Architetture dei Calcolatori

Lezione 11 -- 10/12/2011

Procedure

Emiliano Casalicchio
emiliano.casalicchio@uniroma2.it

Fattoriale: risparmiamo sull'uso dei registri

- Rispetto alla soluzione precedente (lezione 10)
  - L'uso di $s0 e $s1 non è necessario
- Perche?

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

- Regole chiamata:
  - La Chiamante: Salva i registri $t0-$t9, $a0-$a3, $v0-$v1
  - La Chiamata: Salva i registri $s0-$s7, $fp e $ra

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
```

fact è proc. sia chiamante che chiamata

Esecuzione:
fact(3)=3*fact(2)=3*2*fact(1)=…=3*2*fact(0)=6

sp	fact(3)	fact(2)	fact(1)	fact(0)	fact(1)	fact(2)
sp=e00						
	ra=x1	ra=x1	ra=x1	ra=x1	ra=x1	ra=x1
sp=df4	a0=3	a0=3	a0=3	a0=3	a0=3	a0=3
		ra=x2	ra=x2	ra=x2	ra=x2	ra=x2
sp=de8		a0=2	a0=2	a0=2	a0=2	a0=2
			ra=x2	ra=x2	ra=x2	
sp=ddc			a0=1	a0=1	a0=1	
				ra=x2		
				a0=0		
				v0=1	a0=2	a0=3
				sp=ddc	ra=x2	ra=x1
				a0=1	sp=df4	sp=e00
				ra=x2	v0=2	v0=6
				sp=de8	j fact(2)	j main
				v0=1		
				j fact(1)		

```
fact:
    addi $sp, $sp, -8
    sw $ra, 4($sp)
    sw $a0, 0($sp)
    slti $t0, $a0, 1
    beq $t0, $zero, L1
    addi $v0, $zero, 1
    addi $sp, $sp, 8
    jr $ra
L1: addi $a0, $a0, -1
    jal fact
    lw $a0, 0($sp)
    lw $ra, 4($sp)
    addi $sp, $sp, 8
    mul $v0, $a0, $v0
    jr $ra
```

Homework

- Il seguente codice funzionerebbe lo stesso?

```
fact:
    addi $sp, $sp, -8
    sw $ra, 4($sp)
    sw $a0, 0($sp)
    slti $t0, $a0, 1
    beq $t0, $zero, L1
    addi $v0, $zero, 1
# non eseguiamo questa istruzione: addi $sp, $sp, 8
    jr $ra
L1:     addi $a0, $a0, -1
    jal fact
    lw $a0, 0($sp)
    lw $ra, 4($sp)
    addi $sp, $sp, 8
    mul $v0, $a0, $v0
    jr $ra
```

Esempio: Fibonacci

- 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)); //generic case
}
```

```
fib:    addi $sp,$sp,-12        # save registers on stack
        sw $a0, 0($sp)          # save $a0 = n
        sw $s0, 4($sp)          # save $s0
        sw $ra, 8($sp)          # save return address $ra
        bgt $a0,1, gen          # if n>1 then goto generic case
        move $v0,$a0            # out = in if n=0 or n=1 (base of rec)
```

```
sw $4, 0($29)               ; 9: sw $a0, 0($sp) # save $a0 = n
sw $16, 4($29)              ; 10: sw $s0, 4($sp) # save $s0
sw $31, 8($29)              ; 11: sw $ra, 8($sp) # save return address $ra
slti $1, $4, 2              ; 12: bgt $a0,1, gen # if n>1 then goto generic case
beq $1, $0, 12 [gen-0x0040004c]
addu $2, $0, $4             ; 13: move $v0,$a0 # output = input if n=0 or n=1
```

```
        move $s0,$v0            # save fib(n-1)
        addi $a0,$a0,-1           # set param to n-2
        jal fib                 # and make recursive call
        add $v0, $v0, $s0       # $v0 = fib(n-2)+fib(n-1)
rreg:   lw  $a0, 0($sp)         # restore registers from stack
        lw  $s0, 4($sp)         #
        lw  $ra, 8($sp)         #
        addi $sp, $sp, 12       # decrease the stack size
        jr $ra
```

```
fib:    addi $sp,$sp,-12        # save registers on stack
        sw $a0, 0($sp)          # save $a0 = n
        sw $s0, 4($sp)          # save $s0
        sw $ra, 8($sp)          # save return address $ra
        bgt $a0,1, gen          # if n>1 then goto generic case
        move $v0,$a0            # out = in if n=0 or n=1 (base of rec)
        j rreg                  # goto restore registers
gen:    addi $a0,$a0,-1         # param = n-1
        jal fib                 # compute fib(n-1)
        move $s0,$v0            # save fib(n-1)
        addi $a0,$a0,-1           # set param to n-2
        jal fib                 # and make recursive call
        add $v0, $v0, $s0       # $v0 = fib(n-2)+fib(n-1)
rreg:   lw  $a0, 0($sp)         # restore registers from stack
        lw  $s0, 4($sp)         #
        lw  $ra, 8($sp)         #
        addi $sp, $sp, 12       # decrease the stack size
        jr $ra
```

Esecuzione:
fib(3)=fib(2)+fib(1)=[fib(1)+fib(0)] + fib(1)

sp=e00	fib(3)*	fib(2) **	fib(1)	fib(0)	fib(2) **	fib(3)*	fib(1)	fib(3)*
	ra=2c	ra=2c	ra=2c	ra=2c	ra=2c	ra=2c	ra=2c	ra=2c
	s0=0	s0=0	s0=0	s0=0	s0=0	s0=0	s0=0	s0=0
sp=df4	a0=3	a0=3	a0=3	a0=3	a0=3	a0=3	a0=3	a0=3
		ra=60	ra=60	ra=60	ra=60		ra=6c	
		s0=0	s0=0	s0=0	s0=0		s0=1	
sp=de8		a0=2	a0=2	a0=2	a0=2		a0=1	
			ra=60	ra=6c				
			s0=0	s0=1				
sp=ddc			a0=1	a0=0				
			v0=1	v0=a0=0	v0=v0+s0=1	s0=v0=1	v0=1	**v0=v0+s0=2**
	a0=2	a0=1	s0=0 a0=1	a0=0 s0=0	s0=0 a0=2	a0=1	s0=1 a0=1	a0=3 s0=0
	fib(2)	fib(1)	ra=60	ra=6c	ra=60 sp=df4	s0=1	ra=6c	ra=2c
			sp=de8	v0=0	return fib(3)	fib(1)	sp=df4	sp=e00
			s0=v0=1	s0=1 a0=0			return fib(3)	
			a0=0	ra=6c sp=de8				
			fib(0)	return fib(2)				

ra=2c: syscall di uscita

ra=60: gen+8

ra=6c: gen+20

Homework

- Partendo dal codice discusso precedentemente:
  - Aggiungere le istruzioni necessarie a leggere l'input da tastiera e a stampare il risultato sullo standard output
  - Aggiungere le istruzioni necessarie per calcolare e stampare la sequenza dei numeri di fibonacci dato in input un numero N
- Scrivere la versione non ricorsiva della funzione per il calcolo del numero di fibonacci

Esempio: Vogliamo realizzare una procedura per l'ordinamento di un vettore

- 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);
}
```

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; }
```

Ad. es.

100 = 4

1000 = 4*2

10000 = 4*4

```
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
```

Homework

- Modificare la procedura swap(v,k) come segue:
  - se n è la lunghezza del vettore v garantire $k < n$

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

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 = v[]
       add $s3, $zero, $a1 # $s3 = $a1 = n
       # 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
```

```
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);
}
```

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 for2tst
```

```
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);
}
```

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
```

```
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);
}
```

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
```

Esempio: sort - 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)
          …
          # corpo loop esterno
          …
          addi $s0, $s0, 1          # i=i+1
          j for1tst
exit1:
```

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:
```

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
```

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

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

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
```

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

Homework: Scrivere il codice C

- 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

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(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, $t3# $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
```

Controlla se elem[i] è positivo

Esempio: copia parziale di un vettore - Varianti

Non usare $s0, $s1 e $s2 ma $a0, $a1 e $a2

Risparmio spazione nello stack

```
cp_pos: addi $sp, $sp, -8
        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
```

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

- Usare $t* al posto di $s*: evito di usare lo stack

```
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, $t4, 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
```

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

- Usando i puntatori

```
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
```

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

- Versione Finale: uso gli $a* e $v0 e non uso lo stack
  - Regole chiamata:
    - La Chiamante: Salva i registri $t0-$t9, $a0-$a3, $v0-$v1
    - La Chiamata: Salva i registri $s0-$s7, $fp e $ra

```
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)         # y[i]=*a1=*t6
        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
```

Homework

- Riscrivere il codice della procedura Sort in modo da:
  - ridurre il numero di locazioni di memoria dello stack che vengono utilizzati
  - ridurre il numero di registri utilizzati