UNIVERSITA’ DEGLI STUDI DI VERONA

Elaborato Assembly

ANNO 2020/2021



Zenaro Stefano ***VR456736***

d’Alessandro Marco ***VR461145***

Farina Christian ***VR456150***

## *INDICE*

-***CENNI TEORICI*** 3

***-OBIETTIVO DEL PROGETTO***  4

-***METODI*** 4

-***VARIABILI*** 4

-***DIAGRAMMI DI FLUSSO*** 5, 6, 7, 8

-***PSEUDOCODICE DI POSTFIX***  9, 10, 11

## *CENNI TEORICI*

La notazione polacca inversa (reverse polish notation, RPN) è una notazione per la scrittura di espressioni aritmetiche in cui gli operatori binari, anziché utilizzare la tradizionale notazione infissa, usano quella postfissa; ad esempio, l’espressione 5 + 2 in RPN verrebbe scritta 5 2 +. La RPN è particolarmente utile perché non necessita dell’utilizzo di parentesi.

Si considerino ad esempio le due espressioni:

- 2 \* 5 + 1

- 2 \* (5 + 1)

Nel secondo caso le parentesi sono necessarie per indicare che l’addizione va eseguita prima della moltiplicazione.

In RPN questo non è necessario perché le due espressioni vengono scritte in maniera diversa: mentre la prima corrisponde a 2 5 \* 1 +, la seconda viene scritta come 2 5 1 + \*.

Un altro vantaggio della RPN è quello di essere facilmente implementabile utilizzando uno stack.

Per calcolare il valore di un’espressione, è sufficiente scandirla da sinistra verso destra: quando viene letto un numero lo si salva nello stack, quando viene letta un’operazione binaria si prelevano due numeri dallo stack, si esegue l’operazione tra tali numeri e si salva nuovamente il risultato nello stack.

Ad esempio, volendo valutare il valore dell’espressione 2 5 1 + \*, si procede nel seguente modo:

1. metto il valore 2 nello stack;

2. metto il valore 5 nello stack;

3. metto il valore 1 nello stack;

4. estraggo i primi due valori memorizzati in cima allo stack (5 e 1);

5. faccio la somma e salvo il risultato nello stack;

6. estraggo i primi due valori memorizzati in cima allo stack (2 e 6);

7. faccio la moltiplicazione e salvo il risultato;

8. A questo punto l’intera stringa è stata elaborata e nello stack è memorizzato il risultato finale.

## *OBIETTIVO*

Scrivere un programma in assembly che riesca a prendere in input una stringa contenente un’espressione ben formata in notazione polacca inversa (RPN) e che scriva in output il risultato ottenuto dalla precedente espressione. Si consideri inoltre che le operazioni ammesse sono addizione (+), sottrazione (-), divisione (/) e moltiplicazione (\*).

***VARIABILI***

**Invalid** (byte): stabilisce la validità dell’input

**numeric** (byte): flag che indica se si tratta di un numero

**numero** (long): variabile temporanea che memorizza ogni operando letto in input

**is\_negative** (byte): flag che indica se si tratta di un numero negativo

## *METODI*

Per sviluppare il programma è stato necessario utilizzare alcuni metodi. Qui di seguito sono riportati i loro nomi e il compito che essi svolgono:

***addizione.s:*** metodo che permette di svolgere l’operazione addizione

***divisione.s:*** metodo che ci permette di svolgere l’operazione divisione

***is\_operand.s:*** controlla se quello che è stato inserito è un operando

***is\_operator:*** controlla se quello che è stato inserito è un operatore

***is\_valid\_char.s:*** controlla se il carattere inserito è valido

***itoa.s:*** è in grado di convertire un numero n in una stringa di carattere e stamparla

***main.c:*** scritto in c già precedentemente fornito

***postfix.s:*** funzione principale che richiama tutte le sotto funzioni

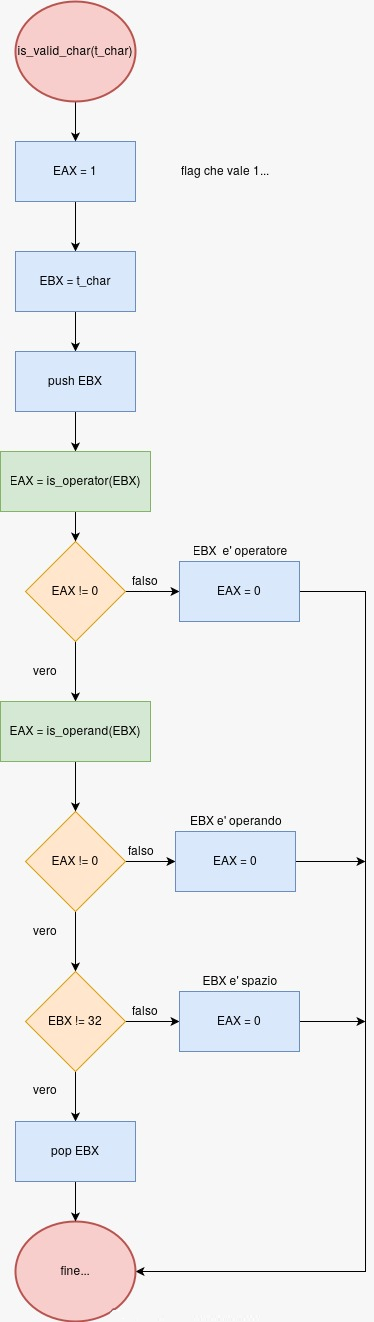
***prodotto.s:*** funzione adibita alla moltiplicazione dei membri

***sottrazione.s:*** funzione che permette la sottrazione dei membri

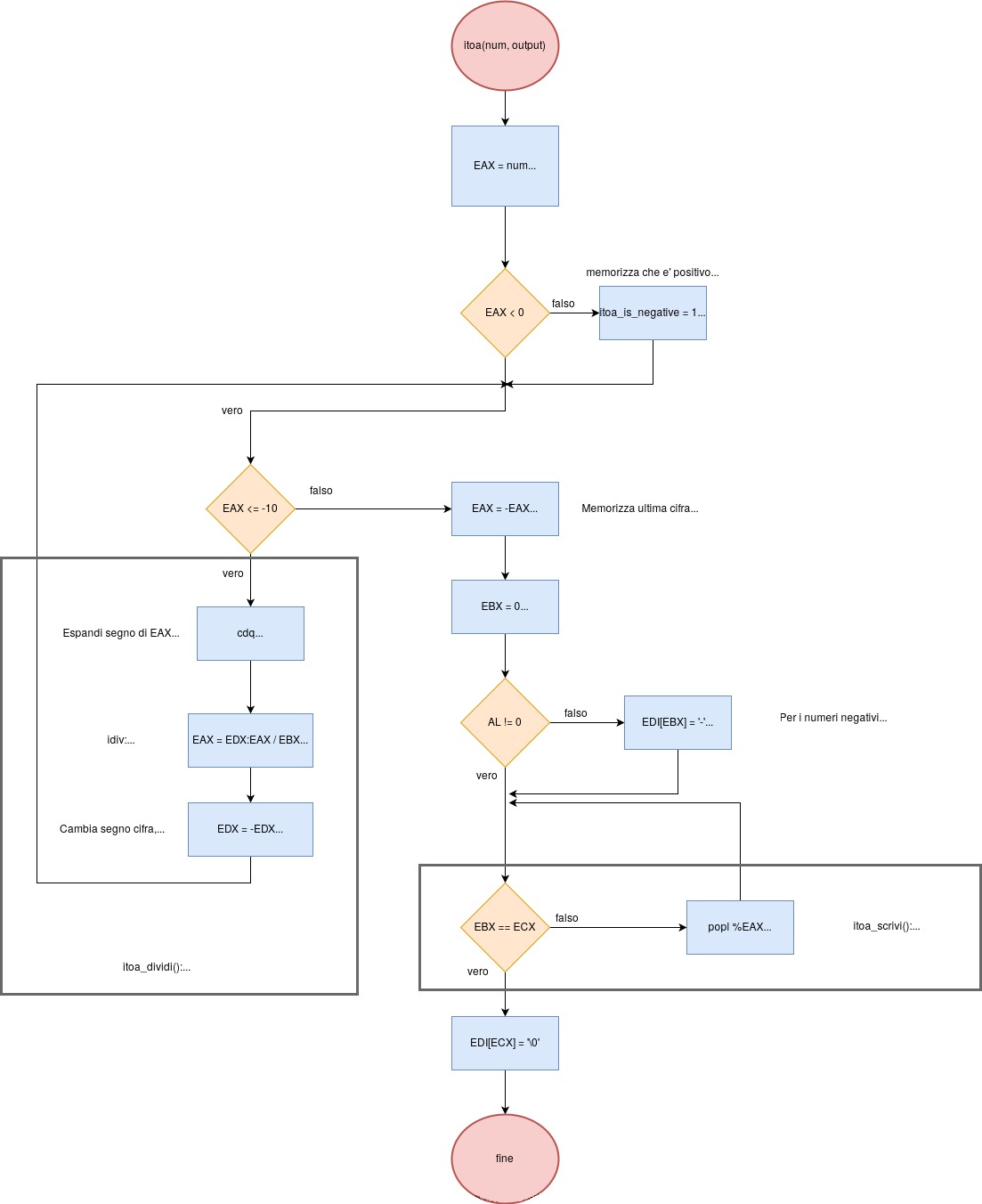
***write\_result.s:*** questa funzione permette la stampa del risultato dell’operazione con anche l’opzione “Invalid”

## *DIAGRAMMI DI FLUSSO*

# *Figura 1* Flowchart per l’operando *Figura 2* Flowchart per l’operatore

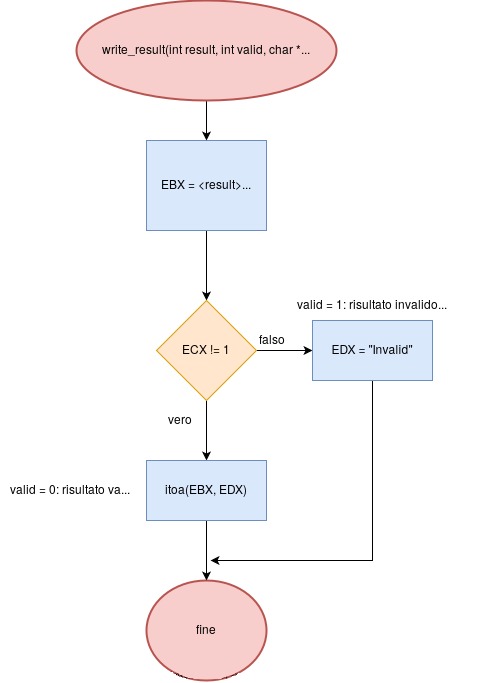


*Figura 3* Flowchart che verifica di validità carattere



*Figura 4* Flowchart itoa

***NOTA BENE:*** *nel metodo itoa() convertiamo i numeri positivi in negativi perché' il valore -2147483648 non può essere convertito in numero positivo mentre 2147483647 può essere convertito in numero negativo.*



*Figura 5* Flowchart write result

## *PSEUDOCODICE DI POSTFIX*

**ELABORAZIONE INPUT**

carattere=primo carattere del file //poi nel loop si incrementerà l’indice

If((carattere!= \0 *or* carattere!=\n) *and* invalid==false):proseguo

else: salto a SCRITTURA OUTPUT//l’input è terminato oppure c’è un carattere invalido

push carattere//salvo il carattere per farlo analizzare dalla funzione is\_valid\_char

chiamo is\_valid\_char//funzione spiegata dai diagrammi di flusso

if(invalido): invalid==true e salto a postfix\_incrementa\_indice

else: proseguo

//ora devo capire che tipo di carattere è

if(carattere è operatore):salto a postfix\_is\_operator//chiamata della funzione is\_operator

If(carattere è cifra): push carattere convertito// chiamata della funzione is\_operand, poi verranno controllate anche le cifre successive e moltiplicate via via per 10

if(carattere è segno di numero negativo):is\_negative=true

if (carattere è spazio): push numero salto a postfix\_incrementa\_indice// controllando is\_negativecaspisco se rendere il numero positivo o negativo

**postfix\_is\_operator**

if(non ci sono sufficienti elementi nella pila):salto a postfix\_not\_enough\_operands

else: chiamo la funzione dell’operazione corrispondente e poi salto a postfix\_incrementa\_indice

**postfix\_not\_enough\_operands**

invalid=true//dato che non ci sono abbastanza numeri nello stack, la formula non è ben formata

salto a postfix\_incrementa\_indice

**postfix\_incrementa\_indice**

incremento l’indice e salto all’inizio

**SCRITTURA OUTPUT**

//il risultato è nella pila

If(invalid==true): salto a postfix\_write\_invalid\_invchar//chiamerà write\_result che scriverà Invalid

else: salto a postfix\_write\_appearently\_valid\_result//non so ancora se è tutto corretto

**postfix\_write\_appearently\_valid\_result**// controlla se l’espressione è ben formata

if(troppi elementi nella pila): salto postfix\_rm\_too\_many\_els\_stack

else: chiamo write\_result//la formula è ben formata, quindi scrivo il risultato e concludo

**postfix\_rm\_too\_many\_els\_stack**

loop(se ci sono troppi elementi nello stack): rimuovo uno a uno gli elementi e dopo proseguo

**postfix\_write\_invalid\_too\_many\_els\_stack**

scrivo “Invalid” (chiamando write\_result) perché l’espressione non era ben formata

***NOTA BENE***

*Questo pseudo codice non è una copia esatta e fedele del reale programma dato che sono stati omessi alcuni controlli (es: numeri a più cifre) al fine di rendere più comprensibile la relazione. Le funzioni già spiegate dai diagrammi non sono state approfondite in questa sezione. L’etichetta “fine”, che si occupa di ripristinare i vari elementi dalla pila e eseguire la return, non è stata inserita poiché si assume che si salti a quell’etichetta ogni qualvolta si termini di scrivere il risultato finale.*