RELAZIONE ELABORATO ASM ARCHITETTURA DEGLI ELABORATORI

A.A 2020 -2021

VR456441 Campostrini Elisa
VR456447 Marano Angelo
VR456907 Gennarelli Stefano

INDICE :

DESCRIZIONE DEL FILE ASSEMBLY

Nell'implementazione di una calcolatrice che usa la RPN notation (reverse polish notation) , ovvero una notazione usata per scrivere espressioni aritmetiche in cui gli operatori binari usano la notazione postfissa, contraria a quella tradizionale infissa. Questo programma prende in input una stringa contenente l'espressione aritmetica scritta in RPN e ne restituisce il risultato in output.

Per fare ciò è stata usata una varaibile :
EBP : salva il valore del registro ebp.

Sono state usate i seguenti label all'interno del file assembly:

main_loop : ciclo iniziale e principale che valuta se il carattere presente nel puntatore alla stringa , contenente l'espressione iniziale , è un numero o un segno o se è qualcos'altro.

torno: verificato che si tratta di un numero , qui puliamo i due registri eax e edx e li utilizziamo per convertire il carattere in numero e metterlo sullo stack se , una volta aumentato il puntatore, troviamo uno spazio, ovvero il numero è terminato . In caso contrario andiamo a **spazio**.

torno_negato: verificato che dopo il segno meno non c'è uno spazio o il carattere di fine stringa , verifichiamo se si tratta di un numero e se lo è , lo convertiamo da carattere a numero e se , dopo aver incrementato il puntatore alla stringa d'ingresso , troviamo uno spazio , neghiamo il numero e lo mettiamo sullo stack , in caso contrario andiamo in **spazio_negato.**

controllo_fine_stringa: compariamo ciò a cui punta il puntatore alla stringa con il carattere di spazio , fine stringa e a capo. Nel primo caso andiamo a **next ,** nel secondo caso terminiamo la lettura della stringa iniziale e andiamo a scrivere il risultato sulla stringa di output (saltiamo a **fine**) , nel terzo caso saltiamo ad **errore**.

controllo : verifichiamo che il carattere a cui punta il puntatore alla stringa d'ingresso sia minore di 57 , ovvero il numero 9 in ascii, per essere certi che sia un numero (il controllo che sia maggiore o uguale a 48, 0 i ascii, viene fatto in **main_loop**).

segno : se il carattere della stringa in ingresso è minore di 48 dal main_loop arriviamo qui dove verifichiamo se si tratta di un segno (+,-,*,/) , di uno spazio , del carattere di a capo , del carattere di fine stringa o di qualcos'altro.

next : verificato che c'è uno spazio , qui incrementiamo il puntare alla stringa d'ingresso e verifichiamo che non ci siano altri spazi di seguito , o che non ci siano il carattere di a capo o di fine stringa , se ci sono andiamo in **errore** , altrimenti proseguiamo con **main_loop**.

addizione : svolgimento dell'addizione

sottrazione : svolgimento della sottrazione

moltiplicazione : svolgimento della moltiplicazione

divisione : svolgimento della divisione e controllo che il dividendo sia positivo e che il divisore non sia 0 .

negativo : dopo aver trovato il segno – in **segno ,** verifichiamo se il carattere successivo è uno spazio o un fine stringa ,quindi andiamo in **sottrazione** , o se si tratta di qualcos ‘altro e quindi andiamo in **torno_negato.**

spazio : se dopo il primo numero non c’è uno spazio , qui verifichiamo che sitratti di un altro numero , lo convertiamo in numero e dopo aver moltiplicato per 10 il primo glielo sommiamo . Poi incrementiamo il puntatore alla stringa d’ingresso e se troviamo lo spazio mettiamo il numero sullo stack , altrimenti proseguiamo con il ciclo ritornando a spazio.

spazio_negato : il numero da considerare è negativo , dopo il primo numero non abbiamo trovato lo spazio , quindi verifichiamo che ci sia un numero , moltiplichiamo per 10 il primo e gli sommiamo il secondo numero convertendolo da carattere a numero. Fatto ciò , incrementiamo il puntatore alla stringa d’ingresso e verifichiamo se c’è uno spazio , allora mettiamo il valore sullo stack , altrimenti proseguiamo con il ciclo ritornando a **spazio_negato**.

errore : se troviamo il carattere di a capo o qualcosa che non sia un numero, o un segno o ,il carattere di fine stringa , scriviamo in output Invalid dopo aver verificato che non ci siano valori sullo stack che no coincidono con quelli iniziali.

elimina : elimina eventuali valori messi sullo stack che non vengono tolti perché si arriva ad errore.

fine : inizia la fase finale , andando a pulire tutti i registri e mettendo il risultato nel registro eax.

init : mette 10 in ebx , confrontiamo il risultato con 10 se è maggiore o uguale andiamo a dividerlo per 10 in modo da ottenere cifra per cifra , se è minore di 0 il numero è negativo e saltiamo a **num_neg.** Se nessuna di queste è verificata si mette il valore sullo stack e si salta alla fase di **stampa** in output.

dividi : divide il numero per 10 nel caso fosse maggiore o uguale e mette le cifre sullo stack.

dividi_2 : nel caso in cui il numero sia negativo e maggiore o uguale a 10 , divide il numero , mette le cifre sullo stack e salta ad **aggiungi_meno** per ottenere il numero negativo.

stampa : prende i valori salvati sullo stack , li converte in caratteri e li mette in uscita dove punta il puntatore della stringa di output e incrementa il puntatore.

tappo : mettiamo il carattere di fine stringa al termine della stringa di output.

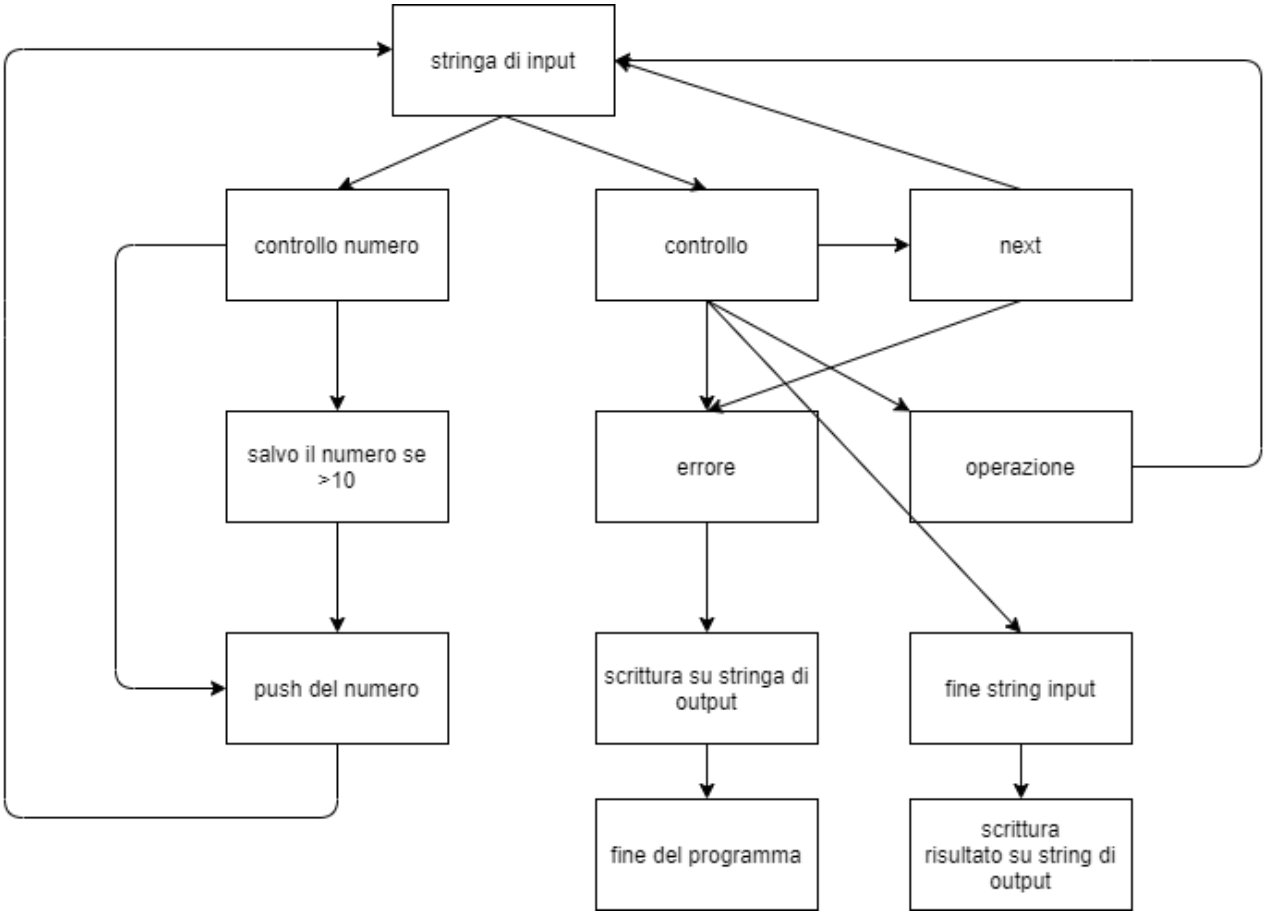
return : ripristino dei valori iniziali dei registri e fine del programma.

num_neg : se il numero è negativo viene negato , e comparato con 10 se è maggiore si va a **dividi_2 ,** altrimenti si prosegue con **aggiungi_meno.**

aggiungi_meno : se il numero è minore di 10 viene messo sullo stack , poi vine messo sulla stringa di output il segno meno e poi si torna a **stampa.** Se il numero è maggiore di 10 si torna a **dividi_2**.

Ci è stato fornito il file **main.c** che chiama la funzione postfix sviluppata in assembly e fa sì che venga letto il file di input e scritto il file di output passando alla funzione le due stringhe.

DIAGRAMMA DI FLUSSO



Dalla stringa di input , contenente l'espressione aritmetica scritta in RPN , attraverso i controlli di numero o segno o fine stringa , si passa rispettivamente a salvare i valori sullo stack , a svolgere le operazioni richieste , o a terminare il programma inserendo nella stringa di output il valore dell'espressione. Se si incontrano caratteri che non sono numeri , segni o il carattere di fine stringa si va in errore dove si inserisce nella stringa di output "Invalid".

DEBUG

Durante la costruzione di *postfix* siamo arrivati a dover usare un debugger per poter leggere e capire i valori dei registri durante l'esecuzione del nostro programma.
Abbiamo usato GDB (GNU Debugger) questo software ci permette di "mettere in pausa" l'esecuzione del nostro programma ed eseguirlo passo passo per controllare il valore delle variabili.
Per prima cosa dobbiamo creare un'etichetta (break point) sul punto nel quale vogliamo che il programma si fermi. Dopo di che dobbiamo lanciare il comando *run* per far ripartire l'esecuzione ed andare nell'istruzione successiva. Con il comando *info_registers* controlliamo il valore attuale che contiene uno qualsiasi dei registri.
Grazie a questo programma siamo riusciti a capire dove si "nascondevano" gli errori per poi poterli sistemare con più facilità.

SCELTE PROGETTUALI

- Non sono state usate altre funzioni oltre a postfix .
- Abbiamo usato il registro ecx come contatore delle volte in cui mettiamo e togliamo i valori dallo stack (push e pop) ,in modo da poter tornare alla situazione dei registri iniziale quando terminiamo il programma.
- Abbiamo messo il controllo anche sul carattere di a capo che non deve esserci nelle stringhe, per cui se si trova il programma va in errore, stampando "Invalid".
- Se si trovano più spazi di seguito o uno spazio e poi il carattere di fine stringa il programma va in errore e stampa "Invalid".
- Nella divisione abbiamo messo il controllo che il dividendo sia positivo , come da specifica , inoltre abbiamo controllato che il divisore non sia 0 .