Spiegazione del programma in Assembly x86

Liam Ferretti

Sezione `.data`

```
section .data
        msg1    db "Inserisci il primo numero (max 10 cifre): ",
           0
        len1    equ $ - msg1
        msg2    db "Inserisci il secondo numero (max 10 cifre): "
           , 0
        len2    equ $ - msg2
        msgRes  db "La somma e': ", 0
        lenRes  equ $ - msgRes
        newline db 0Ah
```

Questa sezione contiene i dati costanti del programma:

- Le stringhe da stampare sullo schermo.
- Le variabili `len1`, `len2`, `lenRes` che contengono la lunghezza delle rispettive stringhe, calcolata come `$ - msgX`.
- `newline` è il carattere di nuova linea (codice ASCII 0Ah).
- lo zero dopo i db è il null terminator e serve a indicare la fine della stringa.

Sezione `.bss`

```
section .bss
        buffer resd 10
        n1 resd 1
        n2 resd 1
```

La sezione `.bss` serve per riservare spazio in memoria per dati variabili:

- `buffer` sarà usato per la conversione del risultato in ASCII.
- `n1` e `n2` conterranno i numeri interi letti da input.

Il direttivo `resd` riserva "doublewords" (4 byte ciascuna), il `resw` riserva "words" (2 byte ciascuna), il `resb` riserva "byte" (1 byte ciascuna), il `resq` riserva "quadwords" ovvero 8 byte.

Sezione `.text` e inizio del programma

```
section .text
global _start

start:
```

Qui inizia il codice eseguibile. L'etichetta `_start` indica il punto d'ingresso del programma (usato da Linux per sapere da dove iniziare l'esecuzione).

Stampa del primo messaggio e lettura del primo numero

```
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, len1
int 0x80
```

Queste istruzioni invocano la **system call write** (numero 4) per stampare `msg1` sullo schermo. I registri significano:

- `eax = 4` → numero della syscall `write`.
- `ebx = 1` → file descriptor dello standard output.
- `ecx = msg1` → indirizzo del messaggio.
- `edx = len1` → lunghezza della stringa.

```
mov eax, 3
mov ebx, 0
mov ecx, n1
mov edx, 10
int 0x80
```

Questa volta la syscall è `read`:

- `eax = 3` → numero della syscall `read`.
- `ebx = 0` → file descriptor dello standard input.
- `ecx = n1` → indirizzo su cui verranno salvati i dati.
- `edx = 10` → lunghezza dell'input.

Conversione del primo numero da ASCII a intero

```
1         xor eax, eax
2         xor ebx, ebx
3         mov esi, n1
4 .loop1:
5         mov bl, [esi]
6         cmp bl, 0x0A
7         je .done1
8         sub bl, '0'
9         imul eax, eax, 10
10        add eax, ebx
11        inc esi
12        jmp .loop1
13        .done1:
14        mov [n1], eax
```

Questo ciclo:

1. Le prime due righe servono a resettare i valori in eax e ebx, mettendo in entrambi 0.
2. muoviamo in esi l'indirizzo in memoria di n1, rendonolo quindi un puntatore che scorre i byte dell'input.
3. .loop, segno l'inizio del ciclo.
4. `mov bl, [esi]`, carica in bl (gli ultimi 8 bit di ebx) il byte puntato da esi, infatti con [ ] si rappresenta il valore dell'indice in memoria, e contiene un carattere ASCII.
5. `cmp bl, 10` confronta il valore di bl con 10, ovvero 0x0A in hex, che corrisponde al carattere newline (\n).
6. Ora `je .done1` nel caso in cui `cmp bl, 10` abbia restituito che bl è uguale a 10, allora salta a `.done1`
7. Sottraggo da bl il carattere 0, `sub bl, '0'` dato che 0 in hex è 0x30, e i numeri da 0 a 9 sono rappresentati da 0x30 a 0x39, si ottiene quindi il numero intero.
8. `imul eax, eax, 10` moltiplica eax per 10, e salva il risultato in eax, per spostare in avanti le cifre, per poter sommare la cifra delle unità
9. `add eax, ebx` somma la cifra corrente, in bl, su eax, ma essendo una operazione a 32 bit, bisogna usare ebx.
10. `inc esi` incremento il puntatore al byte successivo.
11. `jmp .loop1` torno al punto 3.
12. `.done1` etichetta di uscita dal ciclo
13. `mov [n1], eax`, scrive il valore intero del risultato in memoria all'indirizzo n1

Lettura e conversione del secondo numero

```
        mov eax, 4
        mov ebx, 1
        mov ecx, msg2
        mov edx, len2
        int 0x80

        mov eax, 3
        mov ebx, 0
        mov ecx, n2
        mov edx, 10
        int 0x80

        xor eax, eax
        xor ebx, ebx
        mov esi, n2
.loop2:
        mov bl, [esi]
        cmp bl, 0x0A
        je .done2
        sub bl, '0'
        imul eax, eax, 10
        add eax, ebx
        inc esi
        jmp .loop2
.done2:
```

È lo stesso procedimento del primo numero: si legge la stringa da tastiera, poi si converte in valore numerico.

Somma dei due numeri

```
add eax, [n1]
```

Dopo la conversione del secondo numero, `eax` lo contiene. Qui si aggiunge `n1` (che contiene il primo numero). Il risultato rimane in `eax`.

Conversione del risultato in ASCII

```
        mov edi, buffer + 10
        mov ecx, 10
.conv_loop:
        dec edi
        xor edx, edx
        div ecx
        add dl, '0'
        mov [edi], dl
        test eax, eax
        jnz .conv_loop
```

Questo ciclo esegue la conversione da numero binario ad ASCII:

1. `mov edi, buffer + 10`, metti nel registro edi, l'indirizzo alla fine del buffer, ovvero 10 byte.
2. `mov ecx, 10`, carico in ecx il valore 10, per eseguire la divisione.
3. `.conv_loop:` etichetta per l'inizio del ciclo.
4. `dec edi` sposto il puntatore di edi indietro di un byte verso sinistra
5. azzero edx con xor
6. `div ecx` divide il valore a 64 bit in edx:eax per 10 (ecx), e il quoziente finisce in eax, il resto in edx
7. `add dl, '0'` converte la cifra in ASCII
8. `mov [edi], dl` scrive la cifra ASCII in memoria puntata da edi
9. `test eax, eax` verifica che il contenuto di eax sia 0 quindi abbiamo finito di dividere il numero.
10. `jnz .conv_loop` se eax è diverso da 0, quindi quando non abbiamo ancora diviso tutte le cifre.

Stampa del risultato e terminazione

```
mov eax, 4
mov ebx, 1
mov ecx, msgRes
mov edx, lenRes
int 0x80

mov eax, 4
mov ebx, 1
mov ecx, edi
mov edx, buffer + 10
sub edx, edi
int 0x80

mov eax, 4
mov ebx, 1
mov ecx, newline
mov edx, 1
int 0x80

mov eax, 1
xor ebx, ebx
int 0x80
```

Infine: 1. Stampa la stringa “La somma è:”. 2. Stampa il numero convertito. 3. Va a capo. 4. Termina il programma con la syscall `exit (1)`.