

Catena programmatica

Esercitazione 6
Architettura degli elaboratori

Esercizio 1

Dato il seguente frammento di codice:

```
.data 0x10010000
```

ciao:

```
    .ascii "Ciao Ciao."
```

mamma:

```
    .ascii "Mamma"
```

t:

```
    .space 6
```

eol:

```
    .ascii "1"
```

Quale valore assume l'etichetta "t"?

Quale valore assume l'etichetta "eol"?

Esercizio 1 – soluzione

- Dato il seguente frammento di codice:

```
.data 0x10010000
```

```
ciao:
```

```
    .ascii "Ciao Ciao."
```

```
mamma:
```

```
    .ascii "Mamma"
```

```
t:
```

```
    .space 6
```

```
eol:
```

```
    .ascii "1"
```

Quale valore assume l'etichetta "t"?

L'etichetta "t" assume valore 0x10010010, ovvero è l'indirizzo 10+1+5 (=16) dopo l'inizio del segmento data (che inizia dall'indirizzo 0x10010000).

Infatti la direttiva ".ascii" aggiunge anche il carattere di fine stringa mentre la direttiva ".ascii" no

Quale valore assume l'etichetta "eol"?

L'etichetta eol inizia 6 byte dopo l'etichetta t, ovvero all'indirizzo 0x10010016

Esercizio 2

- Cosa sono le pseudoistruzioni e come si comporta un assemblatore quando ne incontra una?

Esercizio 2 – soluzione

- Cosa sono le pseudoistruzioni e come si comporta un assembler quando ne incontra una?
- Le pseudoistruzioni sono istruzioni non implementate dall'hardware (cioè, non native dell'ISA di MIPS)
- Sono tradotte dall'assembler in sequenze di istruzioni native
- Secondo le convenzione d'uso dei registri, il registro \$1 (\$at) è riservato a questo scopo

Esercizio 4

Quali direttive assembly permettono di definire str in modo che possa essere visibile da altri moduli del programma?

In quale porzione della memoria sarà salvata str?

Esercizio 4 – soluzione

Le direttive assembly che permettono di definire str in modo che possa essere visibile da altri moduli del programma sono:

- `.globl str` `#visibile all'esterno`
- `.extern str 16` `#visibile all'esterno e di 16 byte`

La porzione della memoria in cui sarà salvata str è quella riservata ai dati statici

N.B. il simulatore QTSPIM potrebbe memorizzare i dati statici a partire da un indirizzo diverso (e.g. `0x10010000`) da quello indicato dal libro di testo (pag. A-21) che si riferisce a MIPS

 MIPS systems typically dedicate a register (`$gp`) as a *global pointer* to the static data segment. This register contains address `10008000hex`, 

Esercizio 5

Dato il seguente programma (non ci sono istruzioni prima o dopo quelle qui elencate)

```
.data 0x10010000
```

```
.text
```

```
#qui ci sono 2 istruzioni, nessuna pseudoistruzione e non ci sono definizioni o uso di label
```

```
j primaLabel
```

```
secondaLabel:
```

```
#qui ci sono 3 istruzioni, nessuna pseudoistruzione e non ci sono definizioni o uso di label
```

```
terzaLabel:
```

```
#qui ci sono 4 istruzioni, nessuna pseudoistruzione e non ci sono definizioni o uso di label
```

```
j secondaLabel
```

A quante delle label presenti, l'assemblatore assocerà correttamente un indirizzo?

Quante e quali label saranno "unresolved" dopo l'assemblaggio?

Se al simbolo `secondaLabel` è associato il valore `0x00400030`, a quale valore è associato il simbolo "terzaLabel"?

Esercizio 5 – soluzione

Dato il seguente programma (non ci sono istruzioni prima o dopo quelle qui elencate)

```
.data 0x10010000
```

```
.text
```

```
#qui ci sono 2 istruzioni, nessuna pseudoistruzione e non ci sono definizioni o uso di label
```

```
j primaLabel
```

```
secondaLabel:
```

```
#qui ci sono 3 istruzioni, nessuna pseudoistruzione e non ci sono definizioni o uso di label
```

```
terzaLabel:
```

```
#qui ci sono 4 istruzioni, nessuna pseudoistruzione e non ci sono definizioni o uso di label
```

```
j secondaLabel
```

L'assemblatore assocerà correttamente un indirizzo a 2 label, ovvero "secondaLabel" e "terzaLabel"

Sarà invece "unresolved" dopo l'assemblaggio l'etichetta "primaLabel" che viene usata ma non dichiarata

Al simbolo "terzaLabel" verrà associato il valore $0x00400030 + 12_{10} (=3*4) \rightarrow 0x0040003c$ poichè si trova 3 istruzioni (ciascuna della dimensione di una word) dopo secondaLabel

Esercizio 6

- Qual è l'ordine corretto di esecuzione dei passi necessari per trasformare un programma scritto in un linguaggio di alto livello in un programma eseguibile?
 - Caricare, compilare, assemblare e linkare
 - Compilare, assemblare, linkare e caricare
 - Compilare, assemblare e linkare
 - Assemblare, compilare, linkare e caricare
- Qual è l'ordine corretto di esecuzione dei passi necessari per trasformare un programma scritto in un linguaggio di alto livello in un programma in esecuzione su un calcolatore?
 - Caricare, compilare, assemblare e linkare
 - Compilare, assemblare, linkare e caricare
 - Compilare, assemblare e linkare
 - Assemblare, compilare, linkare e caricare
- e per ottenere un file oggetto?
 - Caricare, compilare, assemblare e linkare
 - Compilare, assemblare e linkare
 - Compilare, assemblare, linkare e caricare
 - Compilare e assemblare

Esercizio 6 – soluzione

- Qual è l'ordine corretto di esecuzione dei passi necessari per trasformare un programma scritto in un linguaggio di alto livello in un programma eseguibile?
 - Caricare, compilare, assemblare e linkare
 - Compilare, assemblare, linkare e caricare
 - **Compilare, assemblare e linkare**
 - Assemblare, compilare, linkare e caricare
- Qual è l'ordine corretto di esecuzione dei passi necessari per trasformare un programma scritto in un linguaggio di alto livello in un programma in esecuzione su un calcolatore?
 - Caricare, compilare, assemblare e linkare
 - **Compilare, assemblare, linkare e caricare**
 - Compilare, assemblare e linkare
 - Assemblare, compilare, linkare e caricare
- e per ottenere un file oggetto?
 - Caricare, compilare, assemblare e linkare
 - Compilare, assemblare e linkare
 - Compilare, assemblare, linkare e caricare
 - **Compilare e assemblare**

Esercizio 7

- Quali informazioni sono output di un assembler?
- Quali informazioni sono output di un linker?

Esercizio 7 – soluzione

- L'output di un assembler è il file oggetto (**object file**), il cui contenuto è:
 - **object file header** – dimensione e posizione delle altre parti del file
 - **text segment** – codice macchina delle istruzioni che compongono il programma contenuto nel file (alcune potrebbero far riferimento a riferimenti non risolti - unresolved references)
 - **data segment** – rappresentazione binaria dei dati contenuti nel file (potrebbero essere incompleti a causa di riferimenti a etichette/label che si trovano in altri file)
 - **relocation information** – istruzioni e dati che dipendono da riferimenti assoluti
 - **symbol table** – associa gli indirizzi delle etichette/label globali (visibili all'esterno del file) agli indirizzi e elenca i riferimenti non risolti (unresolved references)
 - **debugging information** – descrizione sintetica (usabile, ad esempio, da un programma di debugging)
- L'output di un linker è il programma eseguibile (**executable file**). Un programma eseguibile ha lo stesso formato del file oggetto MA *non contiene riferimenti non risolti e le relocation information*

Esercizio 8

- Due procedure (A e B) sono state assemblate separatamente e i file oggetto risultanti sono linkati mettendo A prima di B
- Sapendo che:
 - la dimensione del testo di A è di 0x200 e dei suoi dati di 0x10
 - la dimensione del testo di B è di 0x100 e dei suoi dati di 0x30
- Qual è l'indirizzo base a partire dal quale sarà allocato il testo di A nel file eseguibile?
- Qual è, nel file oggetto di A, il valore del campo address di una chiamata alla procedura B (jal B)?
- Qual è, nel file eseguibile, il valore del campo address di una chiamata in A alla procedura B?

Esercizio 8 – soluzione

- Due procedure (A e B) sono state assemblate separatamente e i file oggetto risultanti sono linkati mettendo A prima di B
- Sapendo che:
 - la dimensione del testo di A è di 0x200 e dei suoi dati di 0x10
 - la dimensione del testo di B è di 0x100 e dei suoi dati di 0x30

- Qual è l'indirizzo base a partire dal quale sarà allocato il testo di A nel file eseguibile?

0x0040 0000 (subito dopo area riservata – v. pag A-20)

- Qual è, nel file oggetto di A, il valore del campo address di una chiamata alla procedura B (jal B)?

indefinito, perchè sarà noto e inserito, solo quando A e B verranno uniti dal linker nell'eseguibile

- Qual è, nel file eseguibile, il valore del campo address di una chiamata in A alla procedura B?

Sapendo che B si trova dopo A e che A occupa uno spazio di 0x200, B inizierà all'indirizzo

0x0040 0200

Esercizio 9

- Cosa deve fare un assembler a due passi quando durante il primo passo:
 - incontra un'istruzione preceduta da un'etichetta?
E.g. etichetta: add \$t0, \$t1, \$t2
 - incontra un'istruzione non preceduta da un'etichetta in cui è presente l'uso di un simbolo (etichetta risolta)?
E.g. j pippo
 - incontra un'istruzione senza etichetta in cui non sono presenti dei simboli?

Esercizio 9 – soluzione

- Cosa deve fare un assemblatore a due passi quando durante il primo passo:
 - incontra un'istruzione preceduta da un'etichetta?
E.g. etichetta: add \$t0, \$t1, \$t2
scrive l'etichetta e il corrispondente valore (riferimento assoluto) nella tabella dei simboli
 - incontra un'istruzione non preceduta da un'etichetta in cui è presente l'uso di un simbolo (etichetta risolta)?
E.g. j pippo
cerca il simbolo nella tabella dei simboli e lo sostituisce con il valore corrispondente (se presente)
 - incontra un'istruzione senza etichetta in cui non sono presenti dei simboli?
nulla

Esercizio 10

- Cosa è necessario fare per ottenere un programma eseguibile se le istruzioni che compongono il programma sono scritte in diversi file?

Esercizio 10 – soluzione

- Cosa è necessario fare per ottenere un programma eseguibile se le istruzioni che compongono il programma sono scritte in diversi file?
- Oltre ad assemblare i file separatamente, serve fare tutto ciò che fa il linker, cioè:
 - Ricercare le librerie usate dal programma (e.g. syscall)
 - Determinare (leggendolo dall'object header) la porzione di memoria che dovrà essere occupata dal codice di ogni modulo e riposizionare le istruzioni di ogni modulo sistemando gli eventuali indirizzi assoluti
 - Risolvere i riferimenti tra file
 - ...

Esercizio 11

- Un riferimento non risolto (“unresolved reference”) può indicare un errore del programmatore. Quando?
 - Sempre
 - Se non è risolta dopo che è stato assemblato il file che contiene l’uso del simbolo
 - Se non è risolta dopo che sono stati assemblati tutti i file che compongono il programma
 - Se non è risolta dopo che il linker ha terminato la sua attività

Esercizio 11 – soluzione

- Un riferimento non risolto (“unresolved reference”) può indicare un errore del programmatore. Quando?
 - Sempre
 - Se non è risolta dopo che è stato assemblato il file che contiene l’uso del simbolo
 - Se non è risolta dopo che sono stati assemblati tutti i file che compongono il programma
 - **Se non è risolta dopo che il linker ha terminato la sua attività**

Esercizio 12

- Se un programma è scritto in due file, quando un simbolo dichiarato come global in un file risolve un riferimento utilizzato in un altro file?

Esercizio 12 – soluzione

- Se un programma è scritto in due file, quando un simbolo dichiarato come global in un file risolve un riferimento utilizzato in un altro file?
 - Quando entrambi fanno riferimento ad un'etichetta con lo stesso nome

- Esempio:

- mean_lib.asm: ...
- vector_mean.asm: ... usa mean (dichiarata dove? come?)

- Valore etichetta mean

... dopo assemblaggio (e linking) vector_mean.asm

```

QtSpim
-----
FP Regs Int Regs [16]
-----
Int Regs [16]
-----
User Text Segment [00400000]..[00440000]
PC = 0
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000ff
10
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 0
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 1
R5 [a1] = 7ffffd
R6 [a2] = 7ffffd
e0
R7 [a3] = 0
R8 [t0] = 0
R9 [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
-----
[00400000] 8fa40000 lw $4, 0($29) ; 175: lw $a0, 0($sp)
[00400004] 27a50004 addiu $5, $29, 4 ; 176: addiu $a1, $sp, 4
[00400008] 24a60004 addiu $6, $5, 4 ; 177: addiu $a2, $a1, 4
[0040000c] 00041080 sll $2, $4, 2 ; 178: sll $v0, $a0, 2
[00400010] 00c23021 addu $6, $6, $2 ; 179: addu $a2, $a2, $v0
[00400014] 0c10000a jal 0x00400028 [main] ; 180: jal main
[00400018] 00000000 nop ; 181: nop
[0040001c] 3402000a ori $2, $0, 10 ; 183: li $v0, 10
[00400020] 0000000c syscall ; 184: syscall
[00400024] 00000000 nop ; 186: nop
[00400028] 3c041001 lui $4, 4097 [array] ; 14: la $a0, array
[0040002c] 3c011001 lui $1, 4097 [length] ; 15: la $t0, length
[00400030] 34280005 ori $8, $1, 5 [length]
[00400034] 81050000 lb $5, 0($8) ; 16: lb $a1, ($t0)
[00400038] 23bdfffc addi $29, $29, -4 ; 18: addi $sp, $sp, -4
[0040003c] afbf0000 sw $31, 0($29) ; 19: sw $ra, 0($sp)
[00400040] 0c000000 jal 0x00000000 [mean] ; 21: jal mean
[00400044] 8fbf0000 lw $31, 0($29) ; 23: lw $ra, 0($sp)
[00400048] 23bd0004 addi $29, $29, 4 ; 24: addi $sp, $sp, 4
[0040004c] 2841000a slti $1, $2, 10 ; 26: blt $v0, 10, else
[00400050] 14200006 bne $1, $0, 24 [else-0x00400050]
[00400054] 3c011001 lui $1, 4097 [stringa] ; 27: la $a0, stringa
[00400058] 34240006 ori $4, $1, 6 [stringa]
[0040005c] 34020004 ori $2, $0, 4 ; 28: li $v0, 4 #system
call per print string
[00400060] 0000000c syscall ; 29: syscall
[00400064] 03e00008 jr $31 ; 30: jr $ra
[00400068] 00022021 addu $4, $0, $2 ; 33: move $a0, $v0
[0040006c] 34020001 ori $2, $0, 1 ; 34: li $v0, 1 #system
call per print integer
[00400070] 0000000c syscall ; 35: syscall
[00400074] 03e00008 jr $31 ; 36: jr $ra
  
```

... dopo assemblaggio mean_lib.asm

```

QtSpim
-----
FP Regs Int Regs [16]
-----
Int Regs [16]
-----
User Text Segment [00400000]..[00440000]
PC = 0
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000ff
10
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 0
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 1
R5 [a1] = 7ffffd
R6 [a2] = 7ffffd
e0
R7 [a3] = 0
R8 [t0] = 0
R9 [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
-----
[00400000] 8fa40000 lw $4, 0($29) ; 175: lw $a0, 0($sp)
[00400004] 27a50004 addiu $5, $29, 4 ; 176: addiu $a1, $sp, 4
[00400008] 24a60004 addiu $6, $5, 4 ; 177: addiu $a2, $a1, 4
[0040000c] 00041080 sll $2, $4, 2 ; 178: sll $v0, $a0, 2
[00400010] 00c23021 addu $6, $6, $2 ; 179: addu $a2, $a2, $v0
[00400014] 0c10000a jal 0x00400028 [main] ; 180: jal main
[00400018] 00000000 nop ; 181: nop
[0040001c] 3402000a ori $2, $0, 10 ; 183: li $v0, 10
[00400020] 0000000c syscall ; 184: syscall
[00400024] 00000000 nop ; 186: nop
[00400028] 3c041001 lui $4, 4097 [array] ; 14: la $a0, array
[0040002c] 3c011001 lui $1, 4097 [length] ; 15: la $t0, length
[00400030] 34280005 ori $8, $1, 5 [length]
[00400034] 81050000 lb $5, 0($8) ; 16: lb $a1, ($t0)
[00400038] 23bdfffc addi $29, $29, -4 ; 18: addi $sp, $sp, -4
[0040003c] afbf0000 sw $31, 0($29) ; 19: sw $ra, 0($sp)
[00400040] 0c000000 jal 0x00400078 [mean] ; 21: jal mean
[00400044] 8fbf0000 lw $31, 0($29) ; 23: lw $ra, 0($sp)
[00400048] 23bd0004 addi $29, $29, 4 ; 24: addi $sp, $sp, 4
[0040004c] 2841000a slti $1, $2, 10 ; 26: blt $v0, 10, else
[00400050] 14200006 bne $1, $0, 24 [else-0x00400050]
[00400054] 3c011001 lui $1, 4097 [stringa] ; 27: la $a0, stringa
[00400058] 34240006 ori $4, $1, 6 [stringa]
[0040005c] 34020004 ori $2, $0, 4 ; 28: li $v0, 4 #system
call per print string
[00400060] 0000000c syscall ; 29: syscall
[00400064] 03e00008 jr $31 ; 30: jr $ra
[00400068] 00022021 addu $4, $0, $2 ; 33: move $a0, $v0
[0040006c] 34020001 ori $2, $0, 1 ; 34: li $v0, 1 #system
  
```