numerari i nodi **interni** (cioè tutti i nodi tranne le foglie) secondo una visita postordine;
- Ricordate: dato un albero, una **visita** dei suoi nodi è una sequenza ordinata in cui ogni nodo compare esattamente una volta;
- La visita postordine si ottiene considerando i nodi come raccomandato dall' algoritmo ricorsivo (pseudocodice C):

```
void VisitaPostordine(Nodo * pn)
{
   VisitaPostordine(pn->figlio1);
   VisitaPostordine(pn->figlio2);
   ... /* eventuali altri figli,
          numerati da sx a dx */
   ConsideraNodo(pn);
}
```

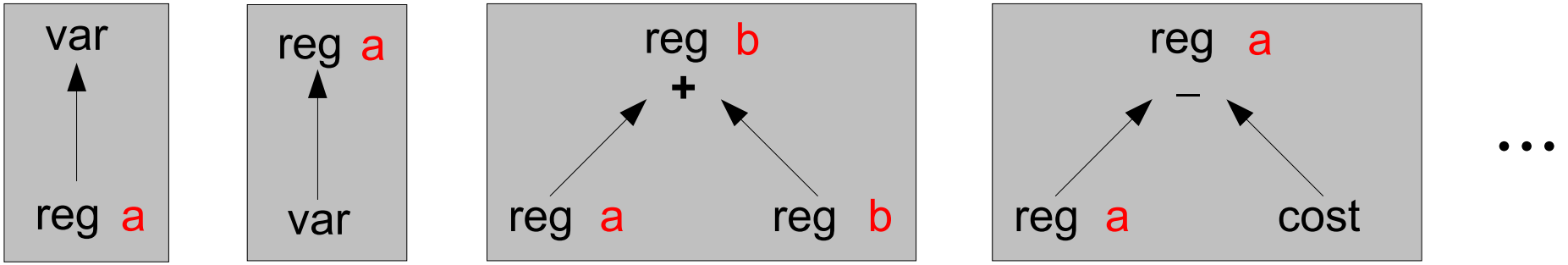
2. Numerazione dei nodi dell'albero

- Applichiamo la visita postordine dei nodi interni:



3. Copertura dell'albero con tegole

- Avete a disposizione un insieme di tegole prefissato (vedere prossime pagine), fatte ad esempio come segue:

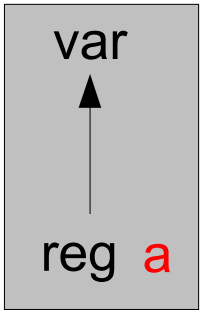


- Disponete le tegole in modo che combacino con i nodi dell'albero; ogni nodo interno (salvo la radice) sia coperto da due tegole;
- Le tegole sono parzialmente sovrapposte; le zone sovrapposte **devono** combaciare;
- Ignorate per ora i simboli in rosso, essi vincolano la fase successiva di allocazione dei registri;

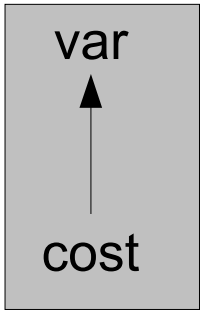
Tegole da usare per la copertura

Per le istruzioni di trasferimento dati:

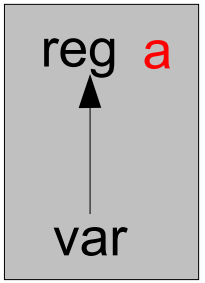
- **`move Da, var`**



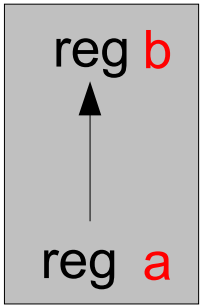
- **`move #cost, var`**



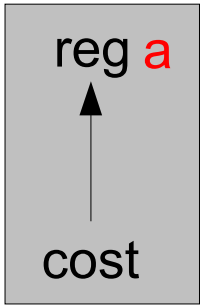
- **`move var, Da`**



- **`move Da, Db`**



- **`move #cost, Da`**

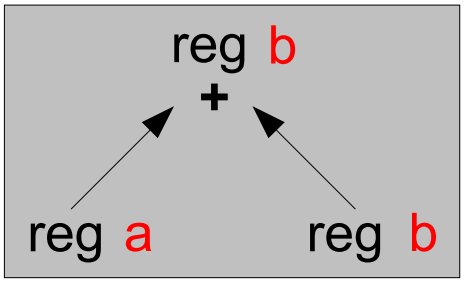


(Per uniformità, attenersi alle tegole date, anche se non sono ottime)

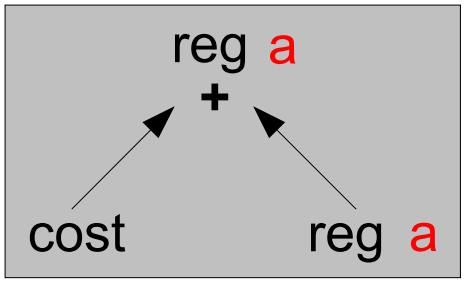
Tegole da usare per la copertura

Operazioni aritmetiche:

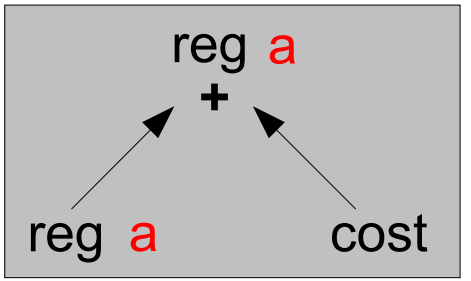
- **`add Da, Db`**



- **`add #cost, Da`**

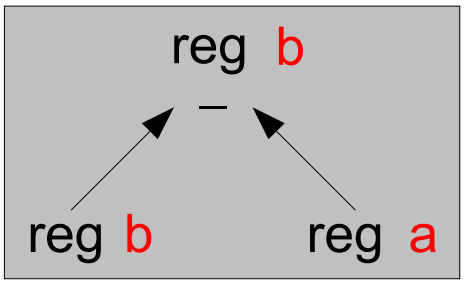


- **`add #cost, Da`**



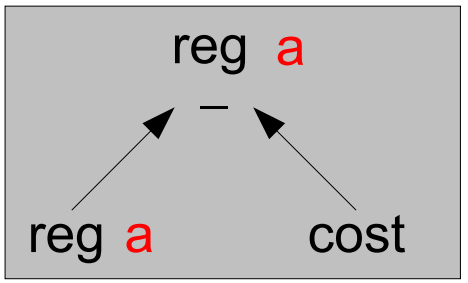
Queste due tegole hanno la stessa traduzione assembly in virtù della commutatività dell'operatore "+".

- **`sub Da, Db`**



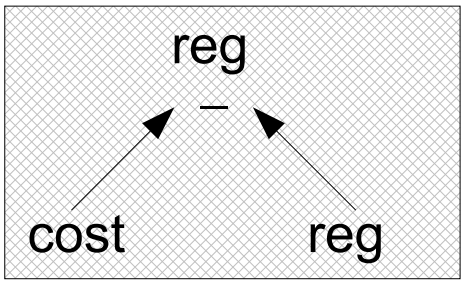
Attenzione all'ordine degli operandi!

- **`sub #cost, Da`**



Attenzione all'ordine degli operandi!

- Nota:



Questa tegola non esiste!

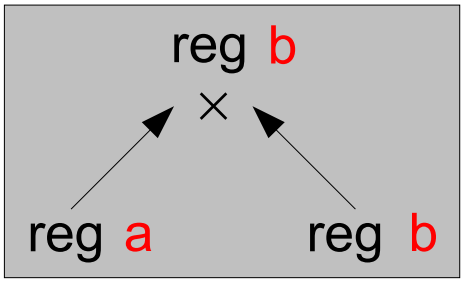
Se si presenta questo sottoalbero, spostare cost in un registro e poi usare la prima tegola a sinistra.

(Per uniformità, attenersi alle tegole date, anche se non sono ottime)

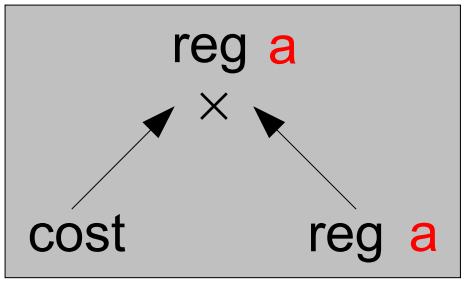
Tegole da usare per la copertura

Ancora operazioni aritmetiche: (come prima)

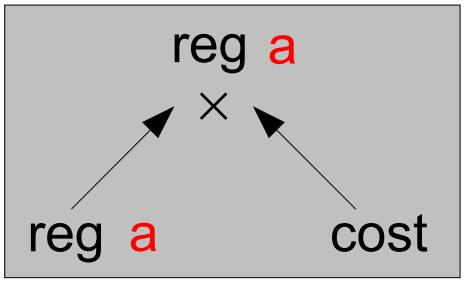
- **`muls Da, Db`**



- **`muls #cost, Da`**

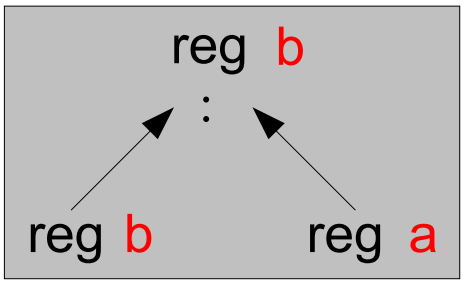


- **`muls #cost, Da`**



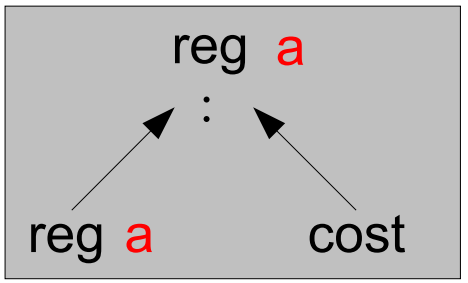
Queste due tegole hanno la stessa traduzione assembly in virtù della commutatività dell'operatore "×".

- **`divs Da, Db`**



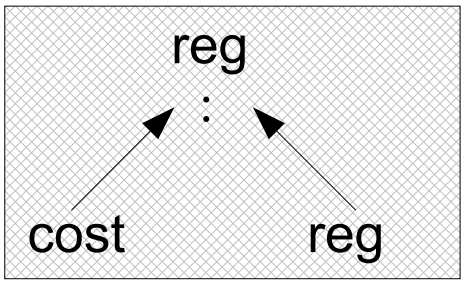
Attenzione all'ordine degli operandi!

- **`divs #cost, Da`**



Attenzione all'ordine degli operandi!

- Nota:



Questa tegola non esiste!

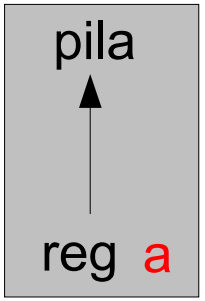
Se si presenta questo sottoalbero, spostare cost in un registro e poi usare la prima tegola a sinistra.

(Per uniformità, attenersi alle tegole date, anche se non sono ottime)

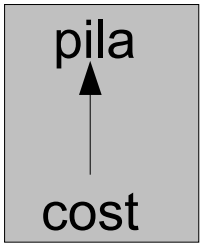
Tegole da usare per la copertura

Push dei parametri di funzione:

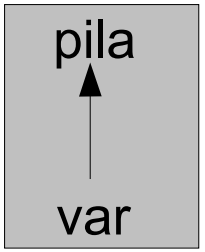
- **`move Da, -(A7)`**



- **`move #cost, -(A7)`**



- **`move var, -(A7)`**

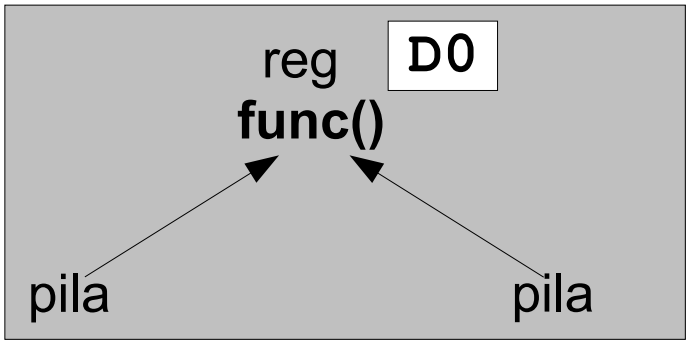


Attenzione:

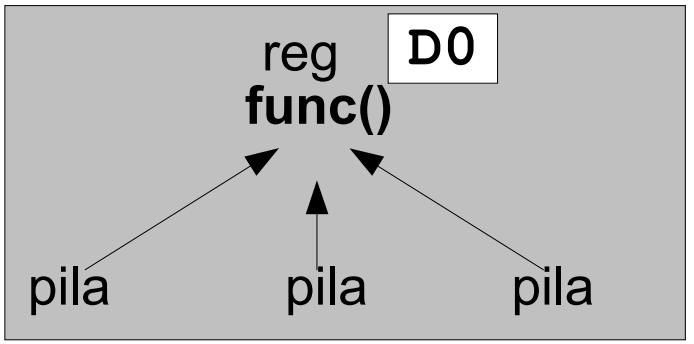
- eccezione alla regola di numerazione dei nodi;
- se dovete caricare i parametri sulla pila in ordine inverso, le tegole "push" vanno numerate da destra a sinistra, invece che da sinistra a destra;

Chiamata di funzione:

- **`BSR func`**
  `ADD #N, A7`
  (ex: 2 parametri L in ingresso sulla pila e valore di ritorno in D0)

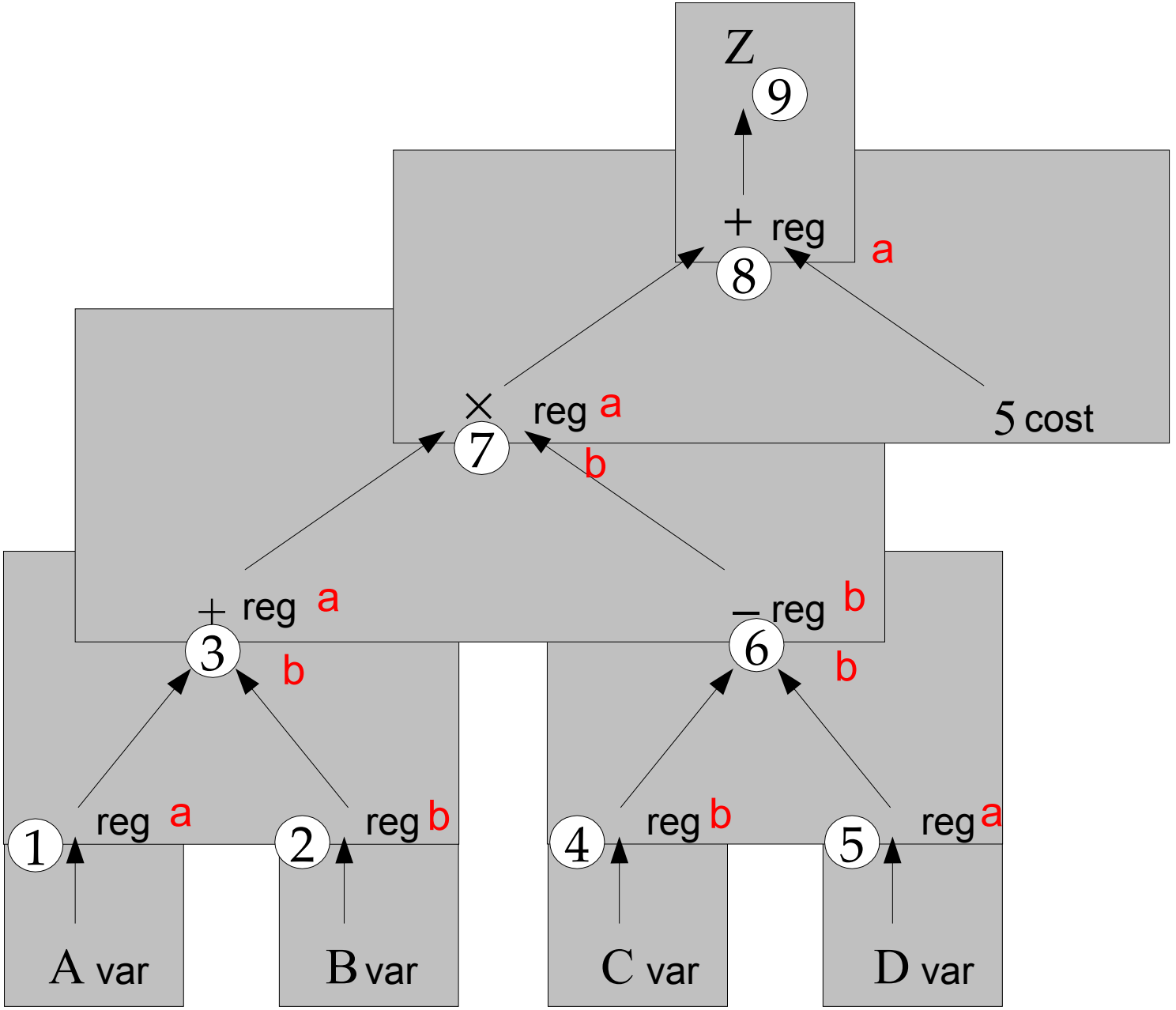


(con 3 parametri, ...)



etc...

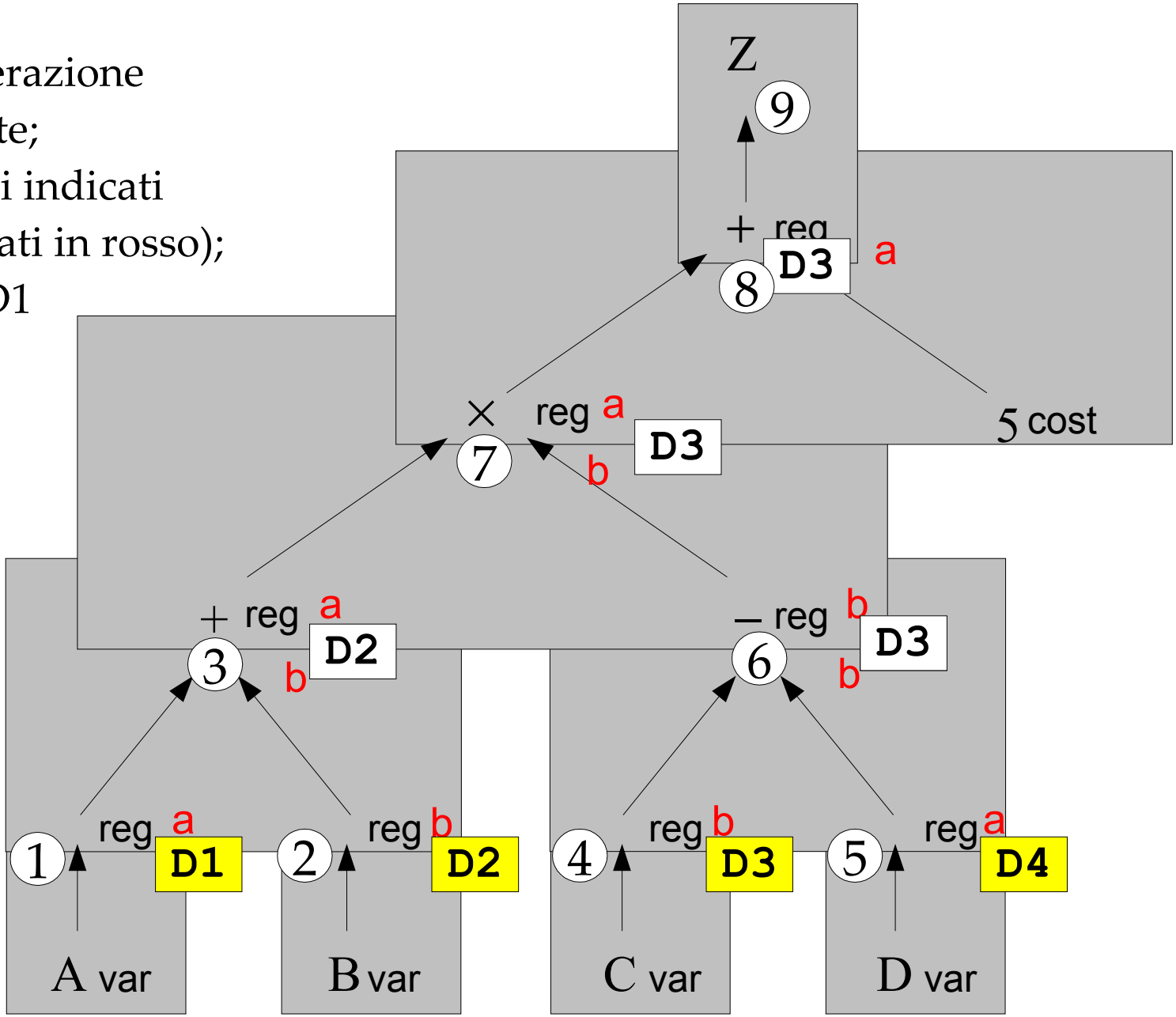
3. Coprire l'albero con le tegole



4. Allocare i registri [TO DO: NO D0]

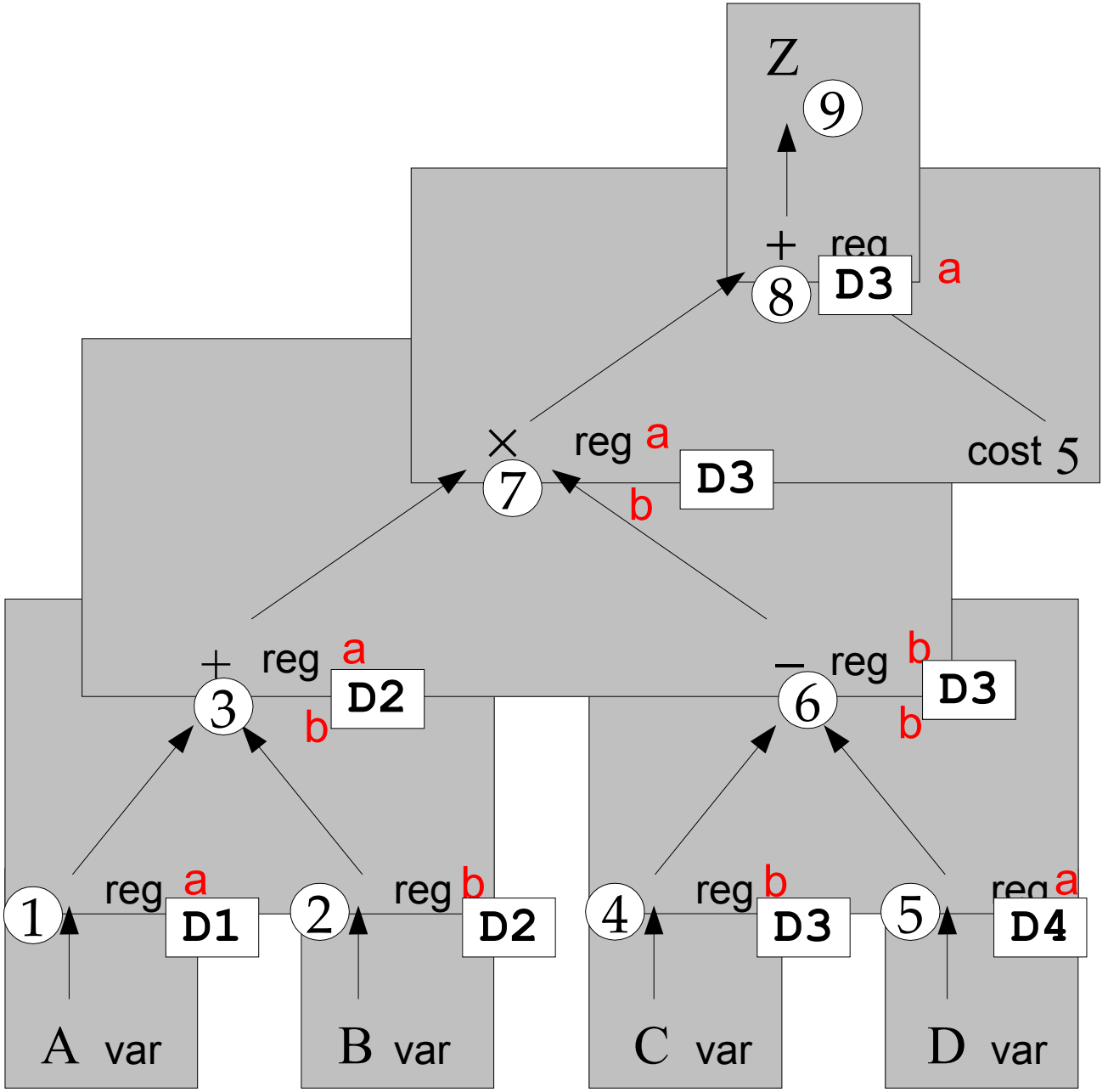
Allocare i registri:

- nell'ordine di numerazione del passo precedente;
- rispettando i vincoli indicati nelle tegole (segnalati in rosso);
- inizio dal registro D1 (D0 è riservato per valori di ritorno);



Nota: i soli registri in giallo sono allocati liberamente; gli altri sono vincolati;

5. Codificare il programma



Codificare in assembly le tegole con i registri allocati:

```
1. MOVE.L A, D1
2. MOVE.L B, D2
3. ADD.L  D1,D2
4. MOVE.L C, D3
5. MOVE.L D, D4
6. SUB.L  D4,D3
7. MULS   D2,D3
8. ADD.L  #5,D3
9. MOVE.L D3,Z
```

Programma finale

Realizzare il programma completo che calcola:

$$Z = (A + B) \times (C - D) + 5$$

con i seguenti valori:

A = 1
B = 2
C = 3
D = –4
Z (calcolata) = 26

Codifica assembly ricavata:

```
1. MOVE.L A, D1
2. MOVE.L B, D2
3. ADD.L  D1,D2
4. MOVE.L C, D3
5. MOVE.L D, D4
6. SUB.L  D4,D3
7. MULS   D2,D3
8. ADD.L  #5,D3
9. MOVE.L D3,Z
```

```
* Allocazione delle variabili, long word
A     EQU  $100
B     EQU  $104
C     EQU  $108
D     EQU  $10C
Z     EQU  $110

START      ORG  $1000
      * Inizializzazione delle variabili
      MOVE.L    #1, A
      MOVE.L    #2, B
      MOVE.L    #3, C
      MOVE.L    #-4,D
      MOVE.L    #0,Z

      * Calcolo dell'espressione
      MOVE.L A, D1
      MOVE.L B, D2
      ADD.L  D1,D2
      MOVE.L C, D3
      MOVE.L D, D4
      SUB.L  D4,D3
      MULS   D2,D3 * Attn: 16+16 bit
      ADD.L  #5,D3
      MOVE.L D3,Z

      STOP #$2000
      END  START
```