

# Programmazione Assembly: procedure (2)

**Convenzioni di uso dei registri**  
**Utilizzo della memoria**  
**Procedure innestate**  
**Uso dello stack**



# Convenzioni assembly: nomi e usi dei registri

Nome Simbolico	Numero	Uso
\$zero	0	Costante 0
\$at	1	Assembler temporary
\$v0-\$v1	2-3	Functions and expressions evaluation
\$a0-\$a3	4-7	Arguments
\$t0-\$t7	8-15	Temporaries
\$s0-\$s7	16-23	Saved Temporaries
\$t8-\$t9	24-25	Temporaries
\$k0-\$k1	26-27	Reserved for OS kernel
\$gp	28	Global pointer
\$sp	29	Stack pointer
\$fp	30	Frame pointer
\$ra	31	Return address

- Appendice A Hennessy-Patterson Sez. A.6
- Usati da assembler, compilatore, sistema operativo
- Secondo specifiche convenzioni
- ...da trattare con cautela se si programma in assembly!!!



# Registri temporanei “salvati” e “non salvati”

- **Se una procedura usa registri, cosa succede del contenuto lasciato nei registri dal chiamante?**
  - Convenzioni: registri \$s e \$t
- **CONVENZIONI su uso dei registri \$t e \$s**
- **I registri \$t (“temporary”) non sono salvati dalla procedura**
  - Il chiamante non si può aspettare di trovare immutati i contenuti dei registri \$t dopo una chiamata a procedura
  - I contenuti dei registri \$t devono essere salvati dal chiamante prima della chiamata a procedura
- **I registri \$s (“saved”) sono salvati dalla procedura**
  - Il chiamante ha il diritto di aspettarsi che i contenuti dei registri \$s siano immutati dopo una chiamata a procedura
  - Se la procedura usa i registri \$s deve salvarne il contenuto all’inizio e ripristinarlo prima del ritorno
- **Dove salvare il contenuto dei registri \$s?**
  - Uso dello stack



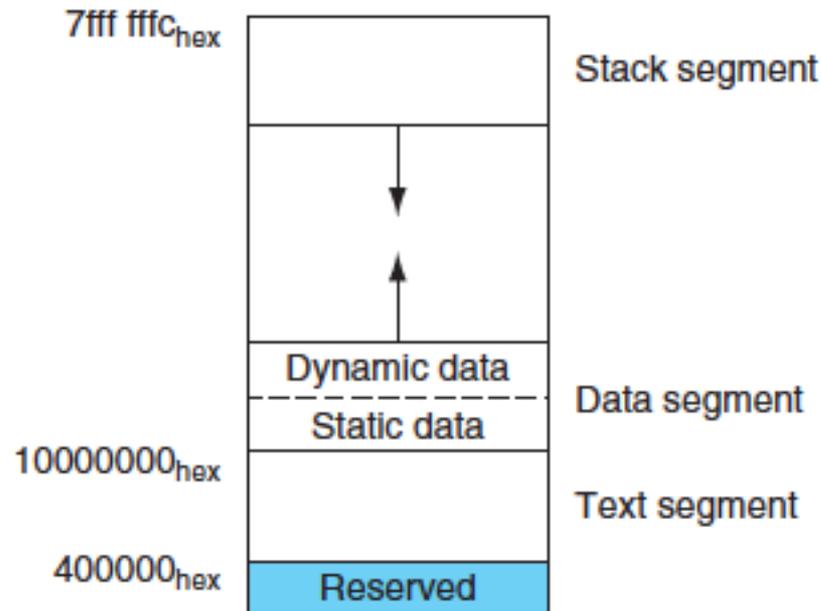
# Procedure innestate

- **Procedure “foglia” e “non foglia”**
  - Una procedura foglia NON chiama altre procedure
  - Una procedura non foglia chiama altre procedure
- **Cosa succede se una procedura ne chiama un'altra?**
  - Si perde il contenuto di \$ra della prima chiamata??
  - Procedure recursive??
  - Bisogna che una procedura “non foglia” salvi il contenuto di \$ra e lo ripristini prima del ritorno
- **Dove salvare il contenuto dei registri \$ra e \$s?**
  - Uso dello stack



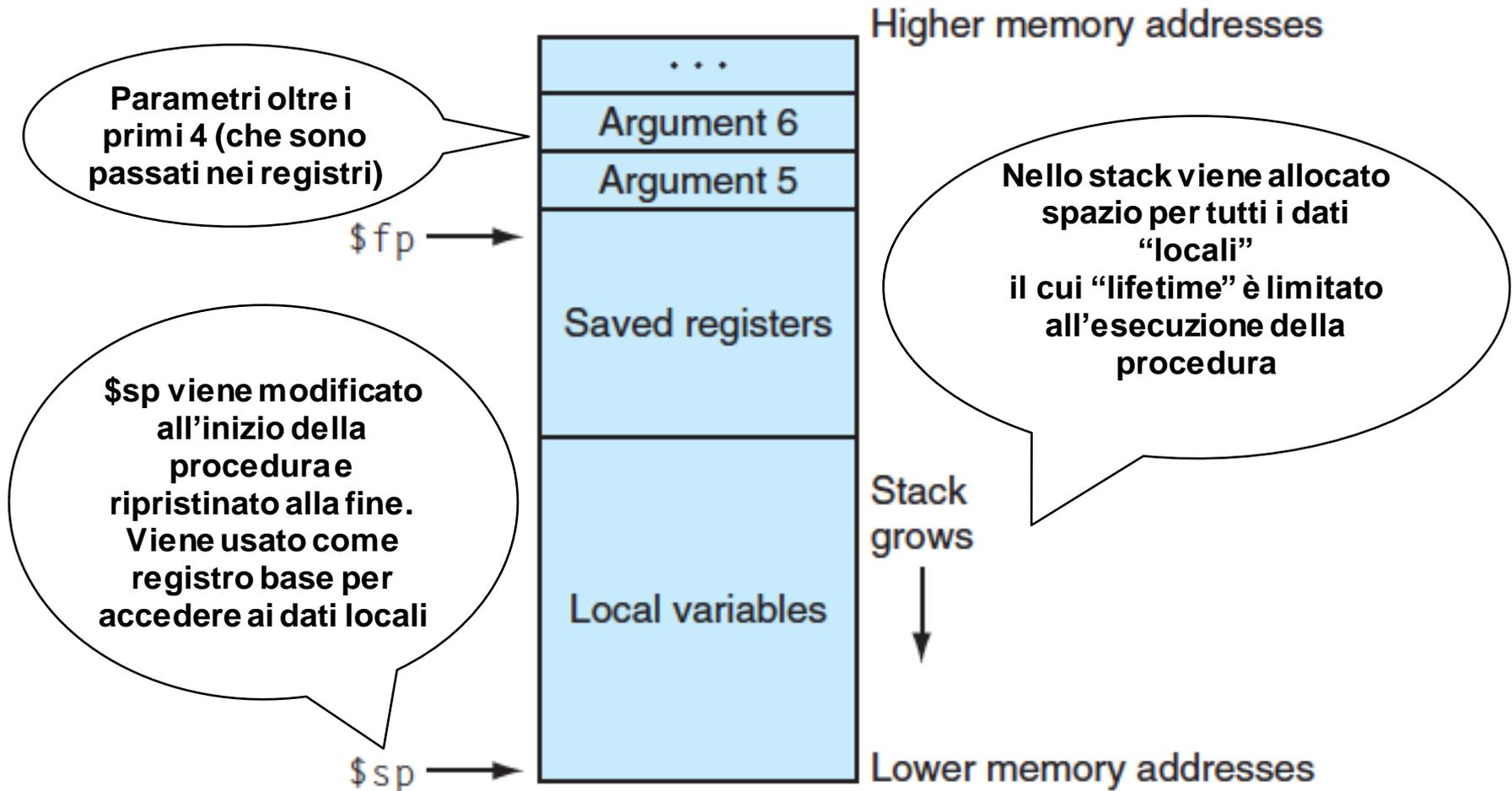
# Convenzioni di uso della memoria

- Convenzioni standard utilizzate da compilatori, assembleri, linker ... e programmatori
- Appendice A Hennesy-Patterson sez. A5 Fig. A.5.1



# Uso dello stack (complessivo)

- Appendice A Hennessy-Patterson sez. A5 Fig. A.5.1



# Uso dello stack (solo salvataggio registri)

- **Cosa fa la procedura?**
- **“Alloca” spazio nello stack**
  - Decrementa \$sp per lasciare in stack lo spazio necessario al salvataggio (1 word per ciascun registro da salvare)
  - (ricordare che lo stack cresce “verso il basso”)
  - Salva \$ra
  - Salva eventuali altri registri usando \$sp come registro base
  - .....
  - Ripristina i registri
  - Incrementa \$sp per riportarlo alla situazione iniziale
  - Jr \$ra (ritorno dalla procedura)
- **Approfondimenti:**
  - Parametri passati in stack
  - “Procedure frame”: l’insieme dei dati locali (\$fp)
  - Come indirizzare variabili locali? (per fortuna ci pensa il compilatore...)
- **Per capire bene: procedure recursive**



## \$sp e \$fp - osservazioni

- **Frame di stack (oppure di chiamata a procedura) – un blocco di memoria associato alla procedura**
- **\$sp – punta alla prima parola del frame**
- **\$fp – punta all’ultima parola del frame**
- **Un frame di solito e’ multiplo della parola doppia (8 byte)**
- **Esempio: un frame di 32 byte**
- **addi \$sp, \$sp, -32 # frame di stack di 32 byte**
- **addi \$fp, \$sp, 28 # imposta il frame pointer**
- **sw \$ra, 0(\$fp) # salva l’indirizzo di ritorno come primo  
# word nel frame sullo stack**
- **Appendice A – Sezione A.6**



## Procedura chiamante deve ...

- ... eseguire i seguenti passi prima di chiamare una procedura:
- Impostare gli argomenti da passare alla procedura in  $\$a0$ - $\$a3$ ; eventuali altri argomenti sono nella memoria o nello stack
- Salvare eventualmente i registri  $\$a0$ - $\$a3$  e  $\$t0$ - $\$t9$  in quanto la procedura chiamata puo' usare liberamente questi registri
- Chiamare la procedura tramite l'istruzione `jal nome_procedura`



## Procedura chiamata deve ...

- ... eseguire i seguenti passi appena e' stata chiamata:
- Allocare il suo stack frame ( $\$sp = \$sp - \text{dimensione frame procedura}$ )
- Salvare i valori disponibili nei registri  $\$s0-\$s7$ ,  $\$fp$ ,  $\$ra$  se intende usarle tali registri per la sua esecuzione; se per esempio la procedura non chiama un'altra procedura non e' necessario salvare il registro  $\$ra$
- Settare il frame pointer (che indica l'indirizzo dell'ultima parola del frame):  $\$fp = \$sp - \text{dimensione frame procedura} + 4$



## Procedura chiamata deve ...

- ... eseguire i seguenti passi quando ha finito la sua esecuzione:
- Mettere il valore di ritorno nei registri \$v0, \$v1
- Ripristinare i valori dei registri salvati sullo stack (\$s0–\$s7, \$fp, \$ra)
- Liberare lo spazio sullo stack:  $\$sp = \$sp + \text{dimensione frame procedura}$
- Eseguire l'istruzione jr \$ra



## Esercizio

- **Si chiede di scrivere un programma assembly che identifica il minimo dei valori di un array di numeri interi.**
- **Si chiede di scrivere un main che chiama una procedura: `min_array` che riceve come parametri in input l'indirizzo di un array di interi e la sua dimensione e ritorna il valore minimo tra gli elementi dell'array.**
- **La procedura `min_array` chiama una procedura: `min2numeri` che riceve come parametri in input due interi e ritorna il valore minimo tra i due numeri.**



## Soluzione – Procedura min2numeri

min2numeri:

# riceve in input 2 interi passati per valore in \$a0 e \$a1

slt \$t0, \$a0, \$a1 # \$t0 = 1 se \$a0 < \$a1, \$t0 = 0 altrimenti

beq \$t0, \$0, then

move \$v0, \$a0

j fine\_min2numeri

then: move \$v0, \$a1

fine\_min2numeri: jr \$ra



# Soluzione – Procedura min\_array

min\_array:

```
#salvare a0, a1, fp e ra sullo stack
#in un frame di 24 byte (6 word)
addi $sp, $sp, -24
sw $fp, 0($sp)
sw $ra, 4($sp)
sw $a0, 8($sp)
addi $fp, $sp, 20
sw $a1, 0($fp)
# fine del set di passi che la procedura
#deve fare prima di eseguire il suo compito
#consideriamo che l'array abbia più di un elemento
#altrimenti si deve considerare il caso con l'array vuoto
#e con l'array di 1 elemento
move $t0, $a0
move $t1, $a1
lw $a0, 0($t0)
lw $a1, 4($t0)
addi $t1, $t1, -2
addi $t0, $t0, 8
ciclo:    jal min2numeri
          addi $t1, $t1, -1
          bltz $t1, fine_array
          lw $a0, 0($t0)
          addi $t0, $t0, 4
          move $a1, $v0
          j ciclo
```

fine\_array:

```
#ripristino registri
#salvati sullo stack
lw $ra, 4($sp)
lw $a0, 8($sp)
lw $a1, 0($fp)
lw $fp, 0($sp)
#libera lo stack
addi $sp, $sp, 24
jr $ra
```



# Soluzione – main (I)

```
.data
    array:          .word -1, 10, 3, 57, -100, 7, 9, -10, 9, 0
    dim_array:      .word 10

.globl main
.text
main:              #anche il main è una procedura
                  #bisogna salvare almeno lo stack pointer e il return address e i parametri in ingresso
                  #sullo stack in un frame di 40 byte
                  addi $sp, $sp, -40
                  sw $fp, 0($sp)
                  sw $ra, 4($sp)
                  sw $a0, 8($sp)
                  sw $a1, 12($sp)
                  sw $a2, 16($sp)
                  sw $a3, 20($sp)
                  addi $fp, $sp, 36
                  # fine del set di passi che la procedura deve fare prima di eseguire il suo compito
                  la $a0, array
                  la $t0, dim_array
                  lw $a1, 0($t0)
                  jal min_array
                  #continua ...
```



## Soluzione – main (II)

```
#continua ...  
#scrivo il min sullo schermo  
move $a0, $v0  
li $v0, 1  
syscall  
  
#rispristino registri dallo stack  
  
lw $ra, 4($sp)  
lw $a0, 8($sp)  
lw $a1, 12($sp)  
lw $a2, 16($sp)  
lw $a3, 20($sp)  
lw $fp, 0($sp)  
addi $sp, $sp, 40  
  
jr $ra
```

