Tutorial Assembly NASM

Corso: Tutoraggio Reti Logiche e Calcolatori - Unical 2024/2025

Indice

Introduzione

Questa è una guida veloce per svolgere passo passo un esercizio assembly

---

Struttura di un file NASM

Un file NASM è composto da **sezioni** ben definite. Analizziamo ogni parte:

1. Include delle librerie

```
%include "utils.nasm"
```

- Simile al `#include` in C
- Ci dà accesso a funzioni utili come `printw`, `exit`, ecc.
- **utils.nasm** contiene macro per stampare valori e gestire I/O

2. Sezione DATA

```
section .data
    V dd 3,-1,-2,6,-12,5    ; vettore di double word (32 bit)
    W dw -5,12,6,2,1,-3     ; vettore di word (16 bit)
    n equ ($ -W)/2          ; calcola automaticamente la lunghezza
```

Tipi di dato:

- `db` = Define Byte (8 bit = 1 byte)

- `dw` = Define Word (16 bit = 2 byte)
- `dd` = Define Double word (32 bit = 4 byte)
- `equ` = Equivale a una costante (come `#define` in C)

Calcolo automatico della lunghezza:

- `$` = indirizzo corrente in memoria
- `($ - W)/2` = (indirizzo finale - indirizzo iniziale) / dimensione elemento
- Divide per 2 perché ogni word è 2 byte

3. Sezione TEXT (codice principale)

```
section .text
    global _start      ; punto di ingresso del programma
    extern proc        ; dichiara che proc è definita in un altro file

_start:
    push n             ; mette n sullo stack (ultimo parametro)
    push W             ; mette indirizzo di W sullo stack
    push V             ; mette indirizzo di V sullo stack (primo parametro)

    call proc          ; chiama la procedura

    printw ax          ; stampa il risultato (contenuto in ax)
    exit 0             ; termina il programma
```

Ordine dei parametri: I parametri vengono messi sullo stack in ordine **inverso** (da destra a sinistra).

Analisi dell'esercizio completo

Analizziamo passo passo l'esercizio che conta le posizioni dove `V[p] = -W[p']`:

File principale (esercizio.nasm)

```
%include "utils.nasm"

section .data
    V dd 3,-1,-2,6,-12,5
    W dw -5,12,6,2,1,-3
    n equ ($ -W)/2

section .text
    global _start
    extern proc

_start:
    push n                ; dimensione n
    push W                ; indirizzo vettore W
    push V                ; indirizzo vettore V

    call proc

    printw ax             ; stampa risultato
    exit 0
```

File procedura (esercizio_proc.nasm)

1. Setup degli offset

```
section .data
    V equ 8      ; offset di V nello stack
    W equ 12     ; offset di W nello stack
    n equ 16     ; offset di n nello stack
```

Perché questi numeri?

```
Stack dopo call proc:
[ebp+16] = n        ← terzo parametro
[ebp+12] = W        ← secondo parametro
[ebp+8]  = V        ← primo parametro
[ebp+4]  = indirizzo di ritorno
[ebp+0]  = vecchio ebp
```

2. Inizializzazione della procedura

```
proc:
    push ebp            ; salva il frame pointer precedente
    mov ebp, esp        ; imposta nuovo frame pointer

    mov esi, [ebp + V] ; esi = indirizzo del vettore V
    mov edi, [ebp + W] ; edi = indirizzo del vettore W
    mov ecx, [ebp + n] ; ecx = dimensione n

    xor edx, edx        ; edx = 0 (indice del ciclo)
    xor eax, eax        ; eax = 0 (contatore risultato)
```

Registri utilizzati:

- `esi` = puntatore al vettore V
- `edi` = puntatore al vettore W
- `ecx` = dimensione n
- `edx` = indice corrente (p)
- `eax` = contatore risultato
- `ebx` = variabile temporanea

3. Il ciclo principale

```
.loop:
    cmp edx, ecx       ; confronta indice con n
    jge .end           ; se indice >= n, vai alla fine

    ; Calcolo posizione complementare p' = n - p - 1
    mov ebx, ecx       ; ebx = n
    sub ebx, edx       ; ebx = n - p
    dec ebx            ; ebx = n - p - 1 = p'

    ; Carica W[p'] e lo nega
    movsx ebx, word [edi + ebx*2]  ; carica W[p'] con estensione segno
    neg ebx                        ; ebx = -W[p']

    ; Confronta -W[p'] con V[p]
    cmp ebx, [esi + edx*4]         ; confronta -W[p'] con V[edx]
    jne .next                      ; se diversi, vai al prossimo

    inc eax            ; incrementa contatore

.next:
    inc edx            ; incrementa indice
    jmp .loop          ; torna all'inizio del ciclo

.end:
    mov esp, ebp       ; ripristina stack pointer
    pop ebp            ; ripristina frame pointer
    ret 12             ; ritorna e pulisce 12 byte dallo stack
```

Esempio di esecuzione

Dati:

```
V = [3, -1, -2,  6, -12,  5]   (posizioni 0,1,2,3,4,5)
W = [-5, 12,  6,  2,   1, -3]   (posizioni 0,1,2,3,4,5)
n = 6
```

Calcoli:

```
p=0 → p'=5: V[0]=3,   -W[5]=-(-3)=3   ✓ UGUALE
p=1 → p'=4: V[1]=-1,  -W[4]=-(1)=-1   ✓ UGUALE
p=2 → p'=3: V[2]=-2,  -W[3]=-(2)=-2   ✓ UGUALE
p=3 → p'=2: V[3]=6,   -W[2]=-(6)=-6   ✗ DIVERSO
p=4 → p'=1: V[4]=-12, -W[1]=-(12)=-12 ✓ UGUALE
p=5 → p'=0: V[5]=5,   -W[0]=-(-5)=5   ✓ UGUALE

Risultato: 5 posizioni soddisfano la condizione
```

Concetti fondamentali

1. Indirizzamento in memoria

```
mov eax, [esi + edx*4]    ; V[edx] - moltiplicazione per 4 (double word)
mov bx, [edi + ebx*2]     ; W[ebx] - moltiplicazione per 2 (word)
```

2. Estensione di segno

```
movsx ebx, word [edi + ebx*2]  ; estende il segno da 16 a 32 bit
```

- Necessario quando si passa da word (16 bit) a double word (32 bit)
- Preserva il segno del numero

3. Operazioni aritmetiche

```
neg ebx         ; ebx = -ebx (complemento a 2)
inc eax         ; eax = eax + 1
dec ebx         ; ebx = ebx - 1
```

4. Controllo del flusso

```
cmp edx, ecx    ; confronta edx con ecx
jge .end        ; salta se edx >= ecx (jump if greater or equal)
jne .next       ; salta se edx != ecx (jump if not equal)
jmp .loop       ; salta incondizionato
```

5. Gestione dello stack

```
push ebp        ; mette ebp sullo stack
pop ebp         ; recupera ebp dallo stack
ret 12          ; ritorna e rimuove 12 byte dallo stack
```

Come compilare ed eseguire

1. **Compilazione:**

```
nasm -f elf32 esercizio.nasm -o esercizio.o
nasm -f elf32 esercizio_proc.nasm -o esercizio_proc.o
```

2. **Linking:**

```
ld -m elf_i386 -o esercizio esercizio.o esercizio_proc.o utils.o
```

3. **Esecuzione:**

```
./esercizio
```

Autore: Emanuele Vita **Corso:** Tutoraggio Reti Logiche e Calcolatori - Unical 2024/2025

Data: 3 Settembre 2025