

Architetture dei Calcolatori

Procedure

Prof. Francesco Lo Presti

Le procedure

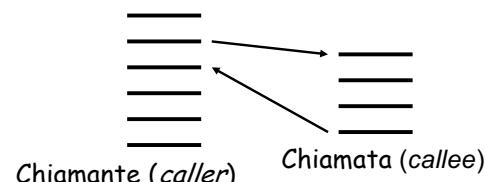
- In ogni linguaggio di programmazione si struttura il proprio codice usando **procedure** (funzioni, metodi, ...)
- L'utilizzo di procedure permette
 - Astrazione
 - Riusabilità del codice
 - Maggiore leggibilità e strutturazione del codice
 - Testing più approfondito
 - Creazione di librerie di funzioni utilizzate di frequente

Procedure

1

Le procedure (2)

- Quando c'è una chiamata di procedura, il programma:
 - Salta all'inizio della procedura
 - Esegue il codice della procedura
 - Torna nel punto di origine in cui era stata chiamata la procedura



- Le procedure sono caratterizzate da parametri di ingresso e la loro esecuzione può modificare lo stato del programma e/o restituire uno o più valori

Le procedure (3)

- Nei linguaggi ad alto livello, i dettagli per la corretta gestione delle chiamate a procedura sono nascoste dal compilatore
- In Assembler il programmatore deve esplicitamente implementare tutti i passi necessari per una corretta invocazione, esecuzione e chiusura delle procedure

Procedure

3

Passi per l'esecuzione di una Procedura

1. La routine principale (caller) pone i parametri da passare in una posizione accessibile alla procedura chiamata (callee)
 - \$a0-\$a3: 4 registri argomento
2. Il chiamante trasferisce il controllo alla procedura chiamata
 - istruzione jal (jump and link)
3. La procedura chiamata acquisisce le risorse di memoria necessarie
 - spazio variabili locali
4. La procedura chiamata esegue il compito richiesto
5. La procedura chiamata pone i risultati in una posizione accessibile al chiamante
 - \$v0-\$v1: 2 registri valore di ritorno
6. La procedura chiamata restituisce il controllo alla procedura chiamante
 - jr \$ra; registro indirizzo di ritorno della chiamata (return address register)

Procedure 4

Riassumendo

- La procedura chiamante deve
 1. Mettere i valori dei parametri da passare alla procedura chiamata nei registri \$a0-\$a3
 2. Utilizzare l'istruzione jal per saltare alla procedura chiamata e memorizzare il valore di PC+4 nel registro \$ra
- La procedura chiamata deve
 1. Eseguire il compito richiesto
 2. Memorizzare il risultato nei registri \$v0, \$v1
 3. Restituire il controllo alla procedura chiamante con l'istruzione jr \$ra

Procedure 6

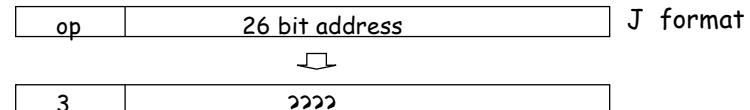
Invocazione di Procedure: Istruzione jal

▫ Istruzione chiamata di procedura MIPS:

jal ProcedureAddress #jump and link

▫ Salva l'indirizzo dell'istruzione che segue la chiamata di procedura (PC+4) nel registro \$ra

▫ Formato Istruzione:



▫ Ritorno al punto di chiamata

jr \$ra #return

Procedure 5

Un Primo Esempio

▫ Codice C:

```
int example(int g, int h, int i, int j)
{
    int f;
    f = (g+h) - (i+j);
    return f;
}
```

▫ Assumiamo che le variabili g, h, i e j corrispondano a \$a0-\$a3 e f a \$s0

Procedure 7

...e la sua codifica

```
.globl main

example:
    add $t0, $a0, $a1 # $t0 = g + h
    add $t1, $a2, $a3 # $t1 = i + j
    sub $s0, $t0, $t1 # f = $t0 - $t1

    add $v0, $s0, $0 # restituisci f come risultato
    jr $ra             # ritorna alla procedura chiamante

main:
    addi $s0, $zero, 1
    addi $s1, $zero, 2
    ...
    addi $a0, $zero, 5
    addi $a1, $zero, 6
    addi $a2, $zero, 4
    addi $a3, $zero, 3
    jal example
    ...
```

Problemi?

Record di Attivazione

- Il record di attivazione (procedure frame) è un'area di memoria per
 - Passare argomenti alla procedura (se >4)
 - Salvataggio dei registri che la procedura potrebbe modificare ma che servono al chiamante
 - Variabili locali alla procedura
- Il programmatore assembler deve provvedere esplicitamente ad allocare/deallocare lo spazio dei record di attivazione
- L'invocazione/ritorno delle procedure e' di tipo LIFO
⇒ Allocazione/Deallocazione ben si esegue su uno Stack
 - Procedure frame=Stack frame
 - 1. Allocazione nello Stack al momento dell'invocazione della procedura
 - 2. Deallocazione dallo Stack al momento del ritorno

Problemi

- Come gestire i registri che la procedura chiamata potrebbe modificare, ma che la procedura chiamante ha bisogno di mantenere inalterati
- Una procedura può avere bisogno di più registri rispetto ai 4 a disposizione per il passaggio dei parametri ed i 2 per la restituzione dei valori
- Fornire/Gestire lo spazio necessario per le variabili locali alla procedura chiamata
- Gestione di **procedure annidate** (procedure che chiamano al loro interno altre procedure) e **procedure ricorsive** (procedure che invocano se stesse)
 - Ho un solo registro \$ra



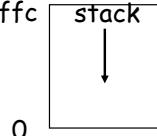
Procedure Frame + Stack +Convenzioni di Chiamata a Procedura
Procedure

Lo Stack

- Lo stack (pila) è una struttura dati costituita da una coda LIFO (last-in-first-out)
- I dati sono inseriti nello stack con l'operazione **push**
- I dati sono prelevati dallo stack con l'operazione **pop**
- E' necessario un puntatore alla cima (top) dello stack per salvare i registri che servono alla procedura chiamata
- Il registro \$sp (**stack pointer**) contiene l'indirizzo del top dello stack e viene aggiornato ogni volta che vengono inserite od estratte delle informazioni

Gestione dello Stack nel MIPS

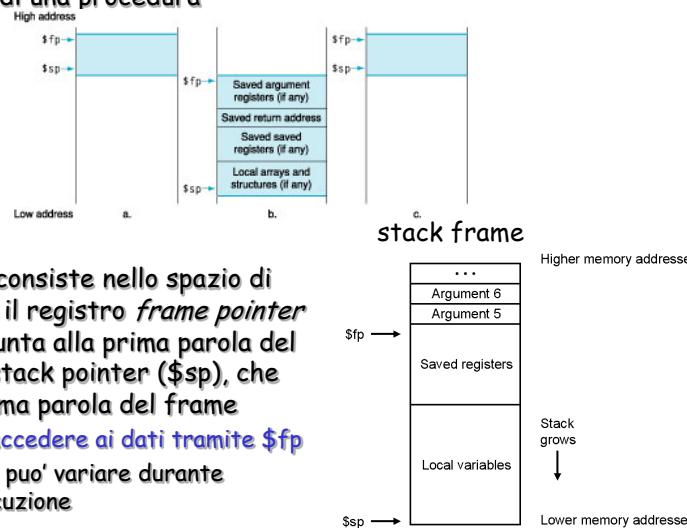
- Lo stack cresce da indirizzi di memoria alti verso indirizzi di memoria bassi
- L'inserimento di un dato nello stack (operazione di push) avviene **decrementando** \$sp per allocare lo spazio ed eseguendo sw per inserire il dato
 - Ex: salvare il registro \$s0 nello stack
`addi $sp, $sp, -4
sw $s0, 0($sp)`
- Il prelevamento di un dato nello stack (operazione di pop) avviene eseguendo lw ed **incrementando** \$sp (per eliminare il dato), riducendo così la dimensione dello stack
 - Es: prelevare la cima dello stack e salvarla in \$s0
`lw $s0, 0($sp)
addi $sp, $sp, 4`



Procedure 12

Record di Attivazione e Stack (2)

- Allocazione dello stack prima (a), durante (b) e dopo (c) la chiamata di una procedura



- Il frame consiste nello spazio di memoria tra il registro **frame pointer** (\$fp), che punta alla prima parola del frame, e lo **stack pointer** (\$sp), che punta all'ultima parola del frame
 - Si può accedere ai dati tramite \$fp
 ✓ \$sp puo' variare durante l'esecuzione

14

Record di Attivazione e Stack

- Tutto lo spazio di cui ha bisogno una procedura (**record di attivazione**) viene esplicitamente allocato dal programmatore
- Lo spazio dello stack viene allocato sottraendo a \$sp il numero di byte necessari
 - addi \$sp, \$sp, -24 # alloca 6 parole nello stack
- Al rientro da una procedura, il record di attivazione viene rimosso (deallocato) dalla procedura stessa, incrementando \$sp della stessa quantità di cui lo si era decrementato alla chiamata
 - addi \$sp, \$sp, 24 # deallo. 6 parole nello stack

Procedure 13

Convenzioni per il salvataggio dei registri/ambiente

- L'esecuzione di una procedura non deve interferire con l'ambiente chiamante
- I registri usati dalla procedura chiamante devono poter essere ripristinati al rientro della procedura chiamata
- Convenzioni adottate dal MIPS
 - Per ottimizzare gli accessi alla memoria, la chiamante e la chiamata salvano sullo stack soltanto un particolare gruppo di registri
 - La chiamante, se vuole che siano preservati, salva i registri \$t0-\$t9, i registri \$a0-\$a3 e passa eventuali argomenti aggiuntivi
 - La chiamata salva \$ra e registri \$s0-\$s7 (se li usa), e usa lo stack per le strutture dati locali (ad es. array, strutture) e le variabili locali

Preservato	Non preservato
Registri saved: \$s0-\$s7	Registri temporanei: \$t0-\$t9
Registro stack pointer: \$sp	Registri argomento: \$a0-\$a3
Registro return address: \$ra	Registri di ritorno: \$v0-\$v1
Stack sopra lo stack pointer	Stack sotto lo stack pointer

15

Convenzioni di Chiamata a Procedura

□ La procedura chiamante deve

1. Mettere i valori dei parametri da passare alla procedura chiamata nei registri \$a0-\$a3. Ulteriori argomenti sono inseriti nello stack (dopo 2).
2. Salva i registri Caller-saved. La chiamante, se vuole che siano preservati, inserisce i registri \$t0-\$t9, i registri \$a0-\$a3, \$v0-\$v1 (Caller-saved) nello stack
 1. perche' la procedura chiamata li usa senza salvarli
3. Utilizzare l'istruzione jal per saltare alla procedura chiamata e memorizzare il valore di PC+4 nel registro \$ra
4. Al ritorno ripristinare i registri salvati nella fase 2

Procedure 16

Convenzioni di Chiamata a Procedura

□ La procedura chiamata deve

1. Allocare la memoria per lo stack-frame sottraendo da \$sp il numero di byte necessari
2. Salvare i registri Callee-saved - \$s0-\$s7, \$fp e \$ra - che verranno modificati dalla procedura stessa
 - ✓ Perche' la procedura chiamante si aspetta che siano preservati
 - ✓ \$fp solo se viene usato
 - ✓ \$ra solo se la procedura ne chiama altre
3. Impostare \$fp = \$sp+dimensione stack-frame-4
 - ✓ Solo se \$fp viene usato
4. Eseguire il compito richiesto
5. Memorizzare il risultato nei registri \$v0, \$v1
6. Ripristinare i registri salvati nella fase 2
 - ✓ quelli effettivamente modificati
7. Deallocare lo stack-frame comando a \$sp la dimensione del frame
8. Restituire il controllo alla procedura chiamante con l'istruzione jr \$ra

Ritorno

Procedure 17

Esempio di procedura foglia

□ E' una procedura che non ha annidate al suo interno chiamate ad altre procedure

- Non serve salvare nello stack \$ra (perche' nessun altro lo modifica)

□ Codice C:

```
int leaf_example(int g, int h, int i, int j)
{
    int f;
    f = (g+h) - (i+j);
    return f;
}
```

□ Assumiamo che le variabili g, h, i e j corrispondono a \$a0-\$a3 e f a \$s0

Procedure 18

Esempio di procedura foglia (2)

```
.globl main
leaf_example:
    addi $sp, $sp, -4          # decrementa lo stack di 4 per salvare $s0
    sw   $s0, 0($sp)           # push di $s0 nello stack

    add  $t0, $a0, $a1          # $t0 = g + h
    add  $t1, $a2, $a3          # $t1 = i + j
    sub  $s0, $t0, $t1          # f = $t0 - $t1

    add  $v0, $s0, $zero        # restituisci f come risultato
    lw   $s0, 0($sp)           # pop di $s0 dallo stack
    addi $sp, $sp, 4            # incrementa lo stack
    jr  $ra                     # ritorna alla procedura chiamante

main:
    addi $s0, $zero, 1
    addi $s1, $zero, 2
    ...
    addi $a0, $zero, 5
    addi $a1, $zero, 6
    addi $a2, $zero, 4
    addi $a3, $zero, 3
    jal leaf_example
    ...
```

Procedure 19

Esempio di procedura ricorsiva

- ❑ E' una procedura che contiene una chiamata a se stessa al suo interno
- ❑ Occorre prestare attenzione al salvataggio nello stack dei registri che occorrono alla procedura chiamante (parametri di input, indirizzo di ritorno)
- ❑ Codice C:

```
int fact(int n)
{
    if (n < 1) return(1);
    else return (n * fact(n-1));
}
```

- ❑ Assumiamo che la variabile n corrisponde a \$a0

Procedure 20

```
int fact(int n)
{
    int i=n, ris;
    if (i < 1) ris=1;
    else ris=i * fact(i-1);
    return ris;
}
```

Procedure 21

Fattoriale (seconda versione)

```
fact:
    addi $sp, $sp, -12      # per salvare 3 registri nello stack
    sw $ra, 8($sp)          # push di $ra nello stack
    sw $s0, 4($sp)          # push di $s0 nello stack
    sw $s1, 0($sp)          # push di $s1 nello stack
    add $s0, $0, $a0          # copia $a0 in $s0
    add $s1, $0, $0          # ris=0;
    slti $t0, $s0, 1          # test se n < 1
    beq $t0, $zero, Else     # se n>= 1, vai a Else
    addi $s1, $0, 1          # se n<1 ris=1
    addi $v0, $0, $s1          # $v0=ris=1
    j Exit
Else: addi $a0, $s0, -1      # if n >= 1 $a0=n-1
    jal fact                # chiama fact(n-1)
    mul $v0, $s0, $v0          # $v0=n*fact(n-1)
Exit: lw $s1, 0($sp)          # pop di $s1 dallo stack
    lw $s0, 4($sp)          # pop di $s0 dallo stack
    lw $ra, 8($sp)          # pop di $ra dallo stack
    addi $sp, $sp, 12          # dealloca stack
    jr $ra                  # ritorno alla procedura chiamante
```

Procedure 22

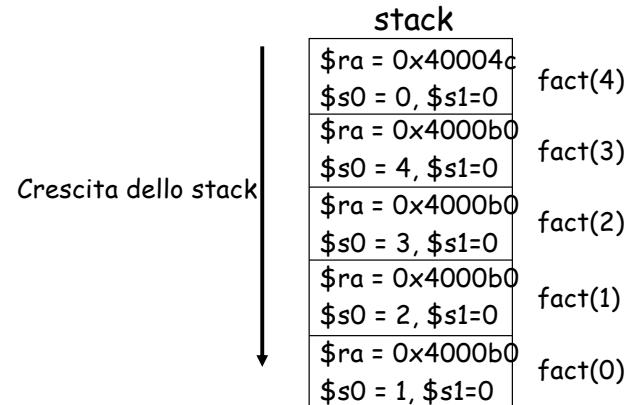
Esempio di procedura ricorsiva

- ❑ Per semplicità consideriamo prima la seguente versione
- ❑ Codice C:

```
int fact(int n)
{
    int i=n, ris;
    if (i < 1) ris=1;
    else ris=i * fact(i-1);
    return ris;
}
```

Esempio di procedura ricorsiva (3)

- ❑ Stack alla chiamata fact(0) nel caso in cui main chiama fact(4)



Procedure 23

Esempio di procedura ricorsiva (versione del libro)

```
fact:  
    addi $sp, $sp, -8          # per salvare 2 registri nello stack  
    sw $ra, 4($sp)            # push di $ra nello stack  
    sw $a0, 0($sp)            # push dell'argomento n nello stack  
    slti $t0, $a0, 1           # test se n < 1  
    beq $t0, $zero, L1         # se n>= 1, vai a L1  
    addi $v0, $zero, 1          # se n < 1, return 1  
    addi $sp, $sp, 8             # incrementa lo stack  
    jr $ra                     # ritorna alla procedura chiamante  
L1:   addi $a0, $a0, -1        # n >= 1: decrementa n di 1  
    jal fact                  # chiama fact(n-1)  
    lw $a0, 0($sp)             # pop di n dallo stack  
    lw $ra, 4($sp)             # pop di $ra dallo stack  
    addi $sp, $sp, 8             # dealloca stack  
    mul $v0, $a0, $v0           # return n*fact(n-1)  
    jr $ra                     # ritorno alla procedura chiamante
```

Procedure 24

Esercizio

- Scrivere una procedura in assembler MIPS che, dato in input un intero n , calcola il numero di Fibonacci ad esso corrispondente $F(n)$

$F(n) = F(n-1) + F(n-2)$, essendo $F(0) = 0$ e $F(1) = 1$

- Codice C

```
int fib(n)  
{  
    if (n == 0) return(0);  
    else  
        if (n==1) return(1);  
        else  
            return(fib(n-1)+fib(n-2));  
}
```

Procedure

25

Esempio: swap

- Codice C
- Assunzioni:
 - v in $\$a0$
 - k in $\$a1$
- Codice MIPS

```
void swap(int v[], int k)  
{  
    int temp;  
    temp = v[k];  
    v[k] = v[k+1];  
    v[k+1] = temp; }
```

swap:

```
sll $t1, $a1, 2          # $t1 = k*4  
add $t1, $a0, $t1          # $t1 = v + (k*4)  
lw $t0, 0($t1)            # $t0 = v[k]  
lw $t2, 4($t1)            # $t2 = v[k+1]  
sw $t2, 0($t1)            # v[k] = $t2  
sw $t0, 4($t1)            # v[k+1] = $t1  
jr $ra
```

Procedure 26

Esempio: sort

- Ordinamento crescente di un vettore di interi
- Codice C

```
sort (int v[], int n)  
{  
    int i, j;  
    for (i=0; i<n; i++)  
        for (j=i-1; j>=0 && v[j]>v[j+1]; j--)  
            swap(v, j);  
}
```

- Assunzioni:

- v in $\$a0$ e n in $\$a1$
- i associato a $\$s0$ e j a $\$s1$

Procedure

27

Esempio: sort (2)

```
# salvataggio di 5 registri nello stack
sort: addi $sp, $sp, -20
    sw $ra, 16($sp)
    sw $s3, 12($sp)
    sw $s2, 8($sp)
    sw $s1, 4($sp)
    sw $s0, 0($sp)
# corpo della procedura
    add $s2, $zero, $a0      # $s2 = $a0
    add $s3, $zero, $a1      # $s3 = $a1
# loop esterno
    add $s0, $zero, $zero    # i = 0
for1tst: slt $t0, $s0, $s3 # $t0=0 se $s0>=$s3 (i >= n)
    beq $t0, $zero, exit1  # go to exit1 se $s0>=$s3 (i >= n)
# loop interno
    addi $s1, $s0, -1       # j = i-1
```

Procedure 28

Esempio: sort (4)

```
# fine loop esterno
exit2: addi $s0, $s0, 1          # i=i+1
    j for1tst

# ripristina i registri salvati nello stack
exit1: lw $s0, 0($sp)
    lw $s1, 4($sp)
    lw $s2, 8($sp)
    lw $s3, 12($sp)
    lw $ra, 16($sp)
    addi $sp, $sp, 20

# ritorno alla procedura chiamante
    jr $ra
```

Procedure 30

Esempio: sort (3)

```
for2tst: slti $t0, $s1, 0      # $t0=1 se $s1<0 (j<0)
    bne $t0, $zero, exit2  # go to exit2 se $s1<0 (j<0)
    sll $t1, $s1, 2          # $t1 = j*4
    add $t2, $s2, $t1        # $t2 = v + (j*4)
    lw $t3, 0($t2)           # t3 = v[j]
    lw $t4, 4($t2)           # t4 = v[j+1]
    slt $t0, $t4, $t3        # $t0=0 if $t4>=$t3 (v[j+1]>=v[j])
    beq $t0, $zero, exit2

# passaggio dei parametri e chiamata di swap
    add $a0, $zero, $s2
    add $a1, $zero, $s1
    jal swap
# fine loop interno
    addi $s1, $s1, -1        # j=j-1
    j for2tst
```

Procedure 29

Esempio: sort - Inizializzazione

```
# salvataggio di 5 registri nello stack
sort: addi $sp, $sp, -20
    sw $ra, 16($sp)
    sw $s3, 12($sp)
    sw $s2, 8($sp)
    sw $s1, 4($sp)
    sw $s0, 0($sp)

    add $s2, $zero, $a0      # $s2 = $a0
    add $s3, $zero, $a1      # $s3 = $a1
```

Procedure 31

Esempio: sort - Loop Esterno

```
add $s0, $zero, $zero      # i = 0
for1tst: slt $t0, $s0, $s3    # $t0=1 se $s0>=$s3 (i >= n)
    beq $t0, $zero, exit1   # go to exit1 se $s0>=$s3 (i >= n)
    ...
    # corpo loop esterno
    ...
    addi $s0, $s0, 1        # i=i+1
    j for1tst
exit1:
```

Procedure 32

Esempio: sort - Loop interno

```
addi $s1, $s0, -1          # j = i-1
for2tst: slti $t0, $s1, 0    # $t0=1 se $s1<0 (j<0)
    bne $t0, $zero, exit2  # go to exit2 se $s1<0 (j<0)
    sll $t1, $s1, 2        # $t1 = j*4
    add $t2, $s2, $t1       # $t2 = v + (j*4)
    lw $t3, 0($t2)          # t3 = v[j]
    lw $t4, 4($t2)          # t4 = v[j+1]
    slt $t0, $t4, $t3       # $t0=0 if $t4>=$t3 (v[j+1]>=v[j])
    beq $t0, $zero, exit2
    ...
    # istr. Loop interno
```

```
addi $s1, $s1, -1          # j=j-1
j for2tst
exit2:
```

Procedure 33

Esempio: sort - Ripristino

```
# ripristina i registri salvati nello stack
exit1: lw $s0, 0($sp)
    lw $s1, 4($sp)
    lw $s2, 8($sp)
    lw $s3, 12($sp)
    lw $ra, 16($sp)
    addi $sp, $sp, 20

# ritorno alla procedura chiamante
jr $ra
```

Procedure 34

Rappresentazione di stringhe

- Tre opzioni per la rappresentazione di stringhe
 - La prima posizione della stringa contiene la sua lunghezza
 - La lunghezza è memorizzata in una variabile separata
 - L'ultima posizione della stringa è segnalata da un carattere speciale (NULL), la cui codifica ASCII è 0
 - ✓ Rappresentazione usata dal linguaggio C

Procedure 35

Esempio: strcpy

- Copiare la stringa y nella stringa x

- Codice C

```
void strcpy (char x[], char y[])
{
    int i;

    i = 0;
    while ((x[i] = y[i]) != 0)
        i++;
}
```

- Assumiamo che \$a0 e \$a1 contengono gli indirizzi base di x e y
- Assumiamo che \$s0 contiene i

Procedure 36

Esempio: strcpy (2)

```
strcpy:
    addi $sp, $sp, -4      # decrementa lo stack per salvare $s0
    sw $s0, 0($sp)         # push di $s0 nello stack
    add $s0, $zero, $zero   # i = 0
L1:   add $t1, $a1, $s0     # indirizzo di y[i] in $t1
    lb $t2, 0($t1)         # $t2 = y[i]
    add $t3, $a0, $s0     # indirizzo di x[i] in $t3
    sb $t2, 0($t3)         # x[i] = y[i]
    addi $s0, $s0, 1       # i = i+1
    bne $t2, $zero, L1    # se y[i] != 0 vai a L1
    lw $s0, 0($sp)         # pop di $s0 dallo stack
    addi $sp, $sp, 4       # incrementa lo stack
    jr $ra                 # ritorna alla procedura chiamante
```

Procedure 37

Esempio: copia parziale di un vettore

- Scrivere una procedura in assembler MIPS che effettua la copia dei soli elementi positivi del vettore di interi elem nel vettore pos_elem; la procedura restituisce il numero di elementi del vettore pos_elem

- Assumiamo che

- \$a0 contenga l'indirizzo base dell'array elem
- \$a1 contenga l'indirizzo base dell'array pos_elem
- \$a2 contenga n, il numero elementi dell'array elem

- Procedura foglia

- Diverse versioni

- Base
- Usando solo variabili temporanee
- Usando i puntatori

Procedure 38

Esempio: copia parziale di un vettore (2)

```
cp_pos: addi $sp, $sp, -20
        sw $s0, 0($sp)
        ...
        sw $s4, 16($sp)
        add $s0, $0, $a0 # $s0=indirizzo base elem
        add $s1, $0, $a1 # $s1=indirizzo base pos_elem
        add $s2, $0, $a2 # $s2=n numero di elementi in elem
        add $s3, $0, $0 # $s3=count (numero di elementi in pos_elem)
        add $s4, $0, $0 # $s4=i (indice)
Loop:   beq $s4, $s2, Exit_if # if(i==n) goto Exit
        sll $t0, $s4, 2 # $t0=i*4
        add $t0, $s0, $t0 # $t0=&elem[i]
        lw $t1, 0($t0) # $t1=elem[i]
        slti $t2, $t1, 1 # $t2=1 iff $t1<1 → iff $t1<=0
        bne $t2, $zero, Exit_if # se $t1 <= 0, vai a Exit_if
        sll $t2, $s3, 2 # $t2=count*4
        add $t2, $s1, $t2 # $t2=&pos_elem[count]
        sw $t1, 0($t2) # pos_elem[count] = $t1 = elem[i]
        addi $s3, $s3, 1 # count++
Exit_if: addi $s4, $s4, 1 # i++
         j Loop
Exit:   add $v0, $0, $s3 # return count
        lw $s4, 16($sp)
        ...
        lw $s0, 0($sp)
        addi $sp, $sp, 20# dealloca lo spazio nello stack
        jr $ra
```

Procedure 39

Esempio: copia parziale di un vettore - Varianti

```

cp_pos: addi $sp, $sp, -8      □ Non usare $s0, $s1 e $s2
      sw $s3, 0($sp)
      sw $s4, 4($sp)
      add $s3, $0, $0 # $s3=count (numero di elementi in pos_elem)
      add $s4, $0, $0 # $s4=i (indice)
Loop: beq $s4, $a2, Exit      # if(i==n) goto Exit
      sll $t0, $s4, 2      # $t0=i*4
      add $t0, $a0, $t0      # $t0=&elem[i]
      lw $t1, 0($t0)        # $t1=elem[i]
      slti $t2, $t1, 1      # $t2=1 iff $t1<1 ➔ iff $t1<=0
      bne $t2, $zero Exit_if # se $t1 <= 0, vai a Exit_if
      sll $t2, $s3, 2      # $t2=count*4
      add $t2, $a1, $t2      # $t2=&pos_elem[count]
      sw $t1, 0($t2)        # pos_elem[count] = $t1 = elem[i]
      addi $s3, $s3, 1      # count++
Exit_if: addi $s4, $s4, 1      # i++
      j Loop
Exit: add $v0, $0, $s3        # return count
      lw $s4, 4($sp)
      lw $s3, 0($sp)
      addi $sp, $sp, 8        # dealloca lo spazio nello stack
      jr $ra

```

Procedure 40

Esempio: copia parziale di un vettore - Varianti (3)

```

cp_pos: add $t3, $0, $0 # $t3=count (numero di elementi in pos_elem)
      add $t5, $0, $a0      # $t5=&elem[0]
      sll $t6, $a2, 2      # $t6=4*n
      add $t6, $a0, $t6      # $t6=&elem[n]
      add $t7, $0, $a1      # $t7=&pos_elem[0]
Loop: beq $t5, $t6, Exit      # if($t5==&elem[n]) goto Exit
      lw $t1, 0($t5)        # $t1=$t5
      slti $t2, $t1, 1      # $t2=1 iff $t1<1 ➔ iff $t1<=0
      bne $t2, $zero Exit_if # se $t1 <= 0, vai a Exit_if
      sw $t1, 0($t7)        # *t7=*t5
      addi $t7, $t7, 4      # $t7 punta al prox elemento di pos_elem
      addi $t3, $t3, 1      # count++
Exit_if: addi $t5, $t5, 4      # $t5 punta al prox elemento di elem
      j Loop
Exit: add $v0, $0, $t3        # return count
      jr $ra

```

Procedure 42

Esempio: copia parziale di un vettore - Varianti (2)

□ Usare \$t* al posto di \$s*

```

cp_pos: add $t3, $0, $0 # $t3=count (numero di elementi in pos_elem)
      add $t4, $0, $0      # $t4=i (indice)
Loop: beq $t4, $a2, Exit      # if(i==n) goto Exit
      sll $t0, $s4, 2      # $t0=i*4
      add $t0, $a0, $t0      # $t0=&elem[i]
      lw $t1, 0($t0)        # $t1=elem[i]
      slti $t2, $t1, 1      # $t2=1 iff $t1<1 ➔ iff $t1<=0
      bne $t2, $zero Exit_if # se $t1 <= 0, vai a Exit_if
      sll $t2, $t3, 2      # $t2=count*4
      add $t2, $a1, $t2      # $t2=&pos_elem[count]
      sw $t1, 0($t2)        # pos_elem[count] = $t1 = elem[i]
      addi $t3, $t3, 1      # count++
Exit_if: addi $t4, $t4, 1      # i++
      j Loop
Exit: add $v0, $0, $t3        # return count
      jr $ra

```

Procedure 41

Esempio: copia parziale di un vettore - Varianti (4)

□ Versione Finale

```

cp_pos: add $v0, $0, $0 # $v0=count (numero di elementi in pos_elem)
      sll $t6, $a2, 2      # $t6=4*n
      add $t6, $a0, $t6      # $t6=&elem[n]
Loop: beq $a0, $t6, Exit      # if($a0==&elem[n]) goto Exit
      lw $t1, 0($a0)        # $t1=$t0
      slti $t2, $t1, 1      # $t2=1 iff $t1<1 ➔ iff $t1<=0
      bne $t2, $zero Exit_if # se $t1 <= 0, vai a Exit_if
      sw $t1, 0($a1)        # *t7=*t5
      addi $a1, $a1, 4      # $a1 punta al prox elemento di pos_elem
      addi $v0, $v0, 1      # count++
Exit_if: addi $a0, $a0, 4      # $a0 punta al prox elemento di elem
      j Loop
Exit: jr $ra                  # return count

```

Procedure 43