Esercizio 1

Considerare il programma riportato nel file "esercizio1.s". Nella sezione .data è stato dichiarato l'array ***n***, in cui ogni elemento è di tipo word e occupa dunque 4 byte. Sono inoltre presenti due macro che permettono di stampare a video il contenuto del registro R1 in formato esadecimale: in particolare, nel programma, quando si fa uso del registro W1 si utilizza la macro print_32 mentre, quando si fa uso del registro X1 si utilizza la macro print_64. Nel main il programma riporta degli esempi, ciascuno generalmente composto da:

- una istruzione ldr che carica in R1 uno o più elementi dell'array ***n***;
- una invocazione delle macro suddette per visualizzare il contenuto del registro R1.

Considerando un esempio per volta:

a) indicare se l'istruzione ldr effettuata è corretta;
b) indicare il tipo di indirizzamento utilizzato;
c) spiegare il significato dell'istruzione ldr;
d) decommentare soltanto le istruzioni dell'esempio in esame e utilizzare il seguente comando per assemblare il programma, indicare quindi eventuali errori segnalati dall'assemblatore dandone una spiegazione:
```
aarch64-linux-gnu-gcc -o esercizio1 -static esercizio1.s
```
e) utilizzare infine il seguente comando per eseguire il programma e verificare quali dati sono stati caricati nel registro X1 o W1:
```
qemu-aarch64 esercizio1
```

Esercizio 2

Considerare il programma riportato nel file "esercizio2.s". Nella sezione .data è stato dichiarato l'array ***n***, in cui ogni elemento è di tipo word e occupa dunque 4 byte. Sono inoltre presenti due macro che permettono di stampare a video un singolo elemento dell'array oppure tutti i suoi elementi in formato esadecimale. Nel main il programma riporta degli esempi, ciascuno generalmente composto da:

- una istruzione str che sostituisce uno o più elementi dell'array ***n*** con il contenuto del registro X3 che contiene il valore esadecimale ffff ffff ffff ffff;
- una invocazione della macro che stampa tutti gli elementi dell'array.

Considerando un esempio per volta:

a) indicare se l'istruzione str effettuata è corretta;
b) indicare il tipo di indirizzamento utilizzato;
c) spiegare il significato dell'istruzione str;
d) decommentare soltanto le istruzioni dell'esempio in esame e utilizzare il seguente comando per assemblare il programma, indicare quindi eventuali errori segnalati dall'assemblatore dandone una spiegazione:
```
aarch64-linux-gnu-gcc -o esercizio2 -static esercizio2.s
```
e) utilizzare infine il seguente comando per eseguire il programma e verificare come cambia l'array ***n***:
```
qemu-aarch64 esercizio2
```

Esercizio 3

Considerare il programma riportato nel file "esercizio3.s":

a) Aggiungere delle opportune istruzioni ldr e str (utilizzando uno a scelta tra, post o pre indexed immediate offset) per impostare a 0 (utilizzando il registro W0) gli elementi che si trovano in posizioni pari nell'array ***n***. Assemblare ed eseguire poi il programma con i seguenti comandi per verificarne il corretto funzionamento.

```
aarch64-linux-gnu-gcc -o esercizio3 -static esercizio3.s
qemu-aarch64 esercizio3
```

b) Si noti che non è possibile utilizzare delle istruzioni **`stp`** per svolgere il punto a. Per quale ragione?

Esercizio 4

Considerare il programma riportato nel file "esercizio4.s":

a) Aggiungere delle opportune istruzioni ldr e str (utilizzando uno a scelta tra, post oppure pre indexed immediate offset) per impostare al valore *x* (dichiarato nella sezione .data) tutti gli elementi nell'array ***n***. Assemblare ed eseguire poi il programma per verificarne il corretto funzionamento.

```
aarch64-linux-gnu-gcc -o esercizio3 -static esercizio3.s
qemu-aarch64 esercizio3
```

b) Risolvere il punto a) utilizzando ora delle istruzioni **`stp`** con pre-indexed immediate offset.

c) Modificare il programma in modo tale da impostare al valore dell'elemento in posizione 1 (ovvero al valore 13) tutti gli elementi nell'array ***n*** utilizzando delle istruzioni **`stp`** con post-indexed immediate offset.

Esercizio 5

Considerare il programma riportato nel file "esercizio5.s". Tale programma è una variante del programma "load_store_single_register.s" con register offset visto a lezione in cui sono state modificate le istruzioni add. In particolare, `add x3, x3, #4` è stato ovunque sostituito con `add x3, x3, #1` e dunque il contenuto del registro x3 che inizialmente è 0 viene ogni volta incrementato di 1.

Modificare opportunamente le istruzioni ldr usando register offset in modo che tale programma calcoli la somma degli elementi dell'array ***n***, ottenendo così lo stesso comportamento del programma originale visto a lezione.

Esercizio 6

Considerare il programma riportato nel file "esercizio6.s". Tale programma riprende il programma "jump.s" visto a lezione. In particolare, nella funzione main viene memorizzato nel registro 0 il valore 0 e nel registro 1 il valore 1. Successivamente tramite l'istruzione `tst` si effettua l'AND bit a bit tra il contenuto del registro X0 e il contenuto del registro X1, ovvero tra 0 e 1 e vengono impostati in base al risultato ottenuto i flag N e Z. In base al valore del flag Z viene effettuato un salto condizionale, che fa sì che il programma stampi `true` oppure `false`: `true` se entra nel `true_case`, ovvero Z è impostato perché il risultato dell'operazione effettuata è 0; `false`, se entra nel `false_case` ovvero Z non viene impostato perché il risultato dell'operazione effettuata non è 0.

a) Assemblando ed eseguendo il programma con i due comandi riportati di seguito osserviamo che in output si ottiene `true`. Per quale motivo?

```
aarch64-linux-gnu-gcc -o esercizio6 -static esercizio6.s
qemu-aarch64 esercizio6
```

b) Modificare ora il programma sostituendo a `tst x0, x1` con `tst x0, x0` che effettua l'AND tra 0 e 0. Rieseguire i comandi riportati al punto 1 e spiegare il motivo del risultato ottenuto.
c) Modificare ora il programma sostituendo a `tst x0, x0` con `tst x1, x1` che effettua l'AND tra 1 e 1. Rieseguire i comandi riportati al punto 1 e spiegare il motivo del risultato ottenuto.
d) Modificare ora il programma sostituendo a `tst x1, x1` con `tst x1, x0` che effettua l'AND tra 1 e 0. Rieseguire i comandi riportati al punto 1 e spiegare il motivo del risultato ottenuto.