Università degli studi di Verona

Laboratorio di Architettura degli Elaboratori

Elaborato ASM

A.A. 2020/2021

Andrea Liboni VR456665
Luciano Mateias VR456165

Indice

1 Introduzione **2**
1.1 Obbiettivo 3
1.2 Requisiti 3

2 Variabili **4**

3 Funzioni **5**

4 Flow chart **6**

5 Scelte progettuali **7**

1. Introduzione

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

1.1. Obbiettivo

Si scriva un programma in assembly che legga in input una stringa rappresentante un'espressione ben formata in numero di operandi e operazioni in RPN (si considerino solo gli operatori + - * /) e scriva in output il risultato ottenuto dalla valutazione dell'espressione.

Potete usare questo calcolatore online per vedere l'esecuzione passo-passo:

https://www.free-online-calculator-use.com/postfix-evaluator.html

1.2. Requisiti

Le espressioni che verranno usate per testare il progetto hanno i seguenti 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. Esempi:
  - -6 / 2 che in RPN diventa -6 2 / non è valido.
  - 6 / -2 che in RPN diventa 6 -2 / è valido.
- Tra ogni operatore e/o operando vi è 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.

2. Variabili

Elenco delle variabili utilizzate, suddivise per file di appartenenza:

- **postfix.s**
  - `integer` [long]: valore intero che viene aggiornato alla lettura di ogni nuova cifra di un numero.
  - `esp_value` [long]: valore iniziale del registro esp, verrà ripristinato al termine dell'esecuzione della funzione postfix.
  - `invalid_str` [char[8]]: stringa di output in caso di input non valido.
  - `op` [byte]: booleano che indica se il precedente carattere era un operando. Utilizzato per evitare push non volute su stack in fase di salvataggio degli interi.
  - `neg` [byte]: booleano che indica se il numero da salvare è negativo (che è dunque preceduto da un segno negativo).
- **itoa.s**
  - `neg` [byte]: booleano che indica se il numero da salvare è negativo (che è dunque preceduto da un segno negativo).
- **strcpy.s**

  Nessuna variabile utilizzata.

3. Funzioni

- **postfix.s**

  Legge in input una stringa rappresentante un'espressione ben formata in numero di operandi e operazioni in RPN (secondo le specifiche) e scrive in output il risultato ottenuto dalla valutazione dell'espressione.

  La funzione riceve in ingresso due parametri chiamati input e output, puntatori alle corrispettive posizioni di memoria.

  Viene modificato il registro eax.

- **itoa.s**

  Converte un intero nella corrispettiva stringa (array di caratteri).
  Se l'intero è negativo viene aggiunto un segno negativo alla stringa di output prima della conversione.

  La stringa viene copiata (carattere per carattere) alla posizione puntata dal registro `edi`, il cui valore viene fornito dall'unico parametro della funzione, corrispondente alla destinazione.

  Viene modificato il registro eax.

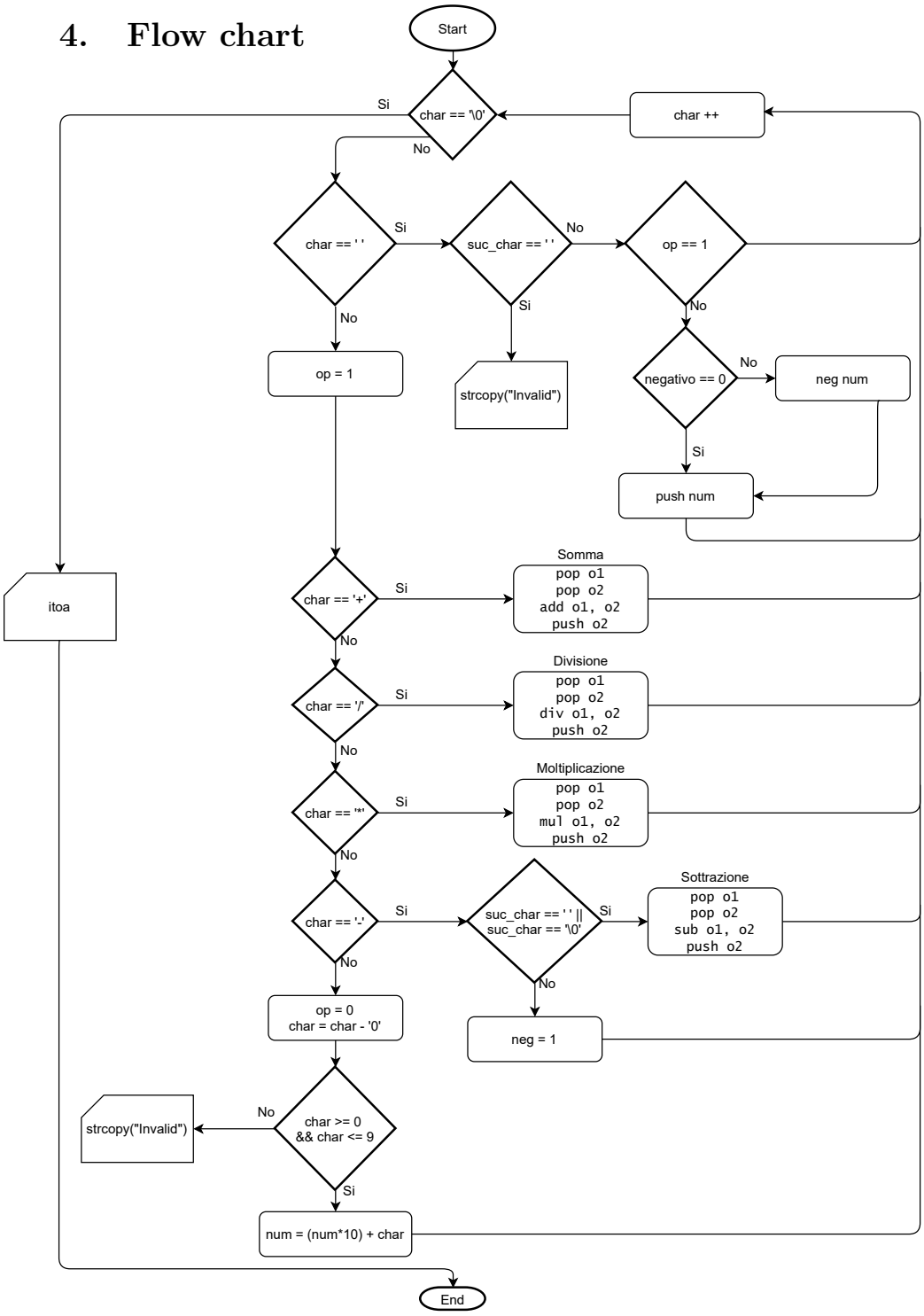
- **strcpy.s**

  Copia tutti i caratteri dalla posizione puntata dal registro `esi` alla posizione puntata dal registro `edi` (spostamento di byte singoli); termina quando il carattere puntato dal registro esi è un terminatore di stringa, cioè '\0'.

  Il valore dei due registri `esi` ed `edi` è dato dai due parametri della funzione, considerabili come sorgente e destinazione della stringa che vogliamo copiare.

  Non vengono modificati registri ma viene azzerato il `direction flag`.

4. Flow chart


Start
char == '\0'
Si
No
char ++
char == ' '
Si
No
suc_char == ' '
No
Si
op == 1
No
op = 1
strcopy("Invalid")
negativo == 0
No
neg num
Si
push num
itoa
char == '+'
Si
Somma
pop o1
pop o2
add o1, o2
push o2
No
char == '/'
Si
Divisione
pop o1
pop o2
div o1, o2
push o2
No
char == '*'
Si
Moltiplicazione
pop o1
pop o2
mul o1, o2
push o2
No
char == '-'
Si
suc_char == ' ' ||
suc_char == '\0'
Si
Sottrazione
pop o1
pop o2
sub o1, o2
push o2
No
op = 0
char = char - '0'
neg = 1
char >= 0
&& char <= 9
No
strcopy("Invalid")
Si
num = (num*10) + char
End

5. Scelte progettuali

Elenco delle scelte progettuali attuate:

- **Output Invalid**

  La restituzione di un valore *Invalid* avviene nei seguenti casi:

  - Presenza di carattere non numerico e/o diverso dai 4 operandi.
  - Presenza di più caratteri spazio consecutivi.

- **strcpy**

  La funzione richiede due parametri: input e output. È stato deciso di fornire questi due valori e di non utilizzare i registri `esi` ed `edi` che già li contengono.
  La funzione non modifica dunque il valore di questi due registri.

  Questo comportamento permette il riutilizzo della funzione in altri script, senza intaccarne il corretto funzionamento.

- **Registro esp**

  Dopo il salvataggio dei registri general purpose a inizio script, viene salvato il valore del registro `esp`. Lo stesso registro viene ripristinato al termine dell'esecuzione dello script, subito prima il ripristino dei registri general purpose.

  In caso di input che generano un output *Invalid* è possibile che lo stack resti *sporco*, ma tramite il ripristino del registro `esp` lo script termina correttamente, ignorando, per così dire, gli elementi aggiuntivi.