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, uno di input (in RPN) e uno di output, calcoli il risultato della stringa di input attraverso le rispettive operazioni in RPN e ne inserisca il valore d'uscita 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 almeno 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.
- Nel caso in cui la stringa inserita non avesse un valore valido secondo lo standard RPN l'uscita deve essere descritta come "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)

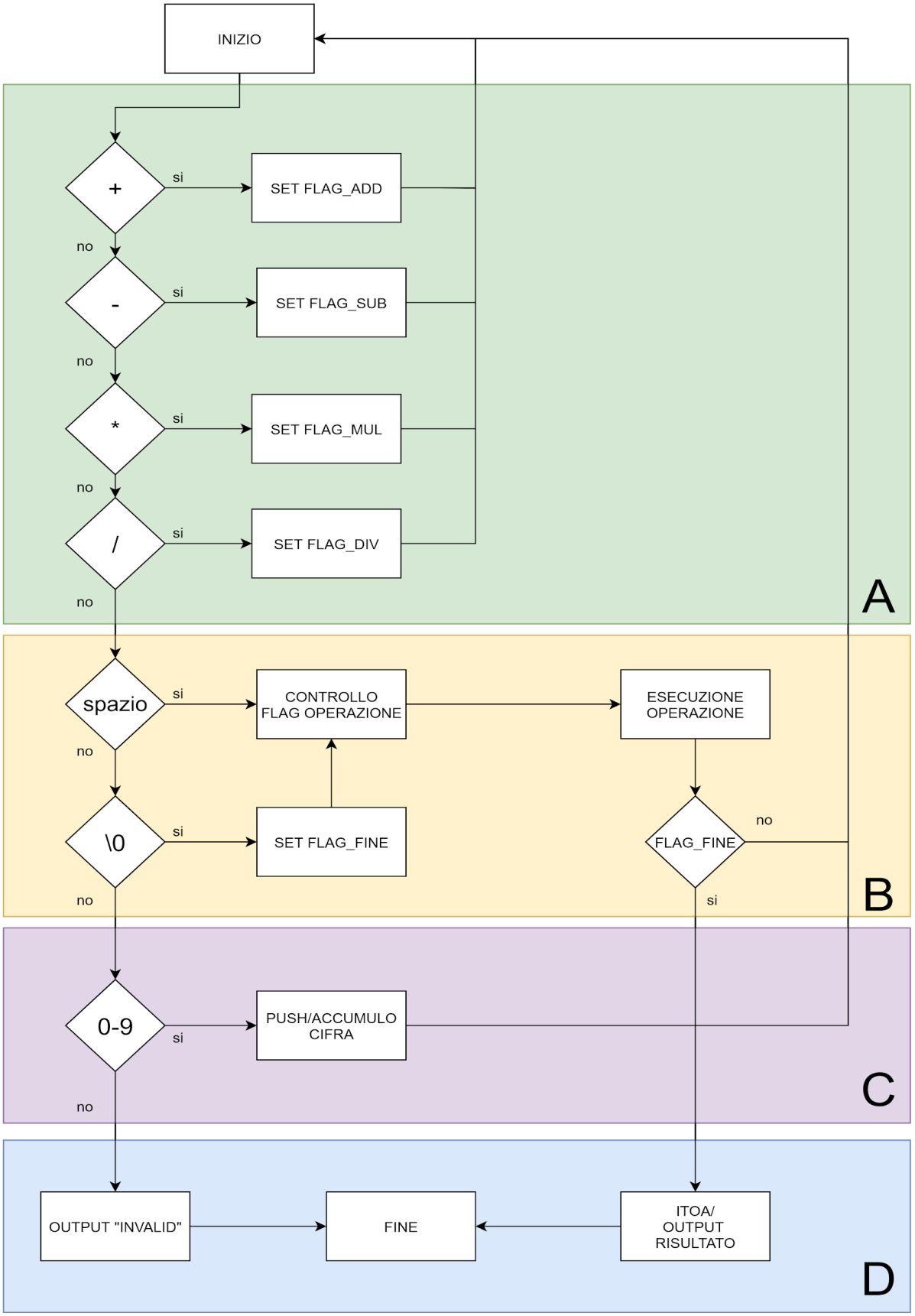
Altri flag (byte)
flag_fine (0x1)
flag_fine_operazione (0x2)
flag_neg (0x3)

Registri

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

- **ESI** - mantenere una copia di **ESP** al momento della chiamata a funzione in modo da permetterne poi nel blocco *D* la sua pulizia
- **EAX** - punta al carattere in lettura
- **CL -** dedicato ai **Flag operazioni** che descrivono l'operazione da compiere sui numeri ottenuti
- **CH** - dedicato ad **Altri flag** utili per gestire la stringa e le sue caratteristiche: inizio di un'operazione, fine della stringa, etc

Flusso del programma



Il programma è suddiviso in tre blocchi operativi diversificati dal tipo di carattere che il sistema deve elaborare. Il blocco finale *D* si occupa della scrittura a schermo del risultato e del suo salvataggio nel file di output.

Blocco A (segni di operazione)

Nel caso in cui il carattere riconosciuto indicasse un segno di operazione il programma imposterebbe in questo blocco il registro **CL** al flag corrispondente.

Blocco B (spazio e \0)

Nel caso in cui il carattere riconosciuto indicasse il carattere nullo (fine striga) il programma, dopo aver impostato il valore del registro **CH** a *flag_fine*, continuerebbe la sua esecuzione simulando il comportamento nel caso il carattere inserito fosse uno spazio.

Nel caso in cui il carattere fosse quello di spazio si controllerebbe la presenza di un operazione, descritta dai **Flag operazioni**, nel registro **CL** in modo da eseguirla nel caso fosse presente.

Blocco C (0-9)

Nel caso in cui il carattere riconosciuto fosse una normale cifra numerica (0-9) si presenterebbero due possibilità:

1) La cifra inserita rappresenta la continuazione del numero attualmente creato.
   In questo caso il comportamento del programma sarebbe quello di eseguire un'istruzione di pop recuperando dallo stack il numero precedentemente inserito, lo si moltiplica per 10 per poi sommarlo all'attuale cifra inserita nel caso in cui il numero fosse positivo, cioè nel caso in cui il registro **CH** non contenesse il *flag_neg*, o sottrarlo nel caso contrario.
2) La cifra inserita è la prima di un nuovo numero. Questa operazione presenta due possibilità:
   a) Numero negativo nel caso in cui il registro **CL** sia impostato a *flag_sub*:
      Viene effettuata una push (salvataggio nello stack) del numero e viene impostato il valore di **CH** a *flag_neg* in modo da proseguire l'inserimento del numero negativo. Viene inoltre azzerato il valore di **CL**.
   b) Numero positivo quindi viene effettuata solamente una push della cifra del nuovo numero nello stack.

Blocco D (output e ret)

Nel caso in cui la lettura della stringa avesse provocato un errore dovuto alla sua struttura, diversa da quella richiesta per lo standard RPN, si assegnerebbe il valore di “Invalid” alla stringa di output, come richiesto da consegna, per poi ripristinare il registro **ESP** al suo valore iniziale eseguendo infine l'operazione di RET per concludere l'esecuzione del programma.

Se invece l'operazione fosse andata a buon fine portando ad un risultato numerico si procede alla lettura del valore tramite l'esecuzione dell'istruzione pop per poi ripristinare **ESP** ed effettuare **itoa**, traduzione da numero a stringa, del numero.
Una volta effettuata la sua conversione si procede con la scrittura nella stringa di output con relativo segno in caso di numero negativo.

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 tramite la chiamata di istruzioni CALL-RET.

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