

Esercizio 1

Considerare il programma riportato nel file “esercizio1.s” ed elencare quali sono i commenti, le etichette, le istruzioni e le direttive di cui si compone. Effettuare quindi l’assemblaggio e il linking del programma con il comando:

```
aarch64-linux-gnu-gcc -o esercizio1 -static esercizio1.s
```

Eeguire poi il programma con il seguente comando. Cosa si ottiene in output?

```
qemu-aarch64 esercizio1
```

N.B.: I comandi su riportati si riferiscono al caso in cui si utilizza l’emulatore suggerito a lezione (già preinstallato nella macchina virtuale fornita).

Esercizio 2

Considerare il seguente programma riportato nel file “esercizio2_parte1.s” in cui vengono dichiarate alcune variabili.

```
.global main

.data
list: .word 0x00112233, 0x44556677,0x8899aabb,0xccddeeff
bool: .byte 0x99
num: .word 0x10203040
```

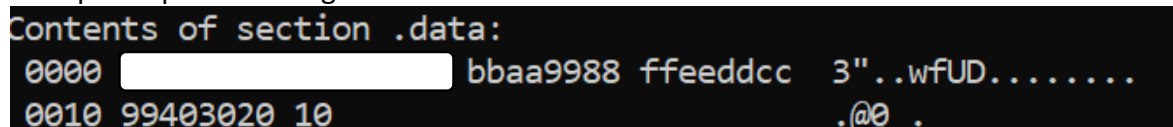
a. Utilizzando i comandi:

```
aarch64-linux-gnu-gcc -c esercizio2_parte1.s
aarch64-linux-gnu-objdump -s esercizio2_parte1.o
```

possiamo ispezionare la memoria.

N.B.: I comandi su riportati si riferiscono al caso in cui si utilizza l’emulatore suggerito a lezione (già preinstallato nella macchina virtuale fornita).

L’output è qualcosa del genere:



Indicare il contenuto della parte coperta in bianco. Specificare inoltre, a quali variabili e a quanti byte corrisponde tale parte.

- b. Perché nella sezione .data troviamo bbaa9988 invece che 8899aabb?
- c. Quale sarebbe l’output dei comandi precedenti se usassimo la codifica **big endian**?
- d. Verificare come varia il contenuto della sezione .data modificando il programma come segue (ovvero aggiungendo .align 2 tra le variabili bool e num).

```
.global main
```

```

.data
list: .word 0x00112233, 0x44556677,0x8899aabb,0xccddeeff
bool: .byte 0x99
.align 2
num: .word 0x10203040

```

Notiamo che ora la memoria risulta allineata. Quanti zero byte sono stati aggiunti?

- e. Consideriamo ora il seguente programma, riportato nel file “esercizio2_parte2.s”.

```

.global main

.data
list: .word 0x00112233,0x44556677,0x8899aabb,0xccddeeff
bool: .byte 0x99,0x88,0x77,0x66
.align 2
num: .word 0x10203040

```

Ispezioniamo la memoria occupata dalla sezione `.data` utilizzando i comandi:

```

aarch64-linux-gnu-gcc -c esercizio2_parte2.s
aarch64-linux-gnu-objdump -s esercizio2_parte2.o

```

Notiamo che la memoria risulta allineata e anche togliendo la direttiva `.align` e ispezionando nuovamente la memoria con i precedenti comandi, la memoria risulta comunque allineata. In altre parole, l’uso della direttiva `.align` non ha alcun effetto. Come mai?

Esercizio 3

Considerare il seguente programma riportato nel file “esercizio3.s”, in cui vengono dichiarate alcune variabili.

```

.global main

.data
a: .byte 0, 1, 2
b: .hword 15
c: .byte 'A'
d: .word 5, 2
e: .quad 9, 4
f: .asciz "Ciao"

```

- Arricchire il programma delle direttive `.align` che si ritengono necessarie facendo in modo di limitare il più possibile il numero di byte 0 aggiunti.
- Analizzare l’allineamento della memoria con i comandi:

```

aarch64-linux-gnu-gcc -c esercizio3.s
aarch64-linux-gnu-objdump -s esercizio3.o

```

N.B.: I comandi su riportati si riferiscono al caso in cui si utilizza l’emulatore suggerito a lezione (già preinstallato nella macchina virtuale fornita).

Quanti byte sono stati aggiunti?

Quanti ne verrebbero aggiunti se, ogni volta che si passa da un tipo piccolo a uno più grande, si usassero soltanto direttive `.align 3`?

- c. Riordinare le direttive in modo tale che non sia necessario aggiungere alcun `.align`.

Esercizio 4

Considerare il seguente programma riportato nel file “esercizio4.s” in cui vengono dichiarate alcune variabili.

```
.global main

.bss
n: .word 0
m: .word 0

.data
x: .byte 'a'
.skip 2
y: .byte 1
.skip 2
arr: .byte 'A', 'B', 'C'
```

- a. Indicare il contenuto della memoria **senza utilizzare il terminale**.
- b. Verificare il punto precedente ed analizzare l’allineamento della memoria con i comandi:

```
aarch64-linux-gnu-gcc -c esercizio4.s
aarch64-linux-gnu-objdump -s esercizio4.o
```

N.B.: I comandi su riportati si riferiscono al caso in cui si utilizza l’emulatore suggerito a lezione (già preinstallato nella macchina virtuale fornita).

- c. Utilizzando la sola direttiva `.skip`, modificare il programma per ottenere:

```
0000 00610000 01000000 414243
```

Determinare ora la dimensione del file `esercizio3.o` con il comando:

```
ls -l esercizio4.o
```

Si noti che `ls -l` dà in output il cosiddetto “long list format” ovvero un formato di output più dettagliato rispetto al semplice `ls`. Tale output contiene diverse informazioni riportate in 7 colonne: la prima colonna indica tipo di file e permessi di lettura/scrittura/esecuzione, la seconda il numero di collegamenti, la terza il nome del proprietario, la quarta il nome del proprietario del gruppo, la quinta la dimensione del file in byte (è questa l’informazione che ci interessa), la sesta data e ora dell’ultima modifica e la settima il nome del file (<https://linuxhint.com/what-does-ls-l-command-do-in-linux/>).

- f. A partire dalla soluzione del punto precedente, sostituire il primo `.skip` nella sezione `.data` con:

```
.skip 1024
```

Ripetere i comandi riportati nei punti b e c per assemblare e determinare nuovamente la dimensione del file `esercizio4.o`.

- g. Modificare ora il programma in questo modo:

```
.global main

.bss
n: .word 0
m: .word 0

.data
.skip 1*1024
x: .byte 'a'
.skip 2*1024
y: .byte 1
.skip 3*1024
arr: .byte 'A', 'B', 'C'
```

Ripetere i comandi riportati nei punti b e c per assemblare e determinare nuovamente la dimensione del file `esercizio4.o`.

Come cambia la dimensione del file `esercizio4.o`?

- h. Arricchire ora la sezione `.bss` come riportato di seguito:

```
.bss
n: .word 0
.skip 1
m: .word 0
```

- 1) Ripetere i comandi riportati nei punti b e c per assemblare e determinare nuovamente la dimensione del file `esercizio4.o`.
- 2) Ritorniamo nella sezione `.bss` e modifichiamola in modo da fare uno skip di 2048 (ovvero come segue):

```
.bss
n: .word 0
.skip 2048
m: .word 0
```

Ripetere i comandi riportati nei punti b e c per assemblare e determinare nuovamente la dimensione del file `esercizio4.o`.

Come cambia la dimensione del file `esercizio4.o`?

Esercizio 5

La sequenza di bit 10010110 in complemento a due, a quale intero in base 10 corrisponde? Per svolgere l'esercizio, effettuare prima il complemento a 1 della sequenza di bit e poi sommare 1.

A tale scopo, ricordiamo che per sommare due numeri binari bisogna incolonnarli e che 1 corrisponde a 00000001. Inoltre, ricordiamo che:

$0+0=0$

$$1+0=1$$

$$0+1=1$$

$1+1=0$ con riporto di 1 alla colonna immediatamente a sinistra

Esercizio 6

La sequenza di bit 1101 0101 in complemento a due, a quale numero intero in base 10 corrisponde?

Questa volta, per svolgere l'esercizio, sottrarre prima 1 alla sequenza di bit e poi effettuare il complemento a 1.

A tale scopo, ricordiamo che per sottrarre due numeri binari bisogna incolonnarli e che 1 corrisponde a 00000001. Inoltre, ricordiamo che:

$$0-0=0$$

$$1-0=1$$

$$0-1=1 \text{ prestando 1 dalla colonna a sinistra}$$

$$1-1=0$$

Esercizio 7

Avendo **8 bit a disposizione** e dato il numero intero -35 in base 10, calcolare la rappresentazione in complemento a 2.

Esercizio 8

Si converta in complemento a due, utilizzando il minor numero di bit possibile, il numero -38 in base 10.

Esercizio 9

Convertire la stringa costituita dal proprio nome seguito da uno spazio, dal proprio cognome e dal carattere NUL nel formato ASCII binario, esadecimale e decimale.

Suggerimento: per svolgere questo esercizio è sufficiente fare riferimento alla tabella sulle slide del corso sullo standard ASCII (vedi "caratteri stampabili").

Esercizio 10

Convertire la seguente sequenza di bit in UTF-8 e determinare la frase da essa codificata:

01001111 01100111 01100111 01101001 00100000 01101110 01101111 01101110 00100000
11000011 10101000 00100000 01101101 01100001 01110010 01110100 01100101 01100100
11000011 10101100 00100001 00000000

Suggerimento: aiutatevi con questa tabella di conversione <https://www.utf8-chartable.de/unicode-utf8-table.pl?utf8=bin>