ASM

Esercitazione 5
Architettura degli elaboratori

Esercizio 1

- Considerando che i valori 10, 5, 20, 100 siano memorizzati rispettivamente nei registri $s0, $s1, $s2, $s3, qual è il valore contenuto nel registro destinazione con l'esecuzione delle seguenti istruzioni:

```
add $t0, $s0, $s1
add $t1, $s2, $s3
sub $t2, $t1, $t0
```

- Qual è la rappresentazione binaria delle istruzioni?

Esercizio 1 – soluzione

- Considerando che
  - $s0 contiene 10
  - $s1 contiene 5
  - $s2 contiene 20
  - $s3 contiene 100
- Il valore ritornato dalle istruzioni e salvato rispettivamente nei registri $t0, $t1 e $t2 è
  - **$t0 = 15**
  - **$t1 = 120**
  - **$t2 = 105**

- Per la rappresentazione binaria delle istruzioni servirà ricavare i numeri dei registri dal nome simbolico facendo riferimento alla tabella a pag A-24, App. A (nomi e numeri dei registri MIPS) – v.slide successiva

Esercizio 1 – soluzione

Facendo riferimento alla tabella a pag A-24, App. A (nomi e numeri dei registri MIPS):

- $t0 è il registro $8 (01000 in binario)
- $t1 è il registro $9 (01001 in binario)
- $t2 è il registro $10 (01010 in binario)
- $s0 è il registro 16 (10000 in binario)
- $s1 è il registro 17 (10001 in binario)
- $s2 è il registro 18 (10010 in binario)

- La rappresentazione binaria delle istruzioni è

add $t0, $s0, $s1 [opcode=0 e func=$32_{10}$]

000000 10000 10001 01000 00000 100000

add $t1, $s2, $s3 [opcode=0 e func=$32_{10}$]

000000 10010 10011 01001 00000 100000

sub $t2, $t1, $t0 [opcode=0 e func=$34_{10}$]

000000 01001 01000 01010 00000 100010

Esercizio 2

Quale valore assumono i registri $t0, $t1, $t2 dopo l'esecuzione del seguente frammento di codice?

```
.data  0x10010000
       numeri: .word 1, 2, 3
.text
       lw $t0, numeri
       la $t1, numeri              #pseudoistruzione
       lw $t2, 8($t1)
```

Esercizio 2 – soluzione

Con l'esecuzione del frammento di codice

```
.data       0x10010000          #opzionale indicare l'indirizzo di inizio del segmento!
numeri:     .word 1, 2, 3
.text
            lw $t0, numeri              #istruzione load word
            la $t1, numeri              #pseudoistruzione load address
            lw $t2, 8($t1)
```

avremo:

- in $t0, la word presente all'indirizzo indicato dall'etichetta numeri e quindi $t0 = 1
- in $t1, l'indirizzo dell'etichetta numeri e quindi $t1=0x1001000 poichè numeri è il primo dei dati presenti nel data segment che inizia all'indirizzo specificato dalla direttiva .data
- in $t2, la word che si trova 8 byte (2 word) dopo l'indirizzo uguale al contenuto di $t1 e quindi $t2 = 3

Esercizio 3

Raddoppiare i primi 37 valori memorizzati a partire dalla locazione di memoria il cui indirizzo è specificato dal contenuto del registro $t0 (in gergo, “puntata da $t0”)

Utilizziamo un ciclo per non riscrivere la stessa cosa 37 volte …

Esercizio 3 – soluzione

```
addi $t1, $zero, 37          #$t1 variabile di controllo del ciclo inizializzata a 37
perdue: lw $t2, 0($t0)       #carico in $t2 il primo valore in memoria all'indirizzo contenuto in $t0
        add $t2, $t2, $t2    #moltiplico per 2 sommando il contenuto del registro con se stesso
        sw $t2, 0($t0)       #salvo in memoria all'indirizzo da cui ho caricato nel registro
        addi $t1, $t1, -1    #decremento variabile controllo del ciclo ($t1) di 1
        beq $t1, $zero, fine #se $t1 ha raggiunto il valore 0, ho finito
        addi $t0, $t0, 4     #incremento $t0 di 4 byte → $t0 punta a successiva word in memoria
        j perdue             #altrimenti, esegue stesse istruzioni (su word successiva)
fine: …
```

N.B. nelle istruzioni lw e sw l'offset è un valore costante e NON può essere sostituito con un registro a cui cambiamo il valore del contenuto

Esercizio 4 (ciclo for)

Dato il frammento di codice assembly

```
.data
numeri:     .word 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
risultato:  .word 0
```

scrivere un programma assembly che calcoli la somma di 10 numeri disponibili in memoria all'etichetta "numeri"

Esercizio 4 (ciclo for) – soluzione

Calcolare la somma di 10 numeri disponibili in memoria

```
.data
numeri:    .word 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
risultato: .word 0
.text
        la $t7, numeri          #indirizzo di inizio
                                #numeri
        li $t6, 10              #numero di numeri
                                #da sommare
        add $s0, $zero, $zero   #somma = 0
        add $t5, $zero, $zero   #indice = 0
ciclo:  lw $t0, 0($t7)          #metto in $t0 il primo numero
        add $s0, $s0, $t0       #aggiungo il numero in $t0 a $s0
        addi $t7, $t7, 4        #mi sposto di 1 word (4 byte)
        addi $t5, $t5, 1        #incremento contatore del ciclo
        blt $t5, $t6, ciclo     #se ho eseguito il ciclo meno di
                                #10 volte, ripeto ciclo
        la $t7, risultato       #terminato il ciclo, uso $t7 per
                                #caricare l'indirizzo a cui
                                #salvare il risultato della somma
                                #dei 10 numeri, ora in $s0
        sw $s0, 0($t7)
```

Esercizio 5 (if-then-else)

- Scrivere un programma assembly che legge 3 valori che si trovano in memoria all'etichetta "valori" e memorizza in memoria, nella locazione subito dopo il terzo valore:
  - il risultato della somma dei tre valori, se il primo valore è positivo
  - il prodotto dei tre numeri, se il primo valore è negativo
  - l'AND del secondo e del terzo numero, se il primo è uguale a zero

Esercizio 5 – soluzione

```
.data     #direttiva che indica l'inizio del data segment
valori:   .word 1, 5, 2          #valori in memoria
          .word 0                #spazio in cui sara' salvato il risultato
.text     #direttiva che indica l'inizio del text segment
main:     la $t7, valori                  #carico nel registro $t7 l'indirizzo a cui si trova l'etichetta "valori" e quindi
                                          #il primo dato di dimensione di una word
          lw $t0, 0($t7)                  #carico in $t0 il primo valore
          lw $t1, 4($t7)                  #carico in $t1 il secondo valore (offset della load_word è 4 byte=1 word)
          lw $t2, 8($t7)                  #carico in $t2 il terzo valore (offset della load_word è 8 byte=2 word)
          bgt $t0, $zero, then            #istruzione branch greater_than_zero (se $t0>0, salto all'etichetta then)
          blt $t0, $zero, else            #istruzione branch less_than_zero (se $t0<0, salto all'etichetta else)
          and $s0, $t1, $t2               #viene eseguita quest'istruzione solo se non sono verificate le due
                                          #condizioni $t0<0 e $t0>0, quindi nel caso dell'opzione $t0=0 (metto il risultato
                                          #temporaneamente in $s0)
          j fine                          #eseguita solo quando $t0=0
else:                                     #scrive temporaneamente in $s0 il prodotto dei 3 numeri con due operazioni mul
          mul $s0, $t0, $t1
          mul $s0, $s0, $t2
          j fine
then:                                     #scrive temporaneamente in $s0 la somma dei 3 numeri con due operazioni add
          add $s0, $t0, $t1
          add $s0, $s0, $t2
          j fine
fine:     sw $s0, 12($t7)                 #scrittura in memoria del contenuto di $s0 3 word (=3*4 byte) dopo il valore dell'etichetta
                                          "valori" che era stata caricata nel registro $t7
jr $ra
```

Esercizio 6

Dato il seguente frammento di codice:

```
        li $t2, 8
ciclo:
        ..............
        sw $t2, 0($t0)
        addi $t0, $t0, 4
        addi $t2, $t2, -1
        j ciclo
fine_ciclo:
        jr $ra
```

Quale delle seguenti istruzioni va collocata al posto dei puntini per scrivere nell'array di word indirizzato da $t0 i valori 8,7,6,5,4,3,2,1,0?

1. bgtz $t2, fine_ciclo
2. bltz $t2, fine_ciclo
3. bgez $t2, fine_ciclo
4. blez $t2, fine_ciclo
5. beqz $t2, fine_ciclo
6. bnez $t2, fine_ciclo
7. nessuna delle altre risposte

Esercizio 6 – soluzione

Dato il seguente frammento di codice:

```
        li $t2, 8
ciclo:
        .............
        sw $t2, 0($t0)
        addi $t0, $t0, 4
        addi $t2, $t2, -1
        j ciclo
fine_ciclo:
        jr $ra
```

Quale delle seguenti istruzioni va collocata al posto dei puntini per scrivere nell'array di word indirizzato da $t0 i valori 8,7,6,5,4,3,2,1,0?

1. bgtz $t2, fine_ciclo #branch se $t2>0
2. **bltz $t2, fine_ciclo** #branch se $t2<0
3. bgez $t2, fine_ciclo #branch se $t2>0 o $t2=0
4. blez $t2, fine_ciclo #branch se $t2<0 o $t2=0
5. beqz $t2, fine_ciclo #branch se $t2=0
6. bnez $t2, fine_ciclo #branch se $t2<>0
7. nessuna delle altre risposte

Poichè vogliamo eseguire il ciclo 9 volte (e ad ogni ciclo decrementiamo $t2 di 1 partendo da un valore iniziale di 8), il branch dovrà essere fatto la prima volta che $t2 è minore di 0 mentre non dovrà essere fatto quando $t2 è maggiore e anche quando è uguale a 0

Esercizio 7

Scrivere un programma assembly che calcola la somma di un array di numeri interi. Il programma deve prevedere una procedura che riceve come parametri la dimensione dell'array e l'indirizzo d'inizio dell'array, e ritorna la somma degli elementi dell'array

Esercizio 7 – soluzione

```
.data
array:    .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
dim:      .word 10
somma:    .word 0
.text
main:
          lw $a0, dim          #primo parametro in $a0
          la $a1, array        #secondo parametro in $a1
          jal calcola          #jump and link
          sw $v0, somma

          li $v0, 10           #system call di terminazione
          syscall              #del programma
```

In questo caso, non essendoci procedure annidate, non è stato necessario fare spazio nello stack e salvarvi $ra
N.B. anche main è una procedura, il cui valore di ritorno è salvato nel registro $v0

```
# Procedura foglia chiamata da main
calcola:
move $t5, $a0          #copio in $t5 il primo parametro dim
move $t7, $a1          #copio in $t7 il secondo parametro (indirizzo
                       #d'inizio dell'array)
move $t1, $zero        #inizializzo a 0 $t1 (usato per somma)
move $t2, $zero        # inizializzo a 0 $t2 (usato come indice)
ciclo:
          lw $t6, 0($t7)       #leggo il numero
          add $t1, $t1, $t6    #lo sommo alla somma corrente
          addi $t2, $t2, 1     #incremento l'indice
          addi $t7, $t7, 4     #incremento l'indirizzo di 4 byte
          blt $t2, $t5, ciclo  #ripeto il ciclo se $t2 < dim
          move $v0, $t1        #eseguito quando $t2=0 per salvare in $v0
                               #il risultato della procedura
          jr $ra               #ritorno all'istruzione dopo jal che ha
                               #chiamato l'esecuzione della procedura
```