# Calcolatori Elettronici
# Programmazione a moduli

Ing. Gestionale e delle Telecomunicazioni

A.A. 2008/09

Gabriele Cecchetti

## Sommario

- Programmazione a moduli
- Programmi con linguaggi misti
- Tempo di vita delle variabili e ambiente
- Riferimento all'ambiente
- Record di attivazione, creazione e distruzione
- Interfacce chiamante e chiamato
- Parametri, variabili locali e valori restituiti
- Esempio di traduzione da C ad Assembler

# Moduli Assembler

- Programma Assembler:
  - Costituito da uno o più moduli
- Modulo:
  - Insieme di entità che stanno su il file sorgente
  - Ogni modulo viene tradotto separatamente dagli altri moduli, dando luogo ad un unico file: il file oggetto
- Traduzione:
  - Avviene attraverso il programma Assemblatore.
- Collegamento:
  - I file oggetti vengono collegati insieme da un programma Collegatore ("*Linker*") per formare un file eseguibile.
- Caricamento
  - Il caricamento e l'eventuale rilocazione di un programma eseguibile vengono eseguiti da un particolare programma detto *Loader*.



# Punto di inizio

- Modulo Assembler principale:
  - contiene il simbolo `start:` ;
  - contiene il programma principale e alcuni sottoprogrammi;
  - è unico.
- Gli altri moduli sono i moduli secondari contengono solo sottoprogrammi.
- In un modulo, i nomi non globali sono propri di quel modulo.

# Programmazione con linguaggi misti

- Un programma organizzato a moduli:
  - può essere scritto utilizzando linguaggi differenti per i vari moduli;
  - ciascun modulo viene tradotto con il proprio traduttore, e tutti i traduttori producono moduli oggetto della stessa forma;
  - viene utilizzato un unico *Collegatore* che produce il programma eseguibile.
- Intero programma:
  - consistente solo se i differenti linguaggi utilizzati fanno uso dello stesso standard, sia per la rappresentazione dati, sia per l'aggancio con i sottoprogrammi.



# Programmi C/C++

- Programma C/C++
  - modulo principale che contiene la dichiarazione della funzione *main*;
  - eventuali moduli secondari.
- Modulo C/C++
  - blocco: tutto ciò che è racchiuso tra {…}
  - particolari blocchi: corpi di funzioni.
- Funzione C/C++
  - può restituire un valore,
  - può essere di tipo *void* (procedura),
  - si utilizza il termine sottoprogramma.

# Esecuzione di un sottoprogramma

- Il sottoprogramma
  - viene chiamato mediante un designatore di funzione;
  - i parametri attuali dipendono dalle regole di visibilità del blocco chiamante;
  - il chiamato ha regole di visibilità del blocco corrispondente, che possono essere diverse da quelle del blocco chiamante.
- Il sottoprogramma può essere eseguito più volte
  - diverse istanze chiamate più volte anche se le istanze precedenti non sono ancora terminate.



# Tempo di vita delle variabili

- Variabili globali
  - hanno lo stesso tempo di vita del programma (var. statiche).
- Variabili locali di un sottoprogramma:
  - esistono per il tempo di esecuzione del sottoprogramma (var. automatiche).

# Ambiente

- Varia da istanza a istanza, anche nell'ambito di uno stesso sottoprogramma
- Ha la stessa forma per tutte le istanze del sottoprogramma:
  - stesso numero e tipo di parametri attuali;
  - stesso numero e tipo di variabili locali.



# Esempio di ambiente

```
void primo (int a) {
   int aa;
   aa=1; a=2+a; …
}

void secondo (int b) {
   int bb;
   primo (b);
   primo (bb); …
}
```

- Pur rimanendo invariati i parametri attuali di primo i loro valori cambiano a seconda dell'istanza.

# Riferimento all'ambiente

- Il sottoprogramma ha una singola copia di codice.
- L'istanza di un sottoprogramma ha un proprio ambiente.
- Il codice deve riferire:
  - tutti i sottoprogrammi (sempre globali);
  - le variabili globali (ambiente globale);
  - i parametri attuali e le variabili locali di quella istanza (ambiente locale).
- Oggetto globale:
  - si riferisce con un indirizzo assoluto.
- Oggetto locale:
  - si riferisce con un indirizzo relativo, costituito da un indirizzo base che individua l'ambiente locale attuale, e da uno spiazzamento (*displacement*), che individua l'oggetto dentro l'ambiente e che è invariante per tutte le istanze.



# Record di attivazione

- Istanza di sottoprogramma
  - costruzione sulla pila di un record di attivazione, che contiene le informazioni utili per quella istanza.
- **Record di attivazione**
  - utilizza un registro puntatore che contiene l'indirizzo base del record e che individua l'ambiente locale.
- Costruzione di un nuovo record di attivazione
  - il valore del registro puntatore viene salvato e successivamente ripristinato quando il record di attivazione viene costruito e distrutto.

## Informazioni contenute nel record di attivazione

1. ***link dinamico***, ossia puntatore destinato a memorizzare il vecchio valore del registro puntatore;
2. ***indirizzo di ritorno al sottoprogramma*** chiamante (tale indirizzo viene messo in pila per effetto dell'istruzione di chiamata)
3. ***parametri attuali***;
4. ***spazio per le variabili locali***.



## Costruzione del record di attivazione

- Iniziata dal sottoprogramma chiamante.
- Completata dal sottoprogramma chiamato, mediante apposito prologo.

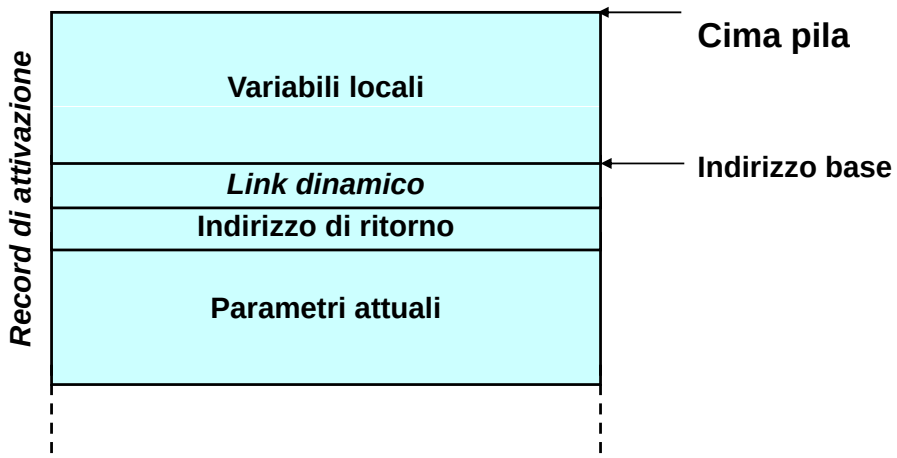
- Il **chiamante** inserisce in pila:
  - i parametri attuali,
  - l'indirizzo di ritorno (istruzione CALL).
- Il **chiamato** inserisce in pila:
  - il link dinamico,
  - lo spazio per le variabili locali.

# Indirizzo base

- Stabilito dal chiamato.
- Corrisponde al punto che separa i parametri attuali dalle variabili locali (coincide con l'indirizzo del link dinamico del record).
- I parametri attuali vengono riferiti con displacement positivi e le variabili locali con displacement negativi.



# Schema del record di attivazione



## Distruzione del record di attivazione

- Iniziata dal chiamato (mediante un apposito epilogo).
- Completata dal chiamante.

- Il **chiamato** rimuove dalla pila:
  - lo spazio per le var. locali,
  - il *link* dinamico,
  - l'indirizzo di ritorno (istruzione RET).
- Il **chiamante** rimuove dalla pila:
  - lo spazio per i parametri attuali.



## Rappresentazione dei dati

- Per il processore 8086:
  - var. di tipo **`char`**: **1 byte**;
  - var. di tipo **`int`**: **2 byte**;
  - var. di tipo **`float`**: **4 byte** (standard IEEE-754).

- Nota:
  - Non trattiamo gli altri tipi di variabili.
  - Consideriamo solo i parametri trasmessi per valore.

# Valori restituiti da funzioni

- Una funzione *non void* restituisce un valore in:
  - **AL**, *se il valore è di tipo carattere*;
  - **AX**, *se il valore è di tipo intero*.



# Interfaccia del chiamante

- Quando deve essere eseguito un sottoprogramma, il chiamante effettua delle
  - azioni iniziali:
    - **immissione in pila dei parametri attuali** in *ordine inverso* a quello che compare nella chiamata;
    - esecuzione dell'**istruzione di chiamata**.
  - azioni di ritorno:
    - **eliminazione dalla pila** dello spazio occupato **dai parametri attuali**.

# Interfaccia del chiamato

- Un sottoprogramma (chiamato) effettua le seguenti azioni di interfaccia:
  - *prologo*:
    - **salvataggio in pila del contenuto del registro BP** (costruzione del link dinamico);
    - **copia nel registro BP del valore attuale del registro SP** (indirizzo base del record di attivazione attuale);
    - riserva di **spazio in pila per variabili locali**.
  - *epilogo*:
    - **eliminazione dalla pila dello spazio riservato alle var. locali**;
    - **ripristino del contenuto del registro BP**;
    - effettuazione del ritorno al chiamante (istruzione RET).



# Riferimento a parametri e variabili (sottoprogramma chiamato)

- Dopo l'esecuzione del prologo risulta completamente costruito il record di attivazione di quella istanza del sottoprogramma.
- Indirizzi relativi:
  - i parametri attuali e le variabili locali vengono riferiti con indirizzamento a registro base rispetto a BP (displacement positivi per i parametri e negativi per le var. locali);
- Indirizzi assoluti:
  - solo per riferire l'ambiente locale.
- Nota:
  - tutte le variabili risiedono nella pila;
  - Nel corpo del sottoprogramma si possono temporaneamente utilizzare tutti i registri generali (tranne BP e SP), provvedendo inizialmente a salvare il loro contenuto a alla fine a ripristinarlo.

## Prologo sottoprogramma chiamato: forma Assembler

```
PUSH BP     ; link dinamico
            ;
MOV BP, SP  ; indirizzo base record di
            ; attivazione
SUB SP, n   ; spazio per gli n byte
            ; delle var. locali
```



## Epilogo sottoprogramma chiamato: forma Assembler

```
MOV SP, BP  ; Eliminazione spazio per
  le
            ; var. locali
POP BP      ; Ripristino del vecchio
            ; indirizzo base
RET         ; Ritorno al chiamante
```

# Indirizzamenti del sottoprogramma chiamato: forma Assembler

- I parametri vengono riferiti come:
  - $PAR_i$= 4, 6, …

    `MOV AX, PAR`$_i$`[BP] ; ad es.  MOV 4[BP], AX`

- Le variabili locali vengono riferite come:
  - $VAR_j$= -2, -4, …

    `MOV AX, VAR`$_j$`[BP] ; ad es.  MOV AX, -2[BP]`



# Interfaccia del chiamante: forma Assembler

- Parametri attuali del chiamante:
  - valore di una var. globale intera:

    **`PUSH nomevar`**
  - indirizzo di una var. globale intera:

    **`LEA AX, nomevar`**

    **`PUSH AX`**
  - Istruzione di chiamata del sottoprogramma:

    **`CALL sottoprogramma`**
  - Eliminazione spazio per parametri (*n* byte)

    **`ADD SP, n`**

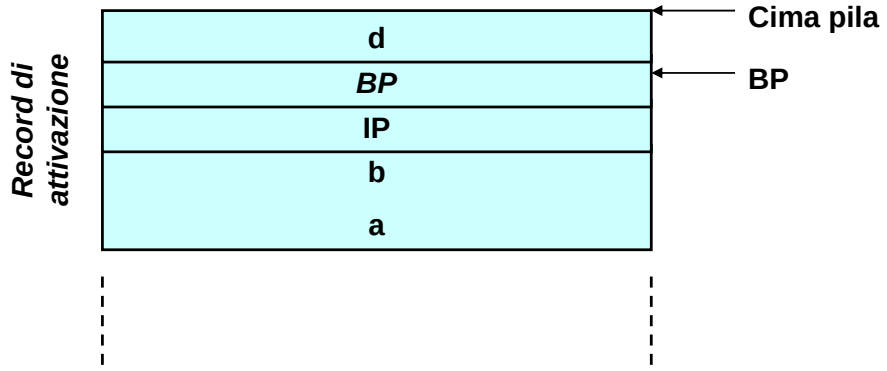
# Esempio: programma C

```
int alfa, beta;

int fai(int a, int b)
{ int d, e;

  e = 1;
  d = a + b;
  d++;
  return d;
}
```

```
int main()
{    int i;

  alfa = 2;
  beta = 3;
  i = fai(alfa, beta);
  return i;
}
```



# Record di attivazione di *fai*



# Versione assembler

```
;int alfa, beta;
alfa      DW 1 DUP (?)
beta      DW 1 DUP (?)
;int fai(int a, int b)
;{  int d, e;
fai proc
; Prologo
 PUSH BP   ; Salvataggio del vecchio BP
 MOV BP,SP ; Copia SP in BP
 SUB SP,2  ; Allocazione var. locale d
;   e = 1;
 MOV AX, 1    ; AX ← 1
 MOV -2[BP],AX ; Copia AX in e
;   d = a + b;
 MOV AX,6[BP]  ; Copia a in AX
 ADD AX,4[BP]  ; Somma b ad AX
 MOV -4[BP],AX ; Lascia il risultato
               ; della somma in d
;   d++;
 MOV AX,-4[BP] ; Copia d in AX
 INC AX        ; Incrementa AX
 MOV -4[BP],AX ; Copia AX in d
;   return d;
 MOV AX,-4[BP] ; Restituisce al chiamante d
    tramite AX
; } Epilogo
 MOV SP,BP ; De-alloca le variabili locali
 POP BP    ; Recupera il vecchio BP
 RET       ; Ritorna al chiamante
fai endp
```

```
;int main()
;{  int i;
main proc
; Prologo
 PUSH BP     ; Salvataggio del vecchio BP
 MOV BP,SP   ; Copia SP in BP
 SUB SP,#2   ; Allocazione var. locale d
;   alfa = 2;
 MOV AX,2    ; Copia 2 in AX
 MOV alfa,AX ; Copia AX in alfa
;   beta = 3;
 MOV AX,3    ; Copia 3 in AX
 MOV beta,AX ; Copia AX in beta
;   i = fai(alfa, beta);
 MOV AX,alfa ; Copia alfa in AX
 PUSH AX ; Salva AX in pila (param. attuale)
 MOV AX,beta ; Copia beta in AX
 PUSH AX ; Salva AX in pila (param. attuale)
 CALL fai  ; Chiama 'fai' (salva in pila IP)
 ADD SP,#4 ; De-alloca dalla pila i param.
             attuali
 MOV -2[BP],AX ; Copia il valore ritornato
                 in i
;   return i;
 MOV AX,-2[BP] ; Restituisce al chiamante i
                 tramite AX
;} Epilogo
 MOV SP,BP ; De-alloca le variabili locali
 POP BP    ; Recupera il vecchio BP
 RET       ; Ritorna al chiamante
main endp
start:
    CALL main ; Chiama la funzione main
```



# Vers. assembl. n. 2

```
;int alfa, beta;
alfa      DW 1 DUP (?)
beta      DW 1 DUP (?)
;int fai(int a, int b)
;{  int d, e;
fai proc
; Prologo
 PUSH BP   ; Salvataggio del vecchio BP
 MOV BP,SP ; Copia SP in BP
 SUB SP,2  ; Allocazione var. locale d
;   e = 1;
 MOV AX, 1     ; AX ← 1
 MOV -2[BP],AX ; Copia AX in e
;   d = a + b;
 MOV AX,6[BP]  ; Copia a in AX
 ADD AX,4[BP]  ; Somma b ad AX
 MOV -4[BP],AX ; Lascia il risultato
               ; della somma in d
;   d++;
 MOV AX,-4[BP] ; Copia d in AX
 INC AX        ; Incrementa AX
 MOV -4[BP],AX ; Copia AX in d
;   return d;
 MOV 6[BP],-4[BP] ; Restituisce al chiamante
    d tramite la posizione del 1° par. form.
; } Epilogo
 MOV SP,BP ; De-alloca le variabili locali
 POP BP    ; Recupera il vecchio BP
 RET       ; Ritorna al chiamante
fai endp
```

```
;int main()
;{  int i;
main proc
; Prologo
 PUSH BP     ; Salvataggio del vecchio BP
 MOV BP,SP   ; Copia SP in BP
 SUB SP,#2   ; Allocazione var. locale d
;   alfa = 2;
 MOV AX,2    ; Copia 2 in AX
 MOV alfa,AX ; Copia AX in alfa
;   beta = 3;
 MOV AX,3    ; Copia 3 in AX
 MOV beta,AX ; Copia AX in beta
;   i = fai(alfa, beta);
 MOV AX,alfa ; Copia alfa in AX
 PUSH AX     ; Salva AX in pila
 MOV AX,beta ; Copia beta in AX
 PUSH AX     ; Salva AX in pila
 CALL fai    ; Chiama 'fai'
 MOV -2[BP],-4[BP] ; Copia il valore
                     ritornato in i
 ADD SP,#4   ; De-alloca dalla pila i param.
               formali
;   return i;
 MOV AX,-2[BP]; Restituisce al chiamante i
                tramite AX
;} Epilogo
 MOV SP,BP ; De-alloca le variabili locali
 POP BP    ; Recupera il vecchio BP
 RET       ; Ritorna al chiamante
main endp
start:
    CALL main ; Chiama la funzione main
```