ELABORATO
ARCHITETTURA DEGLI ELABORATORI

A.A. 2020/2021

Bonoldi Enrico | VR456133
Cappelletti Pietro |VR458771

Introduzione

Descrizione del progetto

Realizzazione di una parte di software dedicata al calcolo aritmetico in notazione RPN (reverse polish notation).

Specifica del progetto

Obiettivo

Scrittura di un sotto programma assembly che riceva in input due puntatori a stringhe, una di input (in RPN) e una di output, e ne calcoli il risultato della stringa di input con relativo output nella stringa di output.

Requisiti e Vincoli

- Gli operatori considerati sono i 4 fondamentali e codificati con i seguenti simboli:
  - + Addizione
  - * Moltiplicazione (non x)
  - - Sottrazione
  - / Divisione (non \)

- Un operando può essere composto da più cifre intere con segno (10, -327, 5670).
- Solo gli operandi negativi hanno il segno riportato esplicitamente in testa.
- Gli operandi hanno un valore massimo codificabile in 32 bit.
- Il risultato di una moltiplicazione o di una divisione può essere codificato al massimo in 32-bit.
- Il risultato di una divisione dà sempre risultati interi, quindi senza resto.
- Il dividendo di una divisione delle istanze utilizzate è sempre positivo, mentre il divisore può essere negativo.
- Tra ogni operatore e/o operando vi è uno spazio che li separa.
- L'ultimo operatore dell'espressione è seguito dal simbolo di fine stringa "\0".
- Le espressioni **NON** hanno limite di lunghezza.
- L'eseguibile generato si dovrà chiamare postfix- Non è consentito l'utilizzo di chiamate a funzioni descritte in altri linguaggi all'interno del codice Assembly.
- Se le stringhe inserite non sono valide (contengono simboli che non sono operatori o numeri) il programma deve restituire la stringa scritta esattamente nel seguente modo: "Invalid"

Realizzazione

Files

il file assembly (postfix.s) è unico e non include altre funzionalità esterne.

Costanti

Caratteri relativi agli operatori (ascii)
char_add
char_sub
char_mul
char_div

Caratteri speciali (ascii)
char_spazio
char_fine

Flag operazioni (byte)
flag_add (0x1)
flag_sub (0x2)
flag_mul (0x4)
flag_div (0x8)

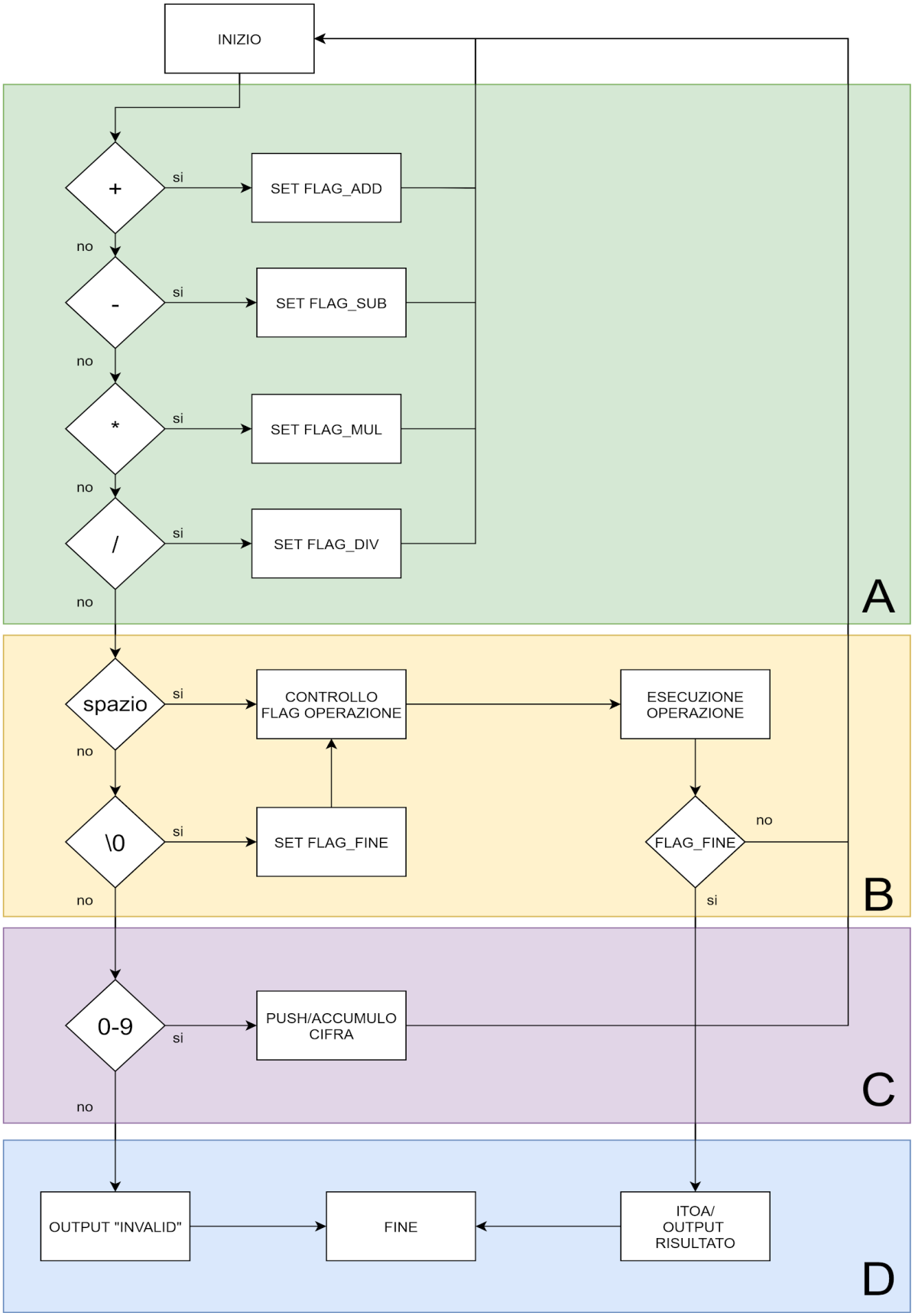
Altri flag (byte)
flag_fine (0x1)
flag_fine_operazione (0x2)
flag_neg (0x3)

Registri

In generale l'uso di questi registri è stato organizzato in questo modo:

- **ESI** mantiene una copia di **ESP** al momento della chiamata a funzione (utilizzato nel *blocco D* nella fase di pulizia)
- **EAX** punta al carattere in lettura
- **CL** è dedicato a **Flag operazioni**
- **CH** è dedicato ad **Altri flag**

Flusso del programma



il programma è stato suddiviso in 3 blocchi operativi in base al tipo di carattere che devono elaborare; l'ultimo blocco si occupa della scrittura dell'output.

Blocco A (segni di operazione)

Se viene riconosciuto un carattere rappresentante un segno di una operazione viene impostato **CL** registro al flag concorde all'operazione.

Blocco B (spazio e \0)

Se viene riconosciuto il carattere nullo (fine stringa) si imposta **CH** a *flag_fine* quindi si prosegue con l'esecuzione come se fosse un carattere di spazio

Se viene riconosciuto il carattere di spazio si controlla **CL** (flag di operazione) e si effettua l'operazione relativa.
Quindi se **CH** è *flag_fine* si salta al blocco D(output risultato) oppure di torna ad INIZIO.

Blocco C (0-9)

Se viene riconosciuto il carattere rappresentate una cifra si accumula o si inserisce un nuovo numero.

Nel primo caso si esegue la pop del numero precedente e lo si moltiplica per 10 quindi lo somma(o sottrae in base a **CH** (*flag_neg*)) con la cifra letta ed infine si esegue una push del nuovo numero.

Nel secondo caso si inserisce la nuova cifra che è:

- **negativa** nel caso **CL** sia impostato a *flag_sub.*
  push del nuovo numero negativo, set di **CH** a *flag_neg* e clear di **CL.**
- altrimenti **positiva**
  push del nuovo numero

Blocco D (output e ret)

Nel caso OUTPUT "INVALID" si imposta la stringa di output a "Invalid", si ripristina **ESP** e si ritorna.
Altrimenti si procede alla lettura del risultato (pop del primo elemento dello stack), si ripristina **ESP**, si effettua **itoa** del risultato e scrittura del risultato nella stringa di output (con segno in caso di numero negativo) e si ritorna

Scelte progettuali

Il programma utilizza lo stack come elemento fondamentale per il calcolo cumulativo degli operandi e delle operazioni perciò abbiamo deciso di non utilizzare sotto programmi (call-ret).

Le ricorrenze di **Altri flag** sono stati inseriti nel codice senza un lettura della costante (hard coded).