3.2 Costrutti di programmazione

Costrutti di programmazione
In questa unità vengono ripercorsi i principali costrutti della programmazione strutturata e per ciascun costrutto vengono individuate le possibili traduzioni in linguaggio assembly. Per ogni problema sono possibili varie soluzioni con alternative che ottimizzano la lunghezza del codice o la velocità di esecuzione.

Dichiarazione di variabili
Le dichiarazioni di variabili sono realizzate con istruzioni di tipo dichiarativo. Le istruzioni di tipo dichiarativo non genera al momento della traduzione codici macchina di istruzioni ma riservano nella zona "dati statici" del programma un'area di memoria di dimensioni che dipendono dal tipo di dato ed associano all'indirizzo iniziale di tale area il nome simbolico (IDENTIFICATORE) definito nell'istruzione dichiarativa. In questo modo tutti i riferimenti fatti a tale nome simbolico sono riferimenti all'indirizzo di memoria associato. Una successiva dichiarazione, trovando aree già riservate, alloca la propria variabile all'indirizzo immediatamente successivo disponibile.
Le variabili possono appartenere a varie categorie (classi di visibilità):

- **Variabili globali**: sono variabili statiche dichiarate in un modulo e rese accessibili ad ogni altro modulo che può riferirsi al loro nome per accedere in modo univoco alla stessa variabile. In "C" si ottiene una variabile globale dichiarando la variabile al livello più esterno di modulo (fuori di ogni graffa). In Assembly si ottiene una variabile globale inserendo il nome della variabile di modulo in una direttiva PUBLIC. In ogni caso è il LINKER che risolve gli indirizzi delle variabili globali sostituendo al posto degli indirizzi lasciati irrisolti dal compilatore/assemblatore gli indirizzi effettivi. L'identificatore deve essere unico nell'ambito dell'intero progetto altrimenti si ottiene un errore di definizione multipla. Nei passaggi C/ASM si deve tenere conto che le denominazioni interne prevedono sempre la presenza di un underscore (_) che precede l'identificatore. Il "C" lo tratta in modo automatico mentre in Assembly va esplicitamente inserito.
  Esempi di variabili globali:
  - **int a;** se dichiarato al livello più esterno di un modulo "C" la variabile a è accessibile da ogni altro modulo.
    Nei moduli "C" è accessibile attraverso la dichiarazione:
    extern int a;
    nei moduli ASM è accessibile attraverso la dichiarazione
    EXTRN _a:DWORD
  - **TfrmMain *frmMain;** questa dichiarazione, automaticamente generata da BBC in un modulo di finestra, definisce un puntatore statico ad un oggetto dinamico (cioè creato al tempo di esecuzione) derivato da una classe definita nell'header del modulo. Per rendere possibile l'accesso all'oggetto (e a tutte le sue proprietà e metodi) da parte di altri moduli "C" si deve includere l'header del modulo che definisce la classe nel modulo che la deve usare. Infatti l'header, oltre a definire la classe e tutte le sue proprietà e metodi termina con la dichiarazione:
    extern PACKAGE TfrmMain *frmMain;
    che rende disponibile il nome anche al modulo esterno.
  - **PUBLIC _b**
    _b LABEL DWORD
    DD 0
    Questa sequenza crea una variabile pubblica di nome _b ed ampiezza 4 bytes inizializzata a 0. Questa variabile è accessibile in "C" attraverso la dichiarazione:
    extern int b;
- **Variabili locali di modulo**: sono variabili statiche dichiarate in un modulo e rese accessibili solo alle funzioni interne al modulo. In "C" si ottiene una variabile locale di modulo dichiarando la variabile al livello più esterno di modulo (fuori di ogni graffa) preceduta dalla parola chiave **static**. In Assembly si ottiene una variabile locale di modulo NON inserendo il nome della variabile di modulo in una direttiva PUBLIC. Le variabili locali

di modulo sono ignorate dal LINKER quindi l'identificatore non è univoco e si può ripetere in moduli diversi.
Esempi di variabili locali di modulo:

- **static int a;** se dichiarato al livello più esterno di un modulo "C" la variabile a è accessibile solo dalle procedure contenuto nel modulo.
- **_b LABEL DWORD**
  DD 0
  Questa sequenza crea una variabile statica di modulo di nome _b ed ampiezza 4 bytes inizializzata a 0.

- **Variabili locali di procedura**: sono variabili dinamiche dichiarate all'interno di una procedura. La loro visibilità è limitata all'interno della procedura. Si può comprendere questo limite se si pensa che una variabile locale non esiste sempre durante l'esecuzione del programma ma viene creata allocando spazio nello stack da parte della procedura nel momento in cui entra in esecuzione e viene rimossa al termine della esecuzione della procedura. Durante il periodo di vita della variabile solo la procedura che l'ha generata conosce la posizione in cui la variabile si trova. Le variabili locali di modulo sono ignorate dal LINKER quindi l'identificatore non è univoco e si può ripetere in moduli diversi e in procedure diverse dello stesso modulo.
  Esempi di variabili locali:
  - **void fun(void) {**
    int a;
    ...
    a=0;
    }
    La variabile a è interna alla funzione fun. Quando la funzione fun entra in esecuzione alloca uno spazio di quattro bytes in stack e lo usa per gli accessi alla variabile a non inizializzata. Prima di terminare la funzione fun rimuove lo spazio allocato dallo stack.
  - **_fun PROC**
    PUSH EBP
    MOV EBP,ESP
    SUB ESP,4
    ...
    MOV [EBP-4],0
    ...
    ADD ESP,4
    POP EBP
    _fun ENDP
    Dopo avere salvato in stack in vecchio valore di EBP e avere allineato EBP alla cima attuale dello stack per avere accesso ai parametri la funzione _fun sposta verso l'alto di quattro byte la cima dello stack creando quindi una nuova cima in grado di ospitare un dato di quattro bytes (ad esempio un intero) non inizializzato. L'accesso alla variabile avviene attraverso un accesso indiretto tramite EBP spostato di 4 perché EBP non è stato spostato. Prima di terminare la procedura rimuove la variabile locale spostando verso il basso la cima dello stack di 4 bytes.

Valutazione di una espressione
La maggior parte delle istruzioni esecutive è costituita da espressioni. Una espressione è l'indicazione delle operazioni da effettuare sugli operandi per ottenere un risultato. La valutazione di una espressione è il processo di elaborazione che l'esecutore fa applicando agli operandi gli operatori. La valutazione di una espressione produce sempre un valore il cui tipo dipende dal tipo degli operandi. Il risultato di una valutazione può essere assegnato ad una variabile oppure utilizzato come condizione in un costrutto di scelta o ciclico.
Esempi di espressione:

- costante: 1, 'A', "abc",0.5, true --> la valutazione produce la costante stessa
- variabile: var1, vet[0] --> la valutazione produce il contenuto della variabile
- espressione aritmetica: (var1 *2)/var2 --> la valutazione produce un valore risultato della applicazione degli operandi agli operatori. Il tipo dipende dai tipi delle variabili

- espressione bit a bit: (var1 & 0x01)|(var2 & 0x80) --> la valutazione produce un valore risultato della applicazione degli operandi agli operatori. Il tipo dipende dai tipi delle variabili
- espressione logica: var1 && var2 --> è la combinazione di due espressioni di qualunque tipo con un operatore logico. I risultati delle valutazioni componenti vengono usati come variabili logiche dall'operatore booleano. Per comprendere la differenza rispetto all'operatore precedente consideriamo il seguente esempio:
  se a=1; b=2; a & b produce 0 (false) mentre a && b produce 1 (true)
- espressione relazionale: var1 > 1 --> è la combinazione di due espressioni di qualunque tipo con un operatore relazionale. I risultati delle valutazioni componenti vengono usati come operandi dell'operatore relazionale che produce sempre un risultato booleano.

Costrutti di scelta
I costrutti di scelta consentono di alterare il flusso sequenziale del programma eseguendo segmenti di programma (sequenze) in modo condizionato in base alla valutazione di una espressione.

Scelta semplice
Il costrutto di scelta semplice consente di eseguire o non eseguire una sequenza in base al risultato della valutazione di una espressione:

se (valutazione di una espressione)
<sequenza>
finese

Il risultato della valutazione dell'espressione indipendentemente dal tipo viene considerato un booleano e quindi e il risultato è diverso da 0 (true) <sequenza> viene eseguita altrimenti viene saltata.
Esempio 1: valutazione di una variabile booleana

codifica C	Codifica ASM
bool var; **...** **if (var) {** **<sequenza>** **}**	**EXTRN _var:DWORD** **...** **MOV EAX,_var** **AND EAX,EAX** **JZ fine** **<sequenza>** **fine: ...**

- La forma completa della espressione in C sarebbe (var==true) tuttavia poiché la variabile è già booleana il confronto non è necessario.
- Nella codifica ASM la lettura della variabile non produce l'alterazione dei flag che vanno aggiornati con una operazione logica che non altera il contenuto del registro.
- La condizione di salto è quello opposta alla condizioni che viene valutata in C; infatti mentre la condizione C indica che la variabile deve essere vera il salto avviene quando la variabile è falsa (JZ)

Esempio 2: valutazione di una espressione booleana

Codifica C	Codifica ASM
Bool var1,var2; **...** **if (var1 & var2) {** **<sequenza>** **}**	**EXTRN _var1,_var2:DWORD** **...** **MOV EAX,_var1** **AND EAX,_var2** **JZ fine** **<sequenza>** **fine: ...**

- L'espressione deve essere valutata prima di effettuare il salto condizionato.

Esempio 3: valutazione di una espressione relazionale

Codifica C	Codifica ASM
Bool var **...** **if (var < 10) {** **<sequenza>** **}**	**EXTRN _var:DWORD** **...** **MOV EAX,_var** **CMP EAX,10** **JNC fine** **<sequenza>** **fine: ...**

Esempio 4: valutazione di una espressione relazionale

Codifica C	Codifica ASM
Bool var **...** **if (var > 10) {** **<sequenza>** **}**	**EXTRN _var:DWORD** **...** **MOV EAX,_var** **CMP EAX,10** **JC fine** **JZ fine** **<sequenza>** **fine: ...**

Esempio 5: valutazione di una espressione relazionale con due espressioni aritmetiche

Codifica C	Codifica ASM
Bool var **...** **if (var1+2 < var2/4) {** **<sequenza>** **}**	**EXTRN _var1,_var2:DWORD** **...** **MOV EAX,_var1** **ADD EAX,2** **MOV EBX,_var2** **SAR EBX,2** **CMP EAX,EBX** **JNC fine** **<sequenza>** **fine: ...**

Scelta doppia

Il costrutto di scelta doppia consente di eseguire in alternativa una delle due sequenze che formano il costrutto in base al risultato della valutazione di una espressione che viene sempre considerato booleano:

se (valutazione di una espressione)
  <sequenza_1>
altrimenti
  <sequenza_1>
finese

Il risultato della valutazione dell'espressione indipendentemente dal tipo viene considerato un booleano e quindi se il risultato è diverso da 0 (true) <sequenza_1> viene eseguita e poi si salta alla fine altrimenti <sequenza_1> viene saltata e viene eseguita <sequenza_2>.

Esempio: viene riportato solo l'esempio con una singola variabile booleana, per gli altri casi fare riferimento ai casi della scelta semplice.

Codifica C	Codifica ASM
bool var; **...** **if (var) {** **<sequenza_1>** **}** **else {** **<sequenza_2>** **}**	**EXTRN _var:DWORD** **...** **MOV EAX,_var** **AND EAX,EAX** **JZ altrimenti** **<sequenza_1>** **JMP fine** **altrimenti:** **<sequenza_2>** **fine: ...**

- La struttura di controllo è uguale a quella della scelta semplice.
- Quando la condizione è soddisfatta si salta quindi in assembly il test è rovesciato rispetto al suo aspetto in C.
- Al termine di <sequenza_1> ci deve essere un salto incondizionato alla fine altrimenti verrebbe eseguita **anche** <sequenza_2>

Scelta multipla

Il costrutto di scelta multipla doppia consente di eseguire una sequenza tra le molte che formano il costrutto in base al risultato della valutazione di una espressione che viene sempre considerato intero:

scelta (valutazione di una espressione)
caso 1:
<sequenza_1>
caso 2:
<sequenza_2>
...
altrimenti:
<sequenza_altrimenti>
finescelta

Esempio 1: realizzato come una sequenza di if

Codifica C	Codifica ASM
int var; **...** **switch (var) {** **case 0:** **<sequenza_1>** **break;** **case 1:** **<sequenza_2>** **break;** **case N:** **<sequenza_N>** **break;** **default:** **<sequenza_altrimenti>** **}**	**EXTRN _var:DWORD** **...** **MOV EAX,_var** **CMP EAX,0** **JZ caso0** **CMP EAX,1** **JZ caso1** **...** **CMP EAX,N** **JZ casoN** **JMP altrimenti** **caso0: <sequenza_0>** **JMP fine** **caso1: <sequenza_1>** **JMP fine** **casoN: <sequenza_N>** **JMP fine** **altrimenti:** **<sequenza_altrimenti>** **fine: ...**

- La valutazione è costituita da una sequenza di test sulla stessa variabile, il primo test che soddisfa la condizione determina il salto.
- In questo esempio emerge l'importanza del fatto che CMP non altera gli operandi, in questo modo non è necessario ricaricare ogni volta il dato.

- Se nessun caso è soddisfatto un salto incondizionato porta al defalut.
- Ogni caso deve terminare con un salto incondizionato altrimenti verrebbe eseguito anche il caso successivo.

Esempio 2: realizzato come un salto indiretto

Codifica C	Codifica ASM
int var; **...** **switch (var) {** **case 0:** **<sequenza_1>** **break;** **case 1:** **<sequenza_2>** **break;** **case N:** **<sequenza_N>** **break;** **default:** **<sequenza_altrimenti>** **}**	**EXTRN _var:DWORD** **...** **MOV EAX,_var** **CMP EAX,N+1** **JNC altrimenti** **MOV EBX,OFFSET tabella** **ADD EBX,EAX** **JMP [EBX]** **tabella:DD caso0** **DD caso1** **...** **DD casoN** **caso0: <sequenza_0>** **JMP fine** **caso1: <sequenza_1>** **JMP fine** **casoN: <sequenza_N>** **JMP fine** **altrimenti:** **<sequenza_altrimenti>** **fine: ...**

- La valutazione è costituita da una sequenza di test sulla stessa variabile, il primo test che soddisfa la condizione determina il salto.
- In questo esempio emerge l'importanza del fatto che CMP non altera gli operandi, in questo modo non è necessario ricaricare ogni volta il dato.
- Se nessun caso è soddisfatto un salto incondizionato porta al defalut.
- Ogni caso deve terminare con un salto incondizionato altrimenti verrebbe eseguito anche il caso successivo.

Costrutti iterativi

I costrutti iterativi consentono di ripetere una sequenza più volte alterando il flusso sequenziale del programma attraverso dei salti condizionati in base alla valutazione di una espressione.

Ciclo mentre

Il costrutto mentre consente di ripetere una sequenza per una quantità indeterminata di volte uscendo dalla iterazione solo quando la valutazione della espressione di controllo produce un risultato falso. Poiché il test è posto all'inizio del ciclo le variabili usate per la valutazione devono essere inizializzate prima di entrare nel ciclo ed è possibile che il ciclo venga eseguito 0 volte. La sequenza deve, dopo un numero finito di iterazioni, alterare i valori usati dall'espressione per garantire la terminazione del ciclo.

mentre (valutazione dell'espressione)
  <sequenza che modifica i valori dell'espressione>
finementre

Esempio: variabile di controllo booleano. Per gli altri casi fare riferimento alla scelta semplice.

Codifica C	Codifica ASM
bool var; **...** **while (var) {** ** <sequenza che modifica var>** **}**	**EXTRN _var:DWORD** **...** **MOV _var,1** **loop: MOV EAX,_var** **CMP EAX,0** **JZ fine** **<sequenza che modifica var>** **JMP loop** **fine: ...**

- La valutazione dell'espressione determina sempre un risultato booleano
- Il test è all'inizio quindi si può subito uscire senza fare alcuna iterazione se l'espressione è già falsa.
- Al termine della sequenza un salto incondizionato riporta alla valutazione del loop.
- La condizione in asm è rovesciata rispetto al C.

Ciclo iterazioni numerate

Il costrutto iterazioni numerate consente di ripetere una sequenza per una quantità predefinita di volte attraverso un conteggio delle iterazioni uscendo dalla iterazione quando il conteggio è completato. Dal punto di vista strutturale non differisce in alcun modo da un costrutto ***mentre*** in cui la variabile di controllo è una espressione che verifica il raggiungimento del fine conteggio.

per (contatore=valore iniziale; confronto con il valore finale; modifica del contatore)
<sequenza>
fineper

Esempio:

Codifica C	Codifica ASM
int var; **...** **for (var=0; var<MAX; var++) {** ** <sequenza>** **}**	**EXTRN _var:DWORD** **...** **MOV _var,0** **loop: MOV EAX,_var** **CMP EAX,10** **JNC fine** **<sequenza>** **INC _var** **JMP loop** **fine: ...**

- Il costrutto for è un caso particolare del costrutto while in cui la variabile di controllo è un contatore.

Ciclo ripeti

Il costrutto ripeti consente di ripetere una sequenza per una quantità indeterminata di volte uscendo dalla iterazione solo quando la valutazione della espressione di controllo produce un risultato falso. Differisce dal costrutto mentre per il test è posto alla fine del ciclo. Le variabili usate per la valutazione non necessitano di inizializzazione perché la sequenza viene eseguita almeno una volta.La sequenza deve, dopo un numero finito di iterazioni, alterare i valori usati dall'espressione per garantire la terminazione del ciclo.

ripeti (valutazione dell'espressione)
  <sequenza che modifica i valori dell'espressione>
mentre (valutazione dell'espressione)

Esempio: variabile di controllo booleano. Per gli altri casi fare riferimento alla scelta semplice.

Codifica C	Codifica ASM
bool var; **...** **do {** ** <sequenza che modifica var>** **} while (var);**	**EXTRN _var:DWORD** **...** **loop: <sequenza che modifica var>** **MOV EAX,_var** **CMP EAX,0** **JNZ loop** **fine: ...**

- La valutazione dell'espressione determina sempre un risultato booleano
- Il test è alla fine quindi si deve sempre eseguire almeno una iterazione
- La condizione in asm non è rovesciata rispetto al C.