UNIVERSITA' DEGLI STUDI DI VERONA

Elaborato Assembly

ANNO 2020/2021

Zenaro Stefano ***VR456736***

d'Alessandro Marco ***VR461145***

Farina Christian ***VR456150***

INDICE

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 DEL PROGETTO

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 (*).

GUIDA ALL'USO

Per prima cosa occorre compilare il programma utilizzando il Makefile eseguendo il comando make da riga di comando. L'eseguibile verrà creato da GCC nella cartella bin.

Eseguire il programma da terminale nel seguente modo:

```
$ ./postfix input.txt output.txt
```

Il file `input.txt` contiene una sola riga con l'espressione in RPN da analizzare ed elaborare.

Il file `output.txt` contiene il risultato della espressione in RPN data in input attraverso `input.txt` (se vengono eseguite operazioni invalide o l'input è invalido il file output.txt conterrà la dicitura "Invalid").

VARIABILI

Le seguenti variabili vengono utilizzate dalla funzione postfix():

- **Invalid** (byte): memorizza la validità dell'input (0: vero, 1: falso)
- **numeric** (byte): flag che indica se si sta leggendo un operando/numero (0: vero, 1: falso)
- **numero** (long): variabile temporanea che memorizza ogni operando letto in input
- **is_negative** (byte): flag che indica se l'operando letto è un numero negativo (0: vero, 1: falso)

La seguente variabile viene utilizzata dalla funzione itoa():

- **itoa_is_negative** (byte): flag che indica se il numero da convertire in stringa è negativo (0: vero, 1: falso)

FUNZIONI

Il programma, per essere mantenibile e testabile, è suddiviso in funzioni e ogni funzione è contenuta nel suo omonimo file.

I parametri vengono passati attraverso lo stack mentre i valori di ritorno, dove presenti, vengono memorizzati nel registro EAX.

Qui di seguito sono riportati i loro nomi e il compito che essi svolgono:

int addizione(int a, int b)	Svolge l'operazione di addizione (restituisce EAX = a + b)
int divisione(int a, int b)	Svolge l'operazione di divisione (restituisce EAX = b / a)
Int prodotto(int a, int b)	Svolge l'operazione di prodotto (restituisce EAX = a * b)
Int sottrazione(int a, int b)	Svolge l'operazione di sottrazione (restituisce EAX = b - a)
int is_operand(char tchar)	Restituisce EAX = 0 se il valore del parametro è un carattere numerico, altrimenti restituisce EAX = 1
int is_operator(char tchar)	Restituisce EAX = 0 se il valore del parametro è un operatore (+, -, *, /), altrimenti restituisce EAX = 1
int is_valid_char(char tchar)	Restituisce EAX = 0 se il valore del parametro è un carattere valido, altrimenti restituisce EAX = 1
void itoa(int num, char * output)	Converte un numero <num> in una stringa di caratteri (viene memorizzata in <output>)
void write_result(***int result,*** ***int valid,*** ***char * output*** ***)***	Se valid e' 0 richiama itoa() per scrivere in <output> il risultato della espressione contenuto in <result>. Se valid e' 1 scrive in <output> la stringa "Invalid"

void postfix(***char * input,*** ***char * output*** ***)***	Scorre la espressione in input <input> e richiama le altre funzioni per: 1. verificare l'input (richiamando le funzioni is_operator(), is_operand() e is_valid_char()) 2. recuperare e convertire gli operandi da stringhe a numeri 3. eseguire le operazioni dettate dagli operatori 4. scrivere in <output> il risultato della espressione oppure "Invalid" utilizzando la funzione write_result()

Il file ***main.c***, scritto in C e già precedentemente fornito, si occupa di:

- Leggere due parametri passati da terminale: il percorso del file in input e il percorso del file in output
- Recuperare dal file in input l'espressione matematica in notazione polacca inversa da calcolare e di memorizzarla in una stringa
- Richiamare la funzione postfix() passando due puntatori: il puntatore alla stringa in input e il puntatore alla stringa che conterrà l'output. La funzione postfix() scriverà la stringa in output.
- Scrivere sul file in output ciò che è stato scritto nella stringa dell'output modificata da postfix()

DIAGRAMMI DI FLUSSO

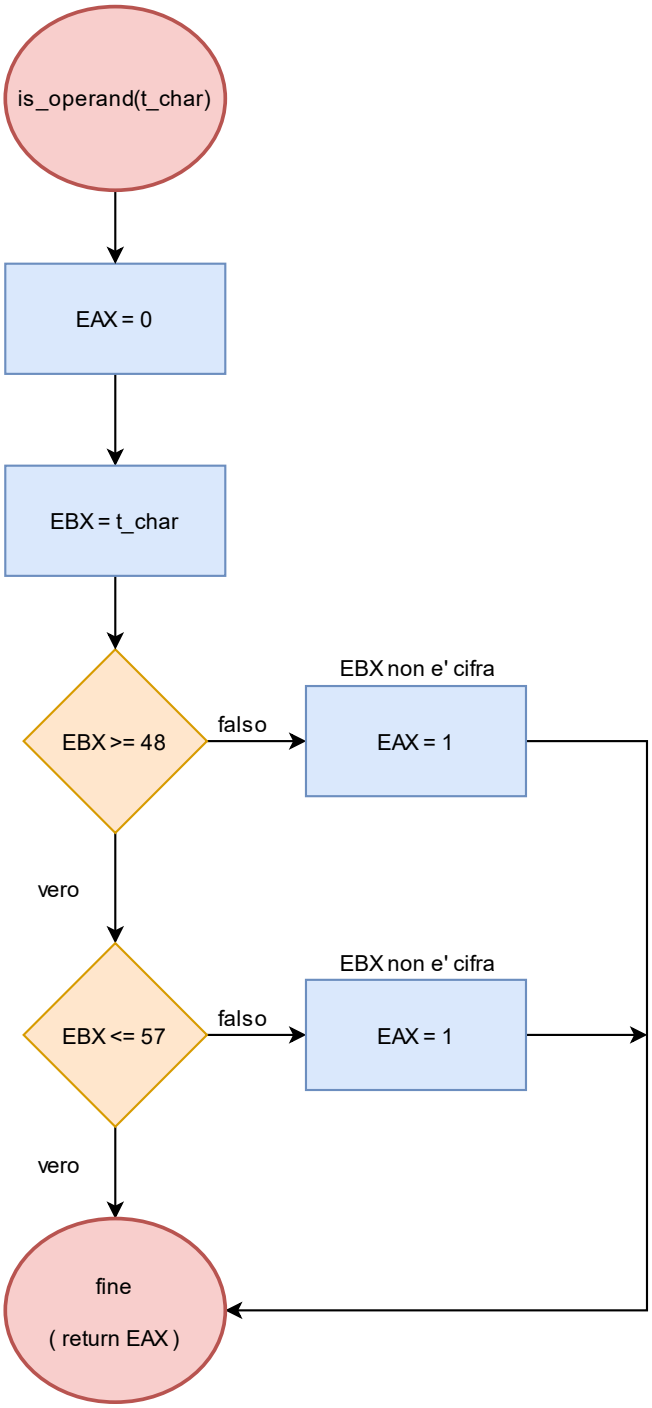

Figura 1 **Flowchart is_operand( )**

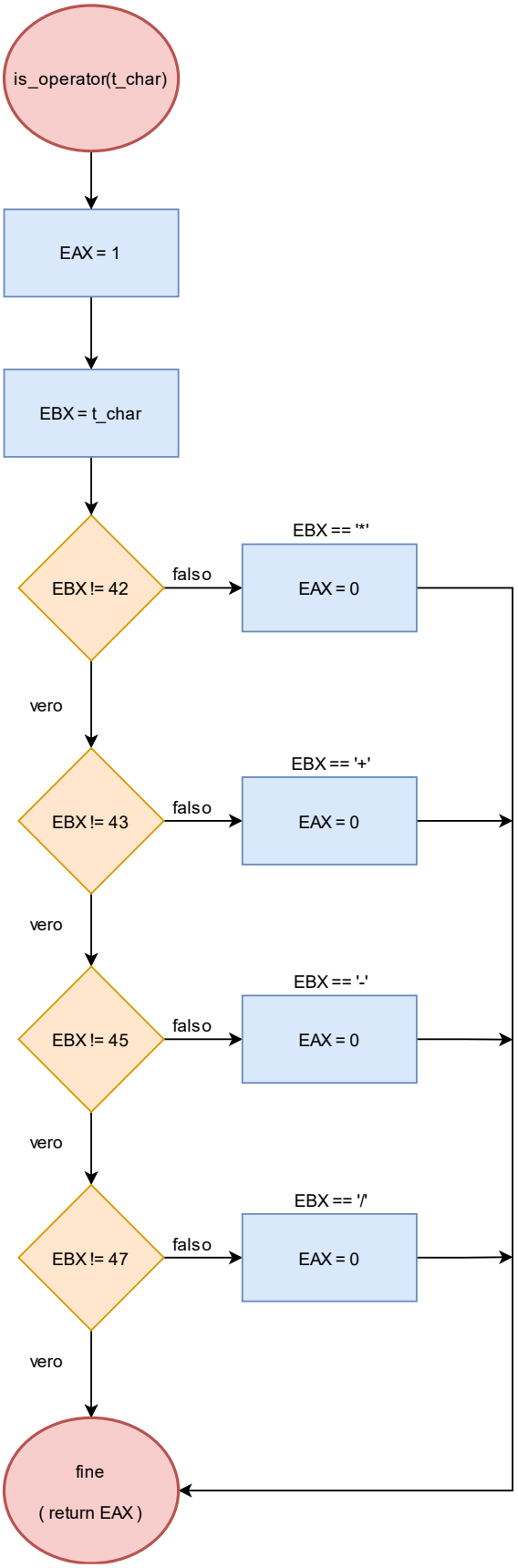

Figura 2 **Flowchart is_operator( )**

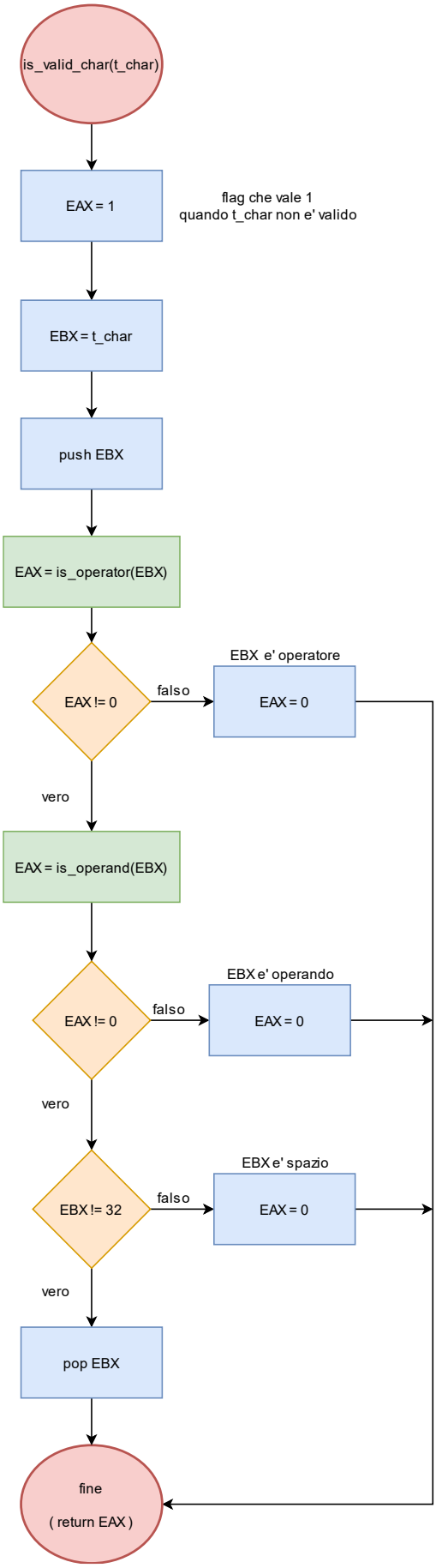

Figura 3 **Flowchart che verifica di validità carattere**

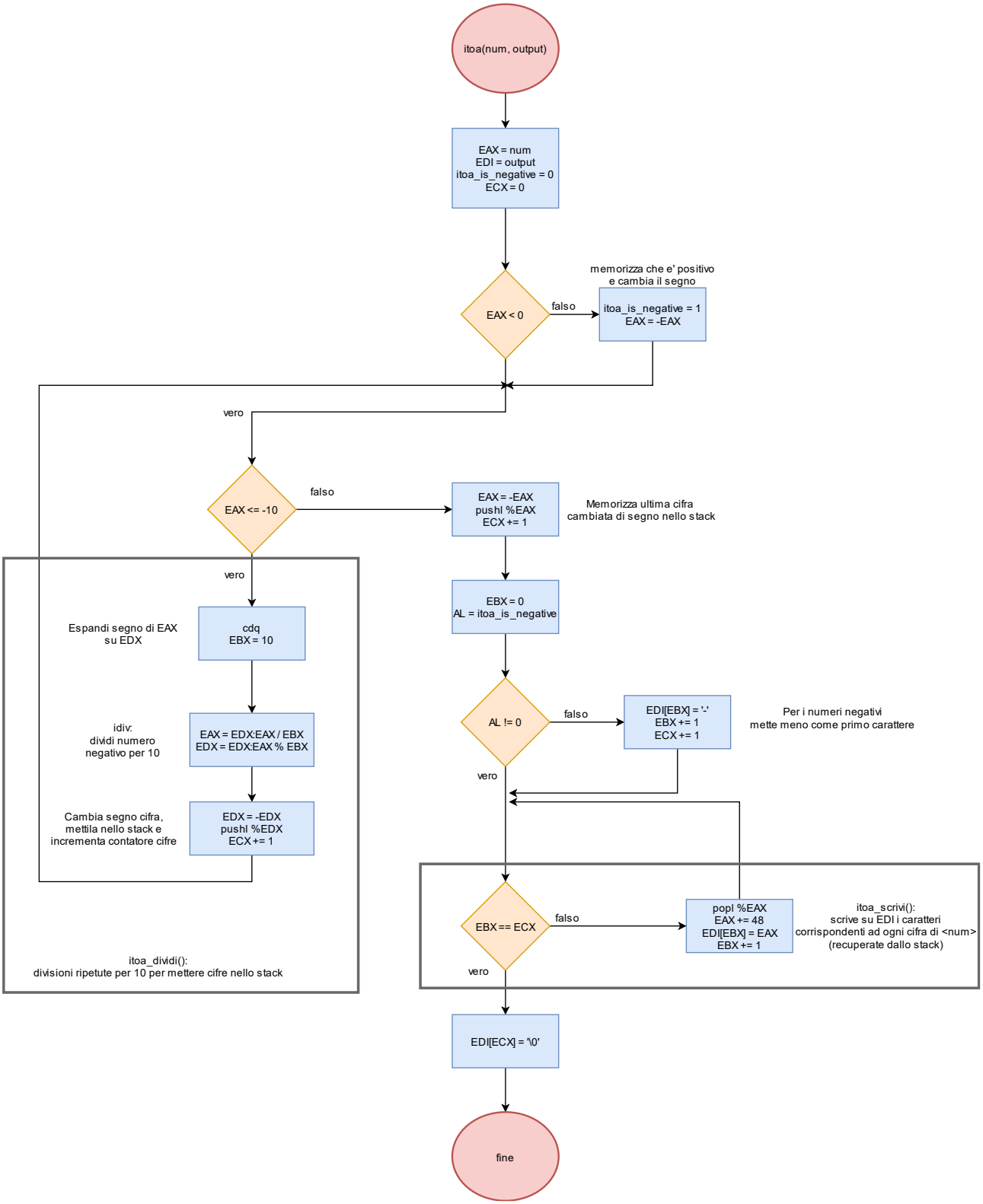

Figura 4 **Flowchart itoa( )**

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.*

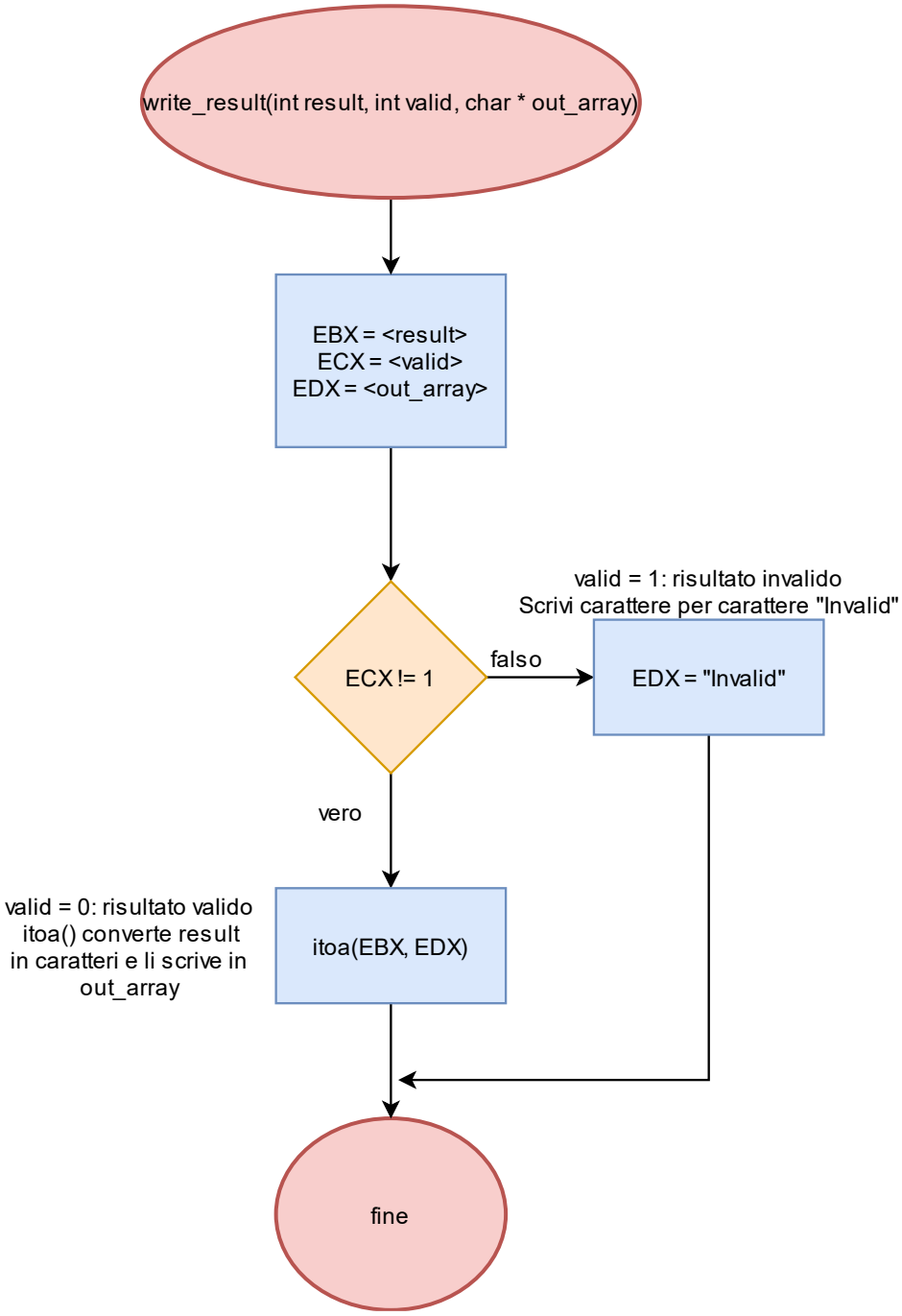

Figura 5 **Flowchart write_result( )**

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
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_negative caspisco 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:
  //non so ancora se è tutto corretto
  salto a postfix_write_appearently_valid_result

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:
  //la formula è ben formata, quindi scrivo il risultato e concludo
  chiamo write_result

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.

SCELTE PROGETTUALI

- Le funzioni ricevono parametri attraverso lo stack e restituiscono il proprio riscontro, quando necessario, nel registro EAX

- Le etichette di ogni funzione iniziano con “<nome_funzione>_” per evitare conflitti di nomi.

- Per gestire il caso in cui l’output sia *-2147483648* (il numero negativo con il valore assoluto più grande) la funzione itoa() gestisce internamente numeri negativi per poter ricavare le singole cifre invece di numeri positivi.

  Questo è necessario perché cambiare di segno il numero *-2147483648* porterebbe ad un overflow (il numero più grande rappresentabile in 32 bit è *2147483647*) e il risultato del programma verrebbe 0.

  Questo non porterà comunque ad un risultato invertito di segno perché itoa() utilizza la variabile itoa_is_negative per ricordare se il numero e’ negativo o meno.

- Le funzioni assembly sono state testate mediante unit test: ogni funzione è stata richiamata da codice C per verificare che i valori di ritorno ottenuti in base ai parametri passati siano corretti.