Esercitazione 2 novembre 2011

ES1_MIPS

- Dato un vettore A di 100 interi memorizzati a partire dalla locazione 2000, scrivere un programma in assembler MIPS che scriva nella locazione 4000 il valore 1 se tutti gli interi nel vettore A sono uguali e scrive nella locazione 4000 il valore 0 altrimenti (cioè se non sono tutti uguali). Si richiede
- (1) la descrizione dell'algoritmo usato (in pseudocodice o programma C),
- (2) la descrizione dell'uso dei registri e
- (3) il programma in assembler commentato linea per linea.
- Non si può assumere la inizializzazione di nessun registro.

Quindi il nostro pseudocodice è il seguente:

```
for (i = 1; i < 100; i++) {      // per ogni elemento del vettore
if (A[i] != A[0])                // se abbiamo trovato un elemento diverso da
    A[0]...
goto Esci                        // allora possiamo saltare (fuori dal ciclo) a Esci
}
Esci: if (i != 100) {
// si esce dal ciclo per due motivi: se i != 100 allora siamo usciti con il goto
// scrivi 0 nella locazione 4000 perchè evidentemente c'è un elemento
    diverso dagli altri
else // altrimenti siamo usciti perchè i ha raggiunto il valore 100 e quindi
// scrivi 1 nella locazione 4000 perchè tutti gli elementi sono uguali al primo
    elemento di A
```

Pseudocodice più vicino all'Assembler, il programma diventa:

```
    i = 1
Cont: if (A[i] != A[0] )
    goto Esci;
    i = i +1
    if (i != 100) goto Cont;
Esci: if (i != 100) goto Scrivi0
    // scrivi il valore 1 in locazione 4000
    goto Fine:
Scrivi0: // scrivi il valore 0 in locazione 4000
Fine: // qualsiasi istruzione
```

Giusto per evitare una label aggiuntiva

- si può utilizzare un trucchetto: alla fine del for viene scritto nella locazione 4000 uno dei due risultati (diciamo che scriviamo un 1) e l'if lo usiamo solamente per riscrivere, se necessario, il valore 0 nella locazione 4000. Così il programma diventa:

```
    i = 1
Cont: if (A[i] != A[0] )
    goto Esci;
    i = i +1
    if (i != 100) goto Cont;
Esci: // scrivi il valore 1 in locazione 4000
    if (i == 100) goto Fine // avevamo ragione! andava scritto 1 in 4000, termina!
// scrivi il valore 0 in locazione 4000 // in caso contrario scriviamo 0 in 4000
Fine: // qualsiasi istruzione
```

Passiamo ora alla descrizione dei registri:

-

$10 indice i

$11 spiazzamento dell'indice i

$12 registro che contiene il valore di A[0] da confrontare man mano

$13 costante 1

$14 registro temporaneo per caricare A[i]

Infine, ecco il programma in Assembler:

```
    addi $10, $0, 1          # i = 1
    addi $11, $0, 4          # spiazzamento di i a 4
    lw $12, 2000($0)         # carico A[0]
    addi $13, $0, 1          # costante 1 in $13
    addi $15, $0, 100        # costante 100
Ciclo: lw $14, 2000($11)     # carica A[i]
    bne $14, $12, Esci       # if (A[i] != A[0]) goto Esci
    addi $10, $10, 1          # i = i + 1
    addi $11, $11, 4         # spiazzamento = spiazzamento + 4
    bne $10, $15, Ciclo      # if (i != 100) goto Ciclo
Esci: sw $13, 4000 ($0)      # scriviamo 1 come output
    beq $10, $15, Fine       # if (i != 100) goto Fine
    sw $0, 4000 ($0)         # scriviamo 0 come output
Fine: add $0, $0, $0         # istruzione fittizia per poter avere
                             una etichetta Fine
```

Es2_MIPS

Scrivere una funzione Assembly MIPS che, ricevendo in ingresso un intero, ne restituisca il valore assoluto.

```
int abs(int valore)
{
	if(valore<0)
	return -valore;
	else
	return valore;
}
```

- valore -> $4;
- il valore di ritorno viene inserito in $5

- Formato delle istruzioni?
- Modalità di indirizzamento?

Es3_MIPS (facoltativo)

- Tradurre in Assembler MIPS il codice C UTILIZZANDO la fissata assegnazione delle variabili ai registri.
- Inserire COMMENTI di spiegazione delle istruzioni Assembler.

```
n = 0 ;
while ( B[n] = = 0 ) {
    A[n] = B[n+1] + B[n] ;
n = n + 1 ; }
test = k + 5
if ( test < 0 )
    B[3] = A[0] ;
else
    B[0] = A[0] + A[2] ;
test = test + k ;
```

k -> $t5; n -> $t6 ; test -> $t7; A[] -> reg.base $s1; B[] -> reg.base $s2

ES4_MIPS (facoltativo)

Tradurre in procedura Assembler MIPS INDICANDO l'assegnazione delle variabili ai registri.

PROGRAMMA CHIAMANTE

```
j = k ;
sum = k + val ( j , k ) ;
k = j – sum
```

FUNZIONE CHIAMATA

```
int val ( int n, int s ) {
int test = n + 1 ;
…
if ( n < s )
return test ;
else
return n ; }
```