Realizzazione delle strutture di controllo in Assembly

In Assembly non esistono le strutture potenti offerte da un HLL, ma è possibile renderle con le istruzioni di controllo Assembly

Strutture di controllo in linguaggio HL	Istruzioni in linguaggio Assembly
• if-then-else	• slt
• while	• beq,bne
• do-while	• bgezal
• for	• j

Nella realizzazione Assembly delle strutture di controllo sarà necessario gestire esplicitamente molti aspetti che nell'HLL sono risolti in maniera implicita ed automatica.

Realizzazione di costrutti di selezione (if-then)

Con le istruzioni di confronto e le istruzioni per il controllo di flusso è possibile realizzare costrutti di selezione del tipo di quelli usati nei linguaggi ad alto livello:

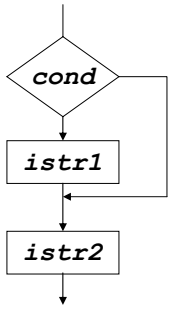
```
if (cond)
  istr1;
istr2;
```

Esempio:

```
if (a>max)
  max=a;
```

```
      if (a<=b)goto next
then  max=a;
next  ...
```

```
      lw   $s0,a
      lw   $s1,max
      slt  $at,$s1,$s0
      beq  $at,$0,next
then  sw   $s0,max
next  ...
```

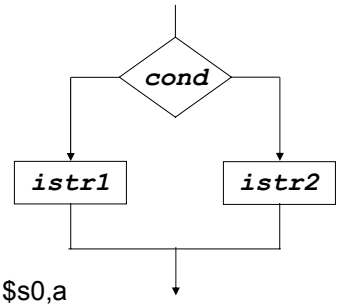


Realizzazione di costrutti di selezione (if-then-else)

Esempio:

```
if (cond)
  istr1;
else
  istr2;
```

```
if (a>b)
  max=a;
else
  max=b;
```




```
      if (a<=b)goto else
then  max=a;
      goto next
else  max=b;
next  …
```

```
      lw   $s0,a
      lw   $s1,b
      slt  $at,$s1,$s0
      beq  $1,$0,else
then  sw   $s0,max
      j    next
else  sw   $s1,max
next  ...
```


Costrutti di selezione

- La condizione per il confronto è opposta alla relazione d'ordine nell'`if` perché il salto si effettua verso il blocco sotto l'`else`
- L'uso di `slt` implica che `a,b` e `max` contengano numeri signed. Nel caso fossero stati numeri unsigned si sarebbe dovuto usare `sltu`.

…e nel caso di relazioni più complesse ?

```
if (a>=min && a<=max)
      {Blocco 1}
else
      {Blocco 2}
```

```
      if (a<min) goto else
      if (a>max) goto else
then  {Blocco 1}
      goto next
else  {Blocco 2}
next  …
```

Strutture di controllo: While

Il ciclo while può essere realizzato tramite l'uso contemporaneo di salti condizionati e salti incondizionati

```
while (a>b)
  {Blocco}
```

```
while if (a<=b)goto next
      {Blocco}
      goto while
next  …
```

```
      lw   $s0,a
      lw   $s1,b
while slt  $at,$s1,$s0
      beq  $at,$0,next
      #{Blocco}
      j    while
next  ...
```


Strutture di controllo: For

```
for(i=0;i<10;i++)
  {Blocco}
```

```
      ifin=10
      i=0
for
      {Blocco}
      i=i+1
      if(i<ifin) goto for
```

```
      ori $s1,$0,10      # imax=10
      ori $s2,$0,1       # step=1
      ori $s0,$0,0       # i=0
for
#
#     {Blocco}
#
      add $s0,$s0,$s2    # i=i+step
      slt $s3,$s0,$s1
      bne $s3,$0,for
```

Strutture di controllo: For

Nel caso in cui nel ciclo non sia necessario accedere alla variabile di conteggio, ma occorra solo assicurare il numero di iterazioni, sono possibili altre soluzioni.

```
      i=9
for
      {Blocco}
      i=i-1
      if(i>=0) goto for
```

```
      ori $s2,$0,9
for
#
#     {Blocco}
#
      addi $s2,$s2,-1
      bgez $s2,for
```


Realizzazione di strutture dati in Assembly

Gli unici tipi di dato in Assembly sono BYTE, HALFWORD, WORD.

Non esistono dati strutturati.

E' necessario gestire esplicitamente tutti gli aspetti relativi alla struttura dati: definizione ed accesso.

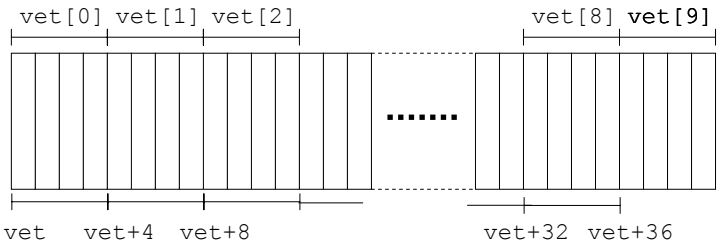
HLL	*Assembly*
Nome	Indirizzo
Tipo	Dimensione (# bytes)
Numero di elementi	Numero di bytes allocati

Strutture dati: Array(1)

Definizione

HLL ***Assembly***

```
int vet[10];          vet   .space 40
```



Strutture dati: Array (2)

Accesso

Bisogna costruire la relazione tra l'indice dell'array (compreso tra 0 e N-1) e la locazione di memoria corrispondente.

HLL ***Assembly***

```
                    ($s3↔x $s0↔i)
vet[i]=x;           add $t1,$s0,0    # calcola l'indirizzo
                    mul $t1,$t1,4
                    sw  $s3,vet($t1)# vet[i]=x
```

Per un accesso sequenziale agli elementi del vettore, la gestione dell'indice potrebbe essere realizzata in modo diverso:

```
            la $t1,vet
loop:
            …
            sw $s3,0($t1)
            la $t1,4($t1)  $t1<-[$t1]+4
            …
endloop:
```

Esempio

```
int vet[10];

for (i=0;i<N,i++)
  vet[i]=2*i
```

2a soluzione

```
# Inizializzazione
        ori   $s0,$0,9
        ori   $s3,$0,0
        la    $t1,vet
        # vet[i]=2*i
for:    sw    $s3,0($t1)
        la    $t1,4($t1)
        addi  $s3,$s3,2
        addi  $s0,$s0,-1
        bgez  $s0,for
```

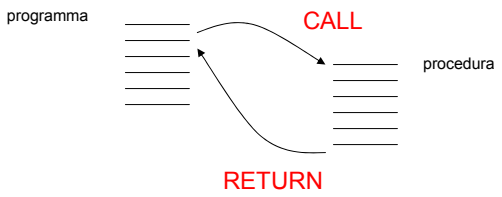
1a soluzione

```
# Inizializzazioni
        ori $s1,$0,10 # imax=10
        ori $s2,$0,1  # step=1
        ori $s0,$0,0  # i=0

for:    add $t1,$s0,$0
# calcola l'indirizzo di vet[i]
        mul $t1,$t1,4
        # vet[i]=2*i
        add $s3,$s0,$s0
        sw  $s3,vet($t1)
        # i=i+step
        add $s0,$s0,$s2
        slt $s3,$s0,$s1
        bne $s3,$0,for
```


Gestione delle procedure

temporaneo passaggio del controllo dal programma in esecuzione ad un sottoprogramma



Subroutine linkage

- CALL: viene salvato l'indirizzo di ritorno e quindi si effettua il salto
- RETURN: viene recuperato l'indirizzo di ritorno e si effettua il salto

Tecniche per il Subroutine Linkage

- Link Register
  - accesso ad un registro interno
  - gestione semplice e veloce
  - rende efficiente il caso di chiamata non innestata
- Stack
  - accesso a registri in memoria centrale
  - gestione più complessa (problema dello stack overflow)
  - risolve automaticamente il problema delle chiamate innestate

Tecniche per il passaggio dei parametri

- Registri interni
  - semplice e veloce
  - efficiente solo nel caso di pochi argomenti e procedure non innestate (caso più frequente ?)
- Stack
  - accesso a registri in memoria centrale
  - gestione più complessa (problema dello stack overflow)
  - robusto rispetto al problema del numero degli argomenti e delle chiamate innestate

Calling conventions

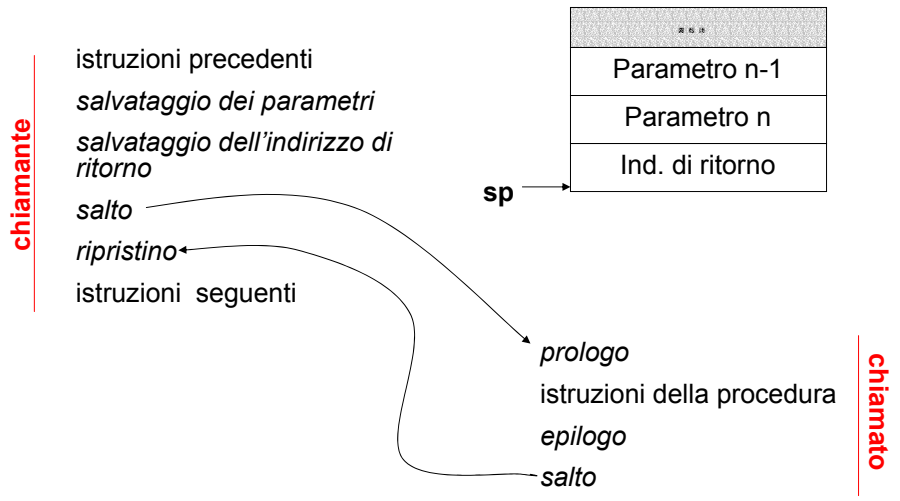
Riguardano le scelte operate dal chiamante relative a:

- Salvataggio dell'indirizzo di ritorno
- Salvataggio dei parametri
- Numero dei parametri
- Tipo dei parametri
- Ordine dei parametri

Leaving conventions

Riguardano le scelte relative a:

- Passaggio dei parametri in uscita
- Rispristino del contesto antecedente alla chiamata

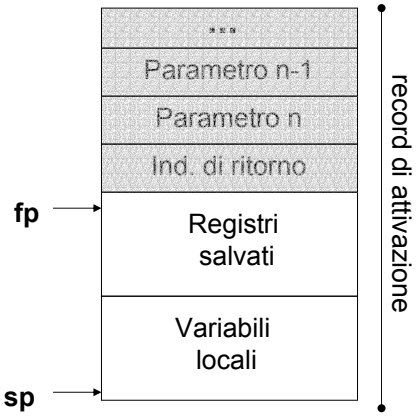


Prologo

- Creazione di uno stack frame
- Salvataggio di registri
- Allocazione di variabili locali

Epilogo

- Deallocazione delle variabili locali
- Ripristino dei registri
- Deallocazione dello stack frame




```
#  Esempio di chiamata a funzione
        .data
x:      .word 3
y:      .word 4
z:      .space 4

        .text
main:   lw $a0, x     # x è in $a0
        lw $a1, y     # y è in $a1
        jal sum
        add $a0,$v0,$0      #stampa il risultato
        li $v0, 1
        syscall
        li $v0, 10    # Codice di chiamata al sistema per la
        syscall               # fine del programma

sum:    add $t0,$a0,$0      # Preleva x
        add $t1,$a1,$0      # Preleva y
        add $t2,$t0,$t1     # Esegue la somma
        add $v0,$t2,$0      # Salva il risultato
        jr $ra        # Ritorna al chiamante
```

```
#include <iostream.h>
#include <stdlib.h>

void leggi_vet(int vet[],int &riemp,int max);
void stampa_vet(int vet[],int riemp);

int main()
{
     int vet[20];
     int riemp;
     int max=20;

     leggi_vet(vet,riemp,max);
     stampa_vet(vet,riemp);

     system("PAUSE");
     return 0;
}
```



```
void leggi_vet(int vet[],int &riemp,int max)
{
     int i;
     do{
        cout << "Riempimento (max = " << max << "): ";
        cin >> riemp;
     }while (riemp > max);

     for(i=0;i<riemp;i++)
     {
       cout << "Elemento(" << i << "): ";
       cin >> vet[i];
     }
}

void stampa_vet(int vet[],int riemp)
{
     int i;
     cout << "Il vettore contiene " << riemp << " elementi:\n";
     for(i=0;i<riemp;i++)
       cout << vet[i] << " ";
     cout << endl;
}
```

```
# Area Dati
       .data
chiediriemp:
       .asciiz "Riempimento (max = "
chiediel:
       .asciiz "Elemento ("
finechiedi:
       .asciiz "): "

       .align 2      # A che cosa serve ?
vet:   .space 80
riemp: .word  0
```



```
# Area Codice

       .text
main:

# Costruisce la lista degli argomenti per leggi_vet4
       la $a0,vet
       la $a1,riemp
       li $a2,20
       jal leggi_vet4

# Costruisce la lista degli argomenti per stampa_vet4
       la $a0,vet
       lw $a1,riemp
       jal stampa_vet4

# Termina l'esecuzione
       li $v0,10
       syscall
```

```
#####################################
######## Procedura leggi_vet4 ########
#####################################

leggi_vet4:

####### PROLOGO ############################
       subu $sp,$sp,12     # Alloca un frame di 3 word nello stack
       sw $a0,0($sp)  #
       sw $a1,4($sp)  # Salva gli argomenti sullo stack
       sw $a2,8($sp)  #
####### FINE PROLOGO ########################
```



```
# Richiede il riempimento
richiesta:
       li $v0,4
       la $a0,chiediriemp # Stampa la 1a parte del messaggio
       syscall

       li $v0,1
       lw $a0,8($sp)
       syscall

       li $v0,4
       la $a0,finechiedi
       syscall

       li $v0,5       # Legge il riempimento
       syscall

       lw $t0,8($sp)
       sgtu $t1,$v0,$t0   # Verifica che riemp<=max
       bnez $t1,richiesta

       lw $t0,4($sp)  # Restituisce il riempimento
       sw $v0,($t0)
```

```
# Inizializza il ciclo
        move $t1,$v0    # imax=riemp
        li $t2,1        # step=1
        li $t0,0        # i=0
        lw $t3,0($sp)   # indirizzo iniziale di vet
for1:
        bgeu $t0,$t1,endfor1
# Richiede l'i-mo elemento
        li $v0,4
        la $a0,chiediel # Stampa la 1a parte del messaggio
        syscall
        li $v0,1
        move $a0,$t0    # Stampa i
        syscall
        li $v0,4
        la $a0,finechiedi
        syscall

        li $v0,5        # Legge il valore
        syscall
        sw $v0,($t3)    # Lo memorizza
        la $t3,4($t3)   # Aggiorna il puntatore
        add $t0,$t0,$t2         # i=i+step
        b for1
```



```
endfor1:
####### EPILOGO ##############################
        lw $a0,0($sp)           #
        lw $a1,4($sp)           # Ripristina gli argomenti
        lw $a2,8($sp)           #
        addiu $sp,$sp,12        # Dealloca il frame sullo stack
####### FINE EPILOGO ########################

# RITORNO
        jr $ra
```

```
#####################################
######## Procedura stampa_vet4 ########
#####################################
       .data
messag1:
       .asciiz "Il vettore contiene "
messag2:
       .asciiz " elementi: \n"
space: .asciiz " "
endl:  .asciiz "\n"
       .text
stampa_vet4:

####### PROLOGO ############################
       subu $sp,$sp,8      # Alloca un frame di 2 word nello stack
       sw $a0,0($sp)       # Salva gli argomenti sullo stack
       sw $a1,4($sp)       #
####### FINE PROLOGO ########################
```



```
   # Stampa iniziale
          li $v0,4
          la $a0,messag1
          syscall

          li $v0,1
          lw $a0,4($sp)
          syscall

          li $v0,4
          la $a0,messag2
          syscall

   # Inizializza il ciclo
          lw $t1,4($sp) # imax=riemp
          li $t2,1      # step=1
          li $t0,0      # i=0

          lw $t3,0($sp) # indirizzo iniziale di vet
```

```
for2:
       bgeu $t0,$t1,endfor2
#
#      Blocco
#

# Stampa l'i-mo elemento
       li $v0,1
       lw $a0,($t3)
       syscall

       li $v0,4
       la $a0,space
       syscall

       la $t3,4($t3) # Aggiorna il puntatore

       add $t0,$t0,$t2      # i=i+step
#
#      Fine blocco
       b for2
```



```
endfor2:
       li $v0,4
       la $a0,endl
       syscall

####### EPILOGO ##############################
       lw $a0,0($sp)        #
       lw $a1,4($sp)        # Ripristina gli argomenti
       addiu $sp,$sp,8      # Dealloca il frame sullo stack
####### FINE EPILOGO ########################

# RITORNO
       jr $ra
```

```
#  Esempio di chiamata a procedura con l'uso di stack
        .data
x:      .word 3
a:      .word 10 3 6 5 9 1
m:      .byte 2
n:      .byte 3

        .text
main:   lw $a0, x           # x è in $a0
        la $a1, a           # l'indirizzo di a è in $a1
        lb $a2, m           # m è in $a2
        lb $a3, n           # n è in $a3
        subu $sp, $sp, 8    # Alloca un frame di 2 word nello stack
        li $t6, 1           # Indice i di riga dell'elemento
        sw $t6, 4($sp)      # Inserisce i
        li $t7, 2           # Indice j di colonna dell'elemento
        sw $t7, 0($sp)      # Inserisce j in cima allo stack
        jal prod

        addiu $sp, $sp, 8   # Ripristina lo stack pointer

        li $v0, 10          # Codice di chiamata al sistema per la
        syscall             # fine del programma
```



```
prod:   lw $t0, 4($sp)      # Preleva i
        subu $t0, $t0, 1    # Lo riduce di 1 per usarlo come offset
        lw $t1, 0($sp)      # Preleva j
        subu $t1, $t1, 1    # Lo riduce di 1 per usarlo come offset
        mul $t2, $t0, $a3   # elemento=i*n+j (partendo da 0)
        addu $t2, $t2, $t1
        mul $t2, $t2, 4     # byte=elemento*4
        addu $t3, $a1, $t2  # Indirizzo dell'elemento selezionato
        lw $t4, 0($t3)      # L'elemento è in $t4
        mul $v0, $t4, $a0   # Esegue il prodotto
        jr $ra              # Ritorna al chiamante
```

```
# Esempio di chiamata a procedura con uso dei registri
# Legge una stringa dalla console e ne conta la lunghezza.

        .data
request:
        .asciiz "Inserisci una stringa: "
answer1:
        .asciiz "la stringa è lunga "
answer2:
        .asciiz " caratteri.\n"

        .text
main:
        la $a0, request     # stampa la richiesta
        jal prtstr

        lui $a0, 0x1000     # prepara il buffer per la stringa
        li $a1, 32          # fissa la lunghezza del buffer
        jal readstr         # legge la stringa

        jal strlen          # salta alla procedura
```



```
        la $a0, answer1     # stampa il messaggio in uscita
        jal prtstr

        move $a0,$v1        # trasferisce il risultato
        jal prtint          # e lo stampa

        la $a0, answer2     # termina l'output
        jal prtstr

        li $v0, 10          # termina il programma
        syscall

readstr:                    # funzione per la lettura della
        li $v0, 8           # stringa
        syscall
        jr $ra

prtstr:                     # funzione per la stampa della
        li $v0, 4           # stringa
        syscall
        jr $ra

prtint:
        li $v0, 1           # funzione per la stampa dell'intero
        syscall
        jr $ra
```

```
# funzione per il calcolo della lunghezza della stringa
#
strlen:
       li $v1, 0            # azzera v1 (contatore)
       li $t0, 10           # carica 10 (CR) in t0
loop:
       lb $t1, 0($a0)       # a0 fa da puntatore all'interno
                            # della stringa

       beq $t1, $t0, ret    # verifica se car. = 10 o car. = 0
       beqz $t1, ret

       addi $v1, $v1, 1     # incrementa il contatore
       addi $a0, $a0, 1     # aggiorna il puntatore
       b loop               # ritorna in loop
ret:
       jr $ra               # ritorna
                            # restituisce in v1 il risultato
```