5 April 2013

04 – Laboratorio 03

Controllo del flusso di un programma
Jump address table

I. Frosio

Sommario

- If... Then...
- If... Then... Else...
- For...
- Break...
- While... Do
- Do... While
- Switch (jump address table)

Es. 3.1 (if... Then...)

- Si traduca in assembly il seguente codice:

```
a = <numero a caso>
b = <numero a caso>
c = 0;
If (a<b)
    c=a+b
```

- Non si utilizzino pseudo-istruzioni (es. blt), ma solo istruzioni dell'ISA (bne, beq, slt).
- Si verifichi la correttezza del valore di c alla fine dell'esecuzione. Per semplicità, si supponga che a, b e c siano associate ai registri $t0, ..., $t2.

Es. 3.1 - Soluzione


```
main:
        addi $t0, $zero, 5        # a = $t0 = 5
        addi $t1, $zero, 27       # b = $t1 = 27
        addi $t2, $zero, 0        # c = $t2 = 0

        slt $t3, $t0, $t1         # $t3 = 1 iff $t0<$t1
        beq $t3, $zero, End       # if $t3==0, goto End:
        add $t2, $t0,$t1                  # Then...
End:
```

Es. 3.2 (if... Then... Else...)

- Si traduca in assembly il seguente codice:

```
a = <numero a caso>
b = <numero a caso>
c = 0;
If (a<b)
    c=a+b;
else
    c=a*b;
```

- Non si utilizzino pseudo-istruzioni (es. blt), ma solo istruzioni dell'ISA (bne, beq, slt).
- Si verifichi la correttezza del valore di c alla fine dell'esecuzione. Per semplicità, si supponga che a, b e c siano associate ai registri $t0, ..., $t2.

Es. 3.2 - Soluzione


```
main:
        addi $t0, $zero, 5  # a = $t0 = 5
        addi $t1, $zero, 27           # b = $t1 = 27
        addi $t2, $zero, 0  # c = $t2 = 0

If:
        slt $t3, $t0, $t1             # $t3 = 1 iff $t0<$t1
        beq $t3, $zero, Else          # if $t3==0, goto Else

Then: add $t2, $t0,$t1                # Then...
        j End                         # Goto End:

Else:  mult $t0, $t1                  # HI, LO =  $t0*$t1
        mflo $t2                      # $t2 = LO

End:
```

Es. 3.3 (For...)

- Si scriva il codice che calcola la somma dei primi (N-1) [ovvero da 0 a (N-1) compreso] numeri elevati al quadrato (N definito dal programmatore).

- Hint:

```
N=50;
Sum=0;
for (n=0; n<N; n++)
    Sum+=n*n;
```

Es. 3.3 - Soluzione


```
main:
        addi $s0, $zero, 100     # N= $s0 = 100, number of samples

        addi $t0, $zero, 0       # $t0 = 0, counter i
        addi $s1, $zero, 0       # $s0 = 0, init sum
for:
        beq $t0, $s0, end        # if $t0 == $s0 (i==N), loop ends
        mult $t0, $t0            # HI, LO = $t0 * $t0
        mflo $t1                 # $t1 = LO
        add $s1, $s1, $t1        # $s1 = $s1 + $t1 (Sum = Sum+i*i)
        addi $t0, $t0, 1         # $t0 = $t0 + 1
        j for                    # loop

end:
```

Es. 3.4 (Break…)

- Si scriva il codice che calcola la somma dei primi N numeri elevati al quadrato (N definito dal programmatore).
- Nel caso in cui l'i-esimo numero da aggiungere (i*i) sia multiplo di 200, si termini prematuramente il ciclo for.

```
Sum=0;
for (i=1; i<N; i++)
{
    If ((i*i)%200==0)
        Break;
    Sum+=i*i;
}
```

Es. 3.4 - soluzione

```
main:
        addi $s0, $zero, 100      # N= $s0 = 100, number of samples

        addi $t0, $zero, 1        # $t0 = 1, counter i
        addi $s1, $zero, 0        # $s0 = 0, init sum
        addi $t3, $zero, 200      # $t3 = 200
for:
        beq $t0, $s0, end         # if $t0 == $s0 (i==N), loop ends
        mult $t0, $t0                         # HI, LO = $t0 * $t0
        mflo $t1                  # $t1 = LO

        div $t1, $t3              # HI, LO = t1/200
        mfhi $t2                  # $t2 = HI (remainder)
        beq $t2, $zero, end       # if ($t2/200 == 0), break

        add $s1, $s1, $t1         # $s1 = $s1 + $t1 (Sum = Sum+i*i)
        addi $t0, $t0, 1          # $t0 = $t0 + 1
        j for                     # loop

end:
```

Es. 3.5 (While... Do)

- Si scriva il codice che effettua la somma dei primi N numeri elevati al quadrato, fermandosi nel momento in cui tale somma eccede il valore di 5500.
- Potenzialmente il loop potrebbe non essere eseguito nemmeno una volta (si verifichi la correttezza del codice inizializzando ad esempio il valore della somma a 100000).

Es. 3.5 - Soluzione

```
main:

        addi $t0, $zero, 0        # $t0 = 0, counter i
        addi $s0, $zero, 0        # $s0 = 0, init sum

while:
        slti $t1, $s0, 5500       # if $s0<5500, $t1=1
        beq $t1, $zero end        # if $s0>=5500, goto end

        mult $t0, $t0             # HI, LO = $t0 * $t0
        mflo $t2                  # $t2 = $t0, $t0
        add $s0, $s0, $t2         # $s0 += $t0^2

        addi $t0, $t0, 1          # $t0 += 1
        j while                   # loop

end:
```

Es. 3.6 (Do... While)

- Si riscriva il codice dell'esercizio predente per implementare il costrutto Do... While invece del costrutto While... Do.

Es. 3.6 - Soluzione

```
main:

        addi $t0, $zero, 1          # $t0 = 0, counter i
        addi $s1, $zero, 0          # $s0 = 0, init sum

do:
        mult $t0, $t0               # HI, LO = $t0 * $t0
        mflo $t2                    # $t2 = $t0, $t0
        add $s0, $s0, $t2           # $s0 += $t0^2

        addi $t0, $t0, 1            # $t0 += 1

        addi $t1, $zero, 0          # $t1 = 0
        slti $t1, $s0, 5500         # if $s0<5500, $t1=1
        bne $t1, $zero, do          # if $s0<5000, goto do

end:
```

Costrutto switch

- Può essere implementata mediante una serie di *if-then-else*
- Alternativa: uso di una *jump address table* cioè di una tabella che contiene una serie di indirizzi di istruzioni alternative
- **switch(k)**
- **{**
- **case 0: f = i + j; break;**
- **case 1: f = g + h; break**
- **case 2: f = g - h; break;**
- **case 3: f = i - j; break;**
- **default: break;**
- **}**

Jump address table

Byte address

Byte address	Memoria principale (RAM)
t4 + 12	L3
t4 + 8	L2
t4 + 4	L1
t4	L0

Memoria principale (RAM)

Enumero degli oggetti tramite un indice k.

Questi oggetti sono gli indirizzi del codice da eseguire per ogni caso dello switch - sono memorizzati a distanza di 4 byte.

In pratica un array di inidirizzi.

STRUTTURA SWITCH/CASE OTTIMIZZATA

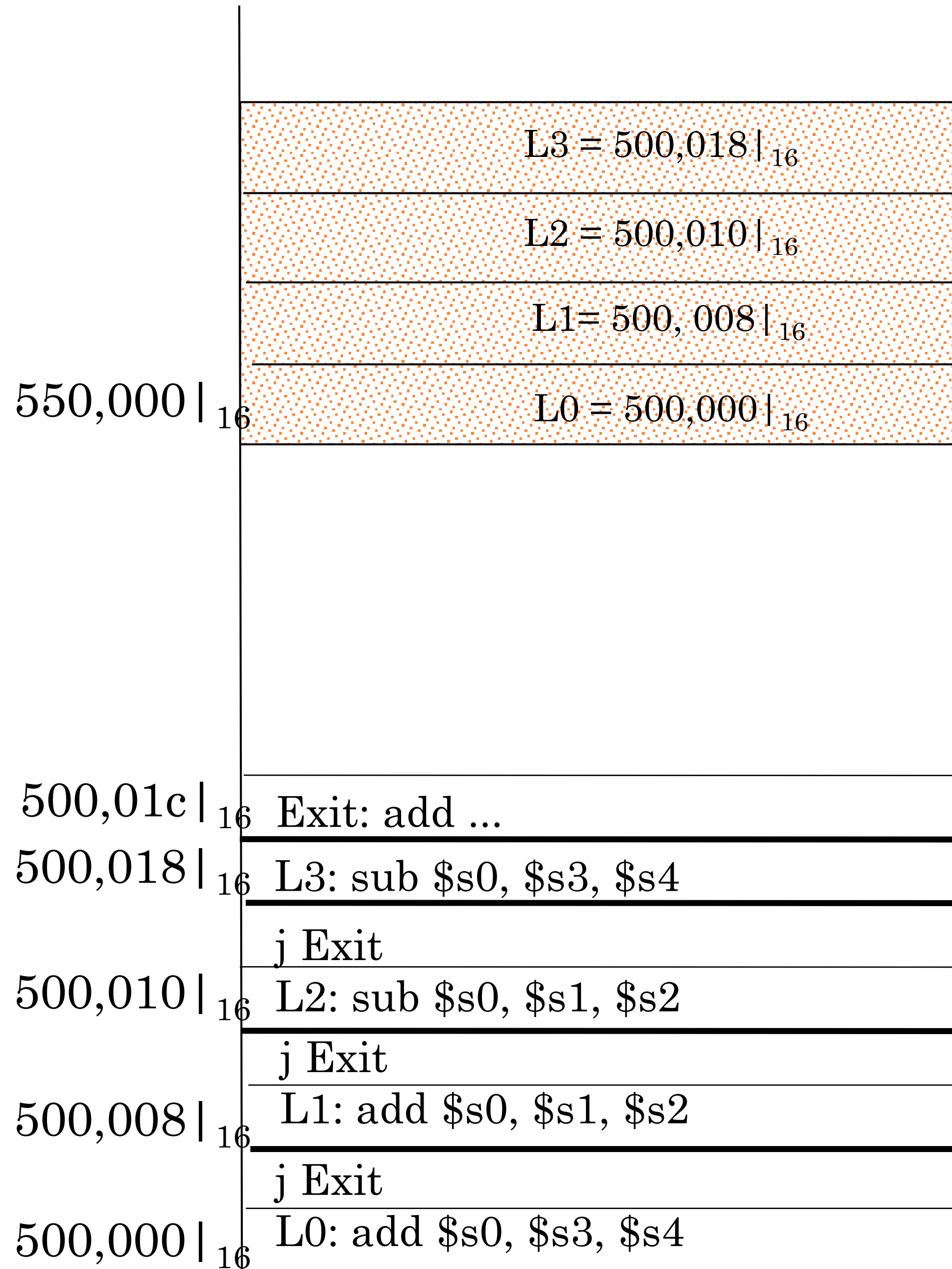
```
#$s0, .., $s5 contengono f,..,k
#$t4 contiene lo start address della jump address table (che si
#               suppone parta da k = 0).

#verifica prima i limiti (default)
   slt  $t3, $s5, $zero
   bne  $t3, $zero, Exit                    # if k<0, Exit
   slti $t3, $s5, 4
   beq  $t3, $zero, Exit                    #if k>4, Exit
#case vero e proprio
   muli    $t1, $s5, 4                      # t1 = k*4 (offset)
   add     $t1, $t4, $t1                    # t1 = address of A[k]
   lw      $t0, 0($t1)
   jr      $t0                              # jump to A[k]


L0: add $s0, $s3, $s4
    j Exit
L1: add $s0, $s1, $s2
    j Exit
L2: sub $s0, $s1, $s2
    j Exit
L3: sub $s0, $s3, $s4
Exit:
```

Esempio

```
switch(k)
 {
   case 0:  f = i + j; break;
   case 1:  f = g + h; break
   case 2:  f = g - h; break;
   case 3:  f = i - j; break;
  default:  break;
 }
```

k -> $s5
550,000 | 16 -> $t4

```
    ……. . .
    muli   $t1, $s5, 4
    add    $t1, $t4, $t1
    lw     $t0, 0($t1)
    jr     $t0
L0: add $s0, $s3, $s4
    j Exit
L1: add $s0, $s1, $s2
    j Exit
L2: sub $s0, $s1, $s2
    j Exit
L3: sub $s0, $s3, $s4
Exit:
    add ...
```



Es. 3.7 – Switch (Jump Address Table)

- Si realizzi, utilizzando una Jump Addres Table, il codice Assembly che traduce la seguente porzione di codice:

a= <numero scelto dal programmatore tra 0, 1 e 2>

b= <numero scelto dal programmatore tra 0, 1 e 2>

Switch (a*b)

Case 0: a=1; b= 2; c=0; break;

Case 1: a=0; c=0; c=100; break;

Case 2: c=a; a=1; b=0.

d= a + b + c;

- Hint: Per “salvare” in un registro l’indirizzo di memoria associato all’etichetta “etich:”, è sufficiente utilizzare:

  la $t0, etich   # la ← &etich:

- Hint: Per allocare spazio in memoria per la Jump Address Table, possiamo usare la riga:

  .data

  lookup: .space 12   # alloca 12 bytes in memoria

Es. 3.7 – Soluzione (1)


```
.data                # Static data here
.align 2
JAT: .space 12       # Allocate 12 bytes for the Jump Address Table

.text                # code here
main:

        # Jump Address Table
        la $t0, JAT            # $t0 contains the JAT base address
        la $t1, case0          # $t1 contains the address of case0
        sw $t1, 0($t0)         # Store $t1 in JAT[0]
        la $t1, case1          # $t1 contains the address of case1
        sw $t1, 4($t0)         # Store $t1 in JAT[1]
        la $t1, case2          # $t1 contains the address of case2
        sw $t1, 8($t0)         # Store $t1 in JAT[2]

        addi $s0, $zero, 1     # a = $s0 = 0
        addi $s1, $zero, 1     # b = $s1 = 1
```

Es. 3.7 – Soluzione (1)


```
mult $s0, $s1          # HI, LO = $s0*$s1 = a*b
mflo $t1               # $t1 = a*b

addi $t4, $zero, 4     # $t4 = 4
mult $t1, $t4          # HI, LO = $t1*4
mflo $t2               # $t2 = $t1*4

add $t3, $t2, $t0      # $t3 contains the JAT proper addres
lw $t5, 0($t3)         # Load the address contained in the JAT

# Jump to the case, save ra register
jal $t5                # Jump to case0, 1, 2

ori $7, $zero, $zero #nop
add $s3, $s0, $s1      # d = $s3 = $s0 + $s1 = a+b
add $s3, $s3, $s2      # d += c

j end                  # Jump to end
```

Es. 3.7 – Soluzione (1)


```
case0:
        addi $s0, $zero, 1      # a = $s0 = 1
        addi $s1, $zero, 2      # b = $s1 = 2
        addi $s2, $zero, 0      # c = $s2 = 0
        jr $ra                  # jump to return address (in ra)

case1:
        addi $s0, $zero, 0                  # a = $s0 = 0
        addi $s1, $zero, 0                  # b = $s1 = 0
        addi $s2, $zero, 100    # c = $s2 = 100
        jr $ra                              # jump to return address (in ra)

case2:
        add $s2, $zero, $s0     # c = a
        addi $s0, $zero, 1                  # a = $s0 = 1
        addi $s1, $zero, 0                  # b = $s1 = 0
        jr $ra                              # jump to return address (in ra)

end:
```