1. **Schema generale di traduzione di un programma**

Lo schema generale di un programma puo' essere cosi schematizzato:

- Inizializzazione delle variabili
- Input dei dati
- Elaborazione
- Output dei risultati
- Chiusura del programma

- Procedure di calcolo

A volte alcuni di questi passi sono omessi o fusi tra loro.

2. **Mappatura delle variabili intere sui registri**

Dato un segmento di codice da tradurre occorre prima di tutto dare una corrispondenza tra variabili e registri (nel caso le variabili siano molte occorre definire la corrispondenza tra variabili e zone di memoia ed implementare il register spilling).

Esempio: Dato il seguente segmento di codice:

```
                t1 = 24 ;
                t4 = 2^20 ;
                t0 = t1 + t4 ;
                for (i=0; i<3; i++) {
                    t3 = t3 + ( t0*i )
                }
si individuano  variabili (t1,t4,t0,i e t3).
```

Eseguo la mappatura tra variabili intere e registri.

t1 ↔ $t1
t3 ↔ $t3
t4 ↔ $t4
t0 ↔ $t0
i ↔ $t2

L'assegnazione è arbitraria. Poteva andare bene (anche se notevolmente più caotica) pure la seguente mappatura :

```
# mappatura caotica
     t1 ↔ $t5
     t3 ↔ $s0
     t4 ↔ $a3
     t0 ↔ $v0
      i ↔ $t1
```

3. **Inizializzazione delle variabili**

Una volta definita la mappatura con i registri occorre inizializzare le variabili in maniera corretta rispetto al programma da tradurre. Nei compilatori moderni le formule risolvibili a compile time vengono computate prima della traduzione e viene direttamente implementato il risultato finale.

4. **Traduzione delle formule potenze di due:**

Moltiplicare per potenze di due (2^n) equivale a realizzare uno shift a sinistra di n posizioni.

Esempio: Consideriamo il segmento di codice precedente e la prima mappatura tra variabili intere e registri data.

2^20 = 2*2*2*2*...*2 venti volte equivale a **1=2^0** shiftato a sinistra di 20 volte. E' possibile ottimizzare il codice traducendo la seconda istruzione, **t4=2^20** in un caricamento di una word a 32 bit nella variabile **t4**, **t4=0x0010.0000**.
Oppure se preferite, può essere tradotta in un caricamento di un 1 in t4 seguito da un shift a sinistra di 20 posizioni, **t4 = 1 ; t4 = t4 << 20**.

L'istruzione:

```
t4 = 2^20 ;
```

Può essere quindi tradotta nel seguente modo:

```
addi $t4 , $0 , 1      ;assegno al registro il valore 1
sll $t4 , $t4 , 20     ;shift a sinistra di 20
```

5. **Traduzione di cicli**

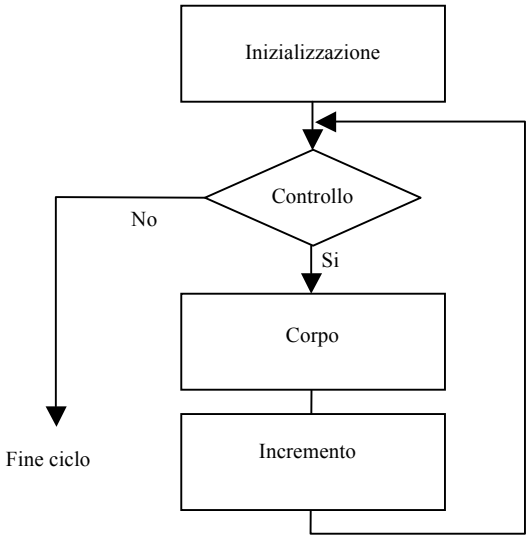
Per tradurre il ciclo FOR dell'esempio possiamo utilizzare il seguente schema (pur non essendo ottimizzato, questo schema ha il pregio di mantenere la struttura del ciclo scritto in C e di essere chiaramente implementabile):

```
for ( INIZIALIZZAZIONE , CONTROLLO, INCREMENTO )
   CORPO
```

e' tradotto in:

```
INIZIALIZZAZZIONE  --> i = 0
CONTROLLO          --> i<3 ? si, vai a CORPO
                           no, vai a fine ciclo
CORPO
INCREMENTO
vai a CONTROLLO
fine ciclo
```

In termini di diagramma di flusso il blocco precedente può essere così schematizzato:



Esempio: Consideriamo il segmento di codice precedente e la prima mappatura tra variabili intere e registri data. Il ciclo:

```
for (i=0; i<3; i++) {
    t3 = t3 + ( t0*i )
}
```

può essere scomposto così:

- Inizializzazione: `i=0`
- Controllo: `i<3`
- Incremento: `i++`

- Corpo: `t3 = t3 + ( t0*i )`

La traduzione seguendo lo schema proposto:

```
for (i=0; i<3; i++) {                    addi $t2, $0, 0      # i=0
                              chkfor:    blt $t2, 3, bodyfor  # i<3?
                                                              # se si, fai ciclo
                                         j endfor             # se no, vai oltre

   t3 = t3 + ( t0*i )         bodyfor:   mult $t0, $t2        # t5 = t0 * i
                                         mflo $t5

                                         add  $t3, $t3, $t5   # t3 = t3 + (to*i)
                                         addi $t2, $t2, 1
                                         j chkfor:
}
                              endfor:    .....
```

BLT, BGT e simili sono pseudo-istruzioni che vengono tradotte in istruzioni SLT e BNE/BEQ dal compilatore rispettando la semantica del confronto. In linea di principio, il confronto all'interno del FOR può essere implementato in qualsiasi modo, purché la semantica del FOR resti inalterata. Nel esempio viene usato BLT poiché era naturale usarlo visto che la condizione è `i<3` (la traduzione risulta più diretta), anche se poi segue un salto che si può ottimizzare se si usa BGE (`bge $t2 , 3 , endfor`).
Stesso discorso vale per l'istruzione BLT con indirizzamento immediato. Nel set delle istruzioni MIPS è presente una istruzione SLTI che usa indirizzamento immediato e che quindi permette di risparmiare un registro in cui andrebbe memorizzato il valore 3. Chiaramente si può usare questo schema quando il limite, 3 in questo caso, è noto al momento di compilare ed è rappresentabile a 16 bit. Negli altri casi bisogna usare un registro di appoggio.

La traduzione completa del segmento di codice C è la seguente:

```
#Segmento di codice C                    # traduzione Assembly MIPS
t1 = 24 ;                                addi $t1 , $0 , 24
t4 = 2^20 ;                              addi $t4 , $0 , 1

                                         sll $t4 , $t4 , 20
t0 = t1 + t4 ;                           add $t0 , $t1 , $t4

for (i=0; i<3; i++) {                    addi $t2, $0, 0          # i=0
                              chkfor:    blt $t2, 3, bodyfor      # i<3?
                                                                  # se si, fai ciclo
                                         j endfor                 # se no, vai oltre

    t3 = t3 + ( t0*i )        bodyfor:   mult $t0, $t2            # t5 = t0 * i
                                         mflo $t5

                                         add  $t3, $t3, $t5       # t3 = t3 + (to*i)
                                         addi $t2, $t2, 1
                                         j chkfor:
}
                              endfor:    .....
```

Nota: Alla fine della terza istruzione la variabile **t0** sarà uguale a **2^20 + 24** , cioè **t0=0x10.0018**. Il successivo ciclo sommerà a **t3** i seguenti valori: **0** , **t0** , **t0*2 = 0x20.0030**, cioè in totale **0x0030.0048**, un numero contenibile in un registro a 32 bit. La variabile **t3** non è inizializzato quindi a priori non so che valore avrà alla fine del codice.

Per completare la specifica del problema sono state fatte queste ulteriori assunzioni:

- poiché non diversamente specificato, e poiché sono presenti solo operazioni tra interi, eseguo tutti i calcoli tra interi a 32 bit con segno.
- poiché non diversamente specificato, ignoro qualsiasi valore precedentemente assegnato a **t3** (la variabile sarà gia inizializzata correttamente dal segmento di codice precedente).