uniVR - Dipartimento Informatica

Elaborato Assembly Laboratorio Architettura degli Elaboratori

Notazione RPN
(*reverse polish notation*)

A.A. 2020/2021

Gruppo di lavoro:
Amos Lo Verde (vr456585)
Nicolò Piccoli (vr459373)
Simone Moratti (vr456083)

Indice

Descrizione progetto

La *RPN* è una notazione per la scrittura di espressioni aritmetiche in cui gli operatori binari usano la notazione postfissa, anziché quella tradizionale.
Si considerano per esempio le seguenti due espressioni:

Caso	**Notazione normale**	**Reverse Polish Notation**
1°	$2 * 5 + 1$	$2\ 5\ *\ 1\ +$
2°	$2 * (5 + 1)$	$2\ 5\ 1\ +\ *$

Nella **notazione normale**, nel secondo caso, le parentesi tonde sono necessarie per indicare che l'addizione va eseguita prima della moltiplicazione; al contrario in **RPN** non sono necessarie le parentesi perché le due espressioni vengono scritte in maniera diversa.

L'espressione viene letta da sinistra verso destra: si caricano sullo STACK i numeri letti, mentre le operazioni vengono effettuate prelevando i primi due valori dallo STACK e si memorizza nuovamente il risultato in memoria.

-Obiettivo:

Realizzare un programma in linguaggio Assembly, richiamato come funzione dal programma MAIN.C, che preleva dal vettore **input** nella HEAP una stringa in **notazione polacca inversa** (*Reverse Polish Notation*) e restituisce il risultato dell'operazione nel vettore **output** nello STACK.

Se la stringa di partenza non è valida, ossia contiene simboli che non sono operatori o numeri, allora il programma deve restituire all'interno del vettore **output** la stringa: *Invalid*.

Variabili

- **esp_value**: contiene il valore iniziale dell'**esp** alla chiamata della funzione, in modo che prima della **ret** si possa ripristinare il valore iniziale.
- **ebp_value**: contiene il valore iniziale dell'**ebp** alla chiamata della funzione, in modo che prima della **ret** si possa ripristinare il valore iniziale.
- **eax_value**: contiene il valore iniziale dell'**eax** alla chiamata della funzione, in modo che prima della **ret** si possa ripristinare il valore iniziale.
- **ebx_value**: contiene il valore iniziale dell'**ebx** alla chiamata della funzione, in modo che prima della **ret** si possa ripristinare il valore iniziale.
- **ecx_value**: contiene il valore iniziale dell'**ecx** alla chiamata della funzione, in modo che prima della **ret** si possa ripristinare il valore iniziale.
- **edx_value**: contiene il valore iniziale dell'**edx** alla chiamata della funzione, in modo che prima della **ret** si possa ripristinare il valore iniziale.
- **scala**: funge da costante contenendo il valore 10.

Funzioni

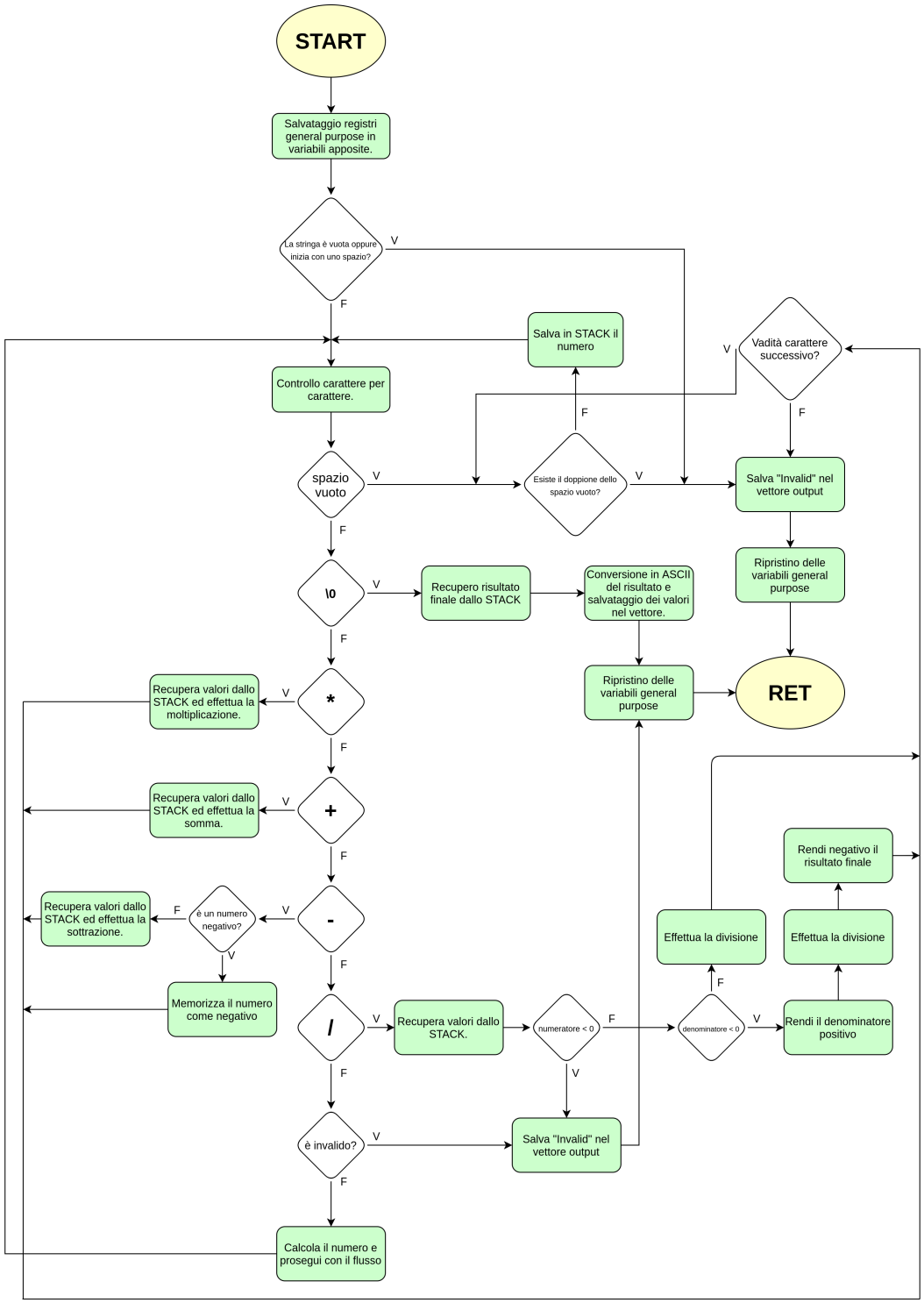
Il file **postfix.s** è la funzione Assembly richiamata dal programma C.
In essa vengono passati come parametri il puntatore alla prima cella del vettore **input** presente sulla HEAP e il puntatore alla prima cella del vettore **output** presente sullo STACK.
Vengono effettuate le operazioni per risolvere il calcolo fornito in **RPN** e inoltre sono presenti controlli sulla validità della stringa.

Il programma si conclude quando:

- Nel vettore **input** c'è un carattere invalido: a questo punto si interrompe il normale flusso d'esecuzione e si salta a una specifica etichetta, nella quale si inserisce carattere per carattere nel vettore **output** la stringa "*Invalid*" e si salta a un'altra etichetta specifica dove si ripristinano i valori iniziali dei registri, per poi tornare al file C con **ret**.

- Si arriva alla fine del vettore **input** (dunque è valida la stringa): in questo caso si interrompe il normale flusso d'esecuzione e si salta a una specifica etichetta, dove si preleva dallo STACK il risultato finale e le singole cifre vengono convertite da intero ad ASCII per poi caricarle, una alla volta, nel vettore **output**. Infine si ripristinano i valori iniziali dei registri e con **ret** si torna al file C.

Flowchart


START
Salvataggio registri general purpose in variabili apposite.
La stringa è vuota oppure inizia con uno spazio?
V
F
Salva in STACK il numero
Controllo carattere per carattere.
spazio vuoto
V
Esiste il doppione dello spazio vuoto?
V
F
Vadità carattere successivo?
V
F
Salva "Invalid" nel vettore output
Ripristino delle variabili general purpose
\0
V
Recupero risultato finale dallo STACK
Conversione in ASCII del risultato e salvataggio dei valori nel vettore.
F
Ripristino delle variabili general purpose
RET
*
V
Recupera valori dallo STACK ed effettua la moltiplicazione.
F
+
V
Recupera valori dallo STACK ed effettua la somma.
F
-
V
è un numero negativo?
F
Recupera valori dallo STACK ed effettua la sottrazione.
V
Memorizza il numero come negativo
F
/
V
Recupera valori dallo STACK.
numeratore < 0
F
V
denominatore < 0
V
Rendi il denominatore positivo
F
Effettua la divisione
Effettua la divisione
Rendi negativo il risultato finale
F
è invalido?
V
Salva "Invalid" nel vettore output
F
Calcola il numero e prosegui con il flusso

Scelte progettuali

1. I valori dei **registri general purpouse** e dei **registri puntatori** dello STACK sono stati salvati nelle apposite variabili (esp_value, ebp_value, eax_value, ebx_value, ecx_value, edx_value) all'inizio della funzione **postfix**.
   Si è evitato di caricarli sullo STACK per non dover effettuare ulteriori controlli sul corretto scaricamento, in caso di invalidità stringa, dei valori inseriti successivamente.

2. Si è creata una costante, chiamata **scala**, con l'obiettivo di moltiplicare per 10 il numero finale a ogni lettura di una cifra, esempio:
   $Numero = 321$, **eax** $= 0$ (possiede il numero finale), **ecx** possiede la singola cifra presa a ogni lettura.

   - Lettura 1:
     $ecx = 3$
     $eax = eax \cdot scala = 0 \cdot 10 = 0$
     $eax = eax + ecx = 3 + 0 = 3$
   - Lettura 2:
     $ecx = 2$
     $eax = eax \cdot scala = 3 \cdot 10 = 30$
     $eax = eax + ecx = 30 + 2 = 32$
   - Lettura 3:
     $ecx = 1$
     $eax = eax \cdot scala = 32 \cdot 10 = 320$
     $eax = eax + ecx = 320 + 1 = 321$

   In questo modo dentro **eax** si ha il numero effettivo da caricare sullo STACK.

3. Nel caso la stringa **input** abbia come primo o ultimo carattere uno spazio vuoto oppure nel mezzo ci siano più spazi vuoti consecutivi, allora la stringa è considerata non ben formata e di conseguenza si restituisce *Invalid* nel vettore **output**.

4. Nel caso la stringa **input** sia vuota allora viene considerata non ben formata e di conseguenza si restituisce *Invalid* nel vettore **output**.