2020-2021

ELABORATO ASSEMBLY

ARCHITETTURA DEGLI ELABORATORI

SILVER GJEKA (VR450793).

INTRODUZIONE E DESCRIZIONE DELL'ELABORATO

Con il progetto da noi presentato si vuole realizzare un programma Assembly che legga da input testuale una stringa rappresentante un'espressione aritmetica espressa attraverso la notazione polacca inversa (*Reverse Polish Notation*, RPN). In seguito, dopo aver calcolato i risultati delle operazioni, il solo risultato deve essere scritto su un file di output.

La RPN sfrutta lo stack per il suo funzionamento. L'espressione viene letta da sinistra a destra, un carattere alla volta. Gli operandi vengono momentaneamente salvati sullo stack attraverso una *push*. Ciò avviene per permettere di tenere in memoria temporaneamente i numeri da valutare in attesa di un operatore valido. Gli operandi vengono infatti eliminati dalla pila, attraverso un'operazione di *pop*, quando nella stringa dell'espressione viene letto uno dei quattro operatori fondamentali che verrà descritto in seguito. Dopo la pop dallo stack viene valutata l'operazione, calcolato il risultato e in seguito quest'ultimo viene salvato nuovamente sullo stack, attraverso un'ulteriore push. Si continua così a valutare il resto dell'operazione fino alla fine della stringa. Si ricorda che il funzionamento dello stack segue la politica LIFO (*Last In First Out)* per cui l'ultimo carattere inserito nella pila è anche quello che viene tolto dalla pila per primo.

L'espressione deve essere ben formata, vale a dire che tra ogni operatore e operando vi deve essere uno spazio e il carattere subito successivo all'ultimo operatore deve essere il carattere di terminazione "\0", non sono permessi spazi o "a capo" ("\n").

Sono stati imposti alcuni vincoli e requisiti quali:

- Le quattro operazioni fondamentali considerate sono addizione, moltiplicazione, sottrazione e divisione, univocamente rappresentate nell'ordine dai seguenti simboli: " + ", " * ", " - ", " / " - Gli operandi sono esclusivamente numeri interi, composti da uno o più cifre e possono essere numeri negativi
- Solamente gli operandi negativi sono rappresentati esplicitamente con segno. In particolare, nel caso di numero negativo il segno e il numero non sono separati da uno spazio - Gli operandi hanno un valore massimo rappresentabile in 32-bit
- Il risultato di moltiplicazione e divisione è rappresentabile solo in 32-bit
- Il risultato di una divisione da solo numeri interi, quindi senza resto
- Il divisore può essere negativo
- Non vi sono limiti di lunghezza per le espressioni
- Se le stringhe inserite non sono valide (contengono simboli diversi da operatori o numeri) o presentano problemi nel modo e ordine in cui sono descritte, il programma deve stampare nel file di output la scritta "Invalid"
- Il programma deve essere denominato "postfix.s" e verrà lanciato da riga di comando con due parametri, nell'ordine il nome del file di input e il nome del file di output

Per controllare il funzionamento del codice è necessario creare i file oggetto e l'eseguibile, spostarsi nella cartella bin dove sono contenuti l'eseguibile e i file test per l'input.

Da linea di comando viene quindi chiamato il programma e vengono passati due parametri, il primo deve essere il nome del file di input che si vuole testare, il secondo il nome del file di output su cui verrà memorizzato il risultato dell'operazione. Inoltre, il programma main.c fa si che avvenga anche una stampa a video per un ulteriore controllo.

Viene qui riportato un esempio che utilizza il file in_1.txt fornito per i test.


```
studente@debian:~/Scrivania/asm$ ls
bin  Makefile  obj  src
studente@debian:~/Scrivania/asm$ cd bin
studente@debian:~/Scrivania/asm/bin$ ls
in.txt  out.txt  postfix
studente@debian:~/Scrivania/asm/bin$ ls
in_1.txt  in_3.txt  in_5.txt   out_2.txt  out_4.txt  postfix
in_2.txt  in_4.txt  out_1.txt  out_3.txt  out_5.txt
studente@debian:~/Scrivania/asm/bin$ ./postfix in_1.txt output.txt
12
```

CONTENUTO DELLA CARTELLA

La cartella asm presenta una struttura fissa:

- asm/
  - src/
    - la cartella contenente i sorgenti, contiene i seguenti file:
      - main.c : contiene il file sorgente C con il compito di chiamare la funzione assembly. È stato fornito dal professore e non può essere editato
      - postfix.s : file assembly principale con il compito di leggere la stringa, determinare se sia valida ed eventualmente svolgere le operazioni descritte, salvando poi il risultato sul file di output.
  - obj/
    - cartella di supporto per la creazione del file oggetto. Va svuotata prima della consegna
  - bin/
    - cartella dove verrà creato l'eseguibile. Da consegnare vuota
  - Makefile
    - file contenente le istruzioni necessarie alla compilazione e al linking. Permette di generare i file oggetto (nella cartella obj) e gli eseguibili (nella cartella bin).
  - Relazione.pdf

GESTIONE DEI PARAMETRI DI RITORNO DELLA FUNZIONE

Convenzionalmente, i parametri eventualmente passati alla funzione Assembly si trovano nello stack a partire dall'ultimo a destra, mentre l'eventuale valore di ritorno va inserito in AL, AX, EAX o EDX:EAX, coerentemente con la sua dimensione in bit.

Il programma main.c sfrutta la funzione *char* retrieve_input(char* filename)* per prendere in input il nome del file su cui è stata descritta l'espressione da valutare e restituisce una stringa che verrà poi passata come parametro alla funzione assembly. Il parametro passato si trova nello stack.

Alla funzione assembly definita come *extern int postfix(char* input, char* output)* vengono passata due parametri di tipo puntatore a char e entrambi si ritrovano all'interno dello stack, come vuole la convenzione di assembly. La comunicazione tra la funzione main.c e postfix.s avviene tramite i due parametri input e output. In particolare, il risultato viene passato al main sfruttando il registro EDI e riempiendo la stringa un carattere alla volta.

FUNZIONAMENTO DEL CODICE

Il file main.c richiama al suo interno la funzione postfix che consiste in un modulo assembly che realizza il calcolo dell'espressione espressa secondo la modalità RPN.

Il file postfix.s dichiara le variabili necessarie a verificare se sono stati commessi errori, determinare se è stata trovata un'operazione da svolgere e se il numero è negativo o positivo.

Quando un programma viene lanciato da linea di comando è possibile fornire alcuni parametri e in questo caso i parametri forniti sono i nomi del file di input e del file di output necessari al test e al controllo del progetto. Le informazioni relative il programma, compresi i parametri forniti precedentemente, vengono memorizzati sullo stack. In particolare, l'ordine di memorizzazione prevede che nella parte più alta, quindi dove punta lo stack pointer (ESP), è contenuto il numero di parametri passati da riga di comando, successivamente la locazione di memoria dove è memorizzato il nome del programma, in seguito l'indirizzo dove è eventualmente memorizzato il primo parametro e nell'ordine, se esistono, gli indirizzi dei successivi parametri.

Le prime righe del codice, dopo la dichiarazione delle variabili, si occupano di recuperare dallo stack i parametri passati dalla funzione main.c. In seguito, viene preso ESP e tramite un indirizzamento indiretto a registro con spiazzamento il punto dello stack dove risiede il primo parametro viene salvato nel registro ESI. In questo modo il source index contiene il puntatore alla stringa di input (per l'esattezza al file di input che verrà utilizzato per verificare il funzionamento del programma). Svolgiamo poi la stessa azione per quanto riguarda il secondo parametro. Ciò permette di salvare nel registro destination index (EDI) il riferimento al file di output dove verrà salvato il risultato del programma.

Ci occupiamo in seguito di salvare su stack (tramite un'operazione di push) lo stato dei registri general purpose in modo tale da non rischiare di perdere informazioni riguardanti il contenuto precedente dei registri e permettendoci così di poterli azzerare per avere la certezza che in essi non vi siano contenuti valori che potrebbero inficiare il corretto funzionamento del codice.

Etichetta control:

Viene poi eseguito un salto incondizionato al controllo della stringa. Quest'ultima deve infatti essere sempre controllata all'inizio del flusso del processo. Il controllo verifica che la stringa passata dal file di input sia ben formata:

- Tra ogni operatore e operando vi deve essere uno spazio
- l'ultimo operatore è seguito immediatamente dal simbolo di terminazione "\0" ● la

stringa può contenere solo numeri e uno o più dei simboli che identificano le operazioni ● la stringa non può iniziare con uno spazio

Sfruttando l'istruzione *compare* vengono confrontati due operandi. L'istruzione setta i flag del registro EFLAGS e questi ultimi verranno utilizzati in seguito per saltare ad etichette del codice con il compito di gestire eventuali problemi. Le istruzioni di salto operano infatti in base ai flag impostati da questa istruzione.

Il contenuto della cella di memoria indirizzata tramite il registro ESI viene quindi comparato con una serie di valori costanti, nell'ordine corrispondenti nella tabella ASCII al simbolo di spazio, ai simboli scelti per rappresentare le operazioni tranne la sottrazione ("+", "*", "/"), al simbolo di terminazione e al simbolo di "a capo" e, in particolare, al simbolo della sottrazione che potrebbe anche essere il simbolo di negativo legato al primo operando.

La stringa deve quindi necessariamente iniziare con un operando corretto quindi un numero intero con segno positivo o negativo.

Nell'eventualità che una di queste comparazioni setti il flag ZF a 0 avviene un salto condizionato all'etichetta che gestisce quel particolare caso. Se l'inizio della stringa risulta invece corretto si prosegue con il controllo degli operandi.

Etichetta operando:

Nel caso l'etichetta di controllo non verifichi nessun problema di validità il jump incondizionato porta al controllo dell'operando.

È necessario verificare che il carattere sia un numero compreso tra 0 e 9 e non un altro qualsiasi carattere della tabella ASCII. Ricordandosi che nella tabella i numeri da 0 a 9 sono rappresentanti con i valori compresi tra 48 e 57, estremi inclusi, si compara il carattere con le costanti rappresentanti i due estremi. Viene poi valutata l'istruzione "*jump if less"* in quanto se il carattere ha un valore minore non può essere un numero e poi viene valutata "*jump if greater"* perché, allo stesso modo, se il carattere ha un valore maggiore non può comunque rappresentare un numero.

Se non avviene un salto si prosegue con l'esecuzione sequenziale delle successive

righe di codice. ## Etichetta take:

Una volta entrati in questa fase del codice, vuol dire che ci si sta leggendo un carattere che corrisponde ad una cifra e si ha quindi il compito di ricreare il numero che rappresenta all'interno di

un registro. Vengono prima di tutto azzerati i registri ebx e edx attraverso un'operazione di OR esclusivo bit a bit.

L'intenzione è quella di convertire i caratteri che rappresentano l'operando in un numero che verrà salvato nel registro EAX.

Sposto il contenuto di ESI, quindi il carattere corrente, nella parte bassa del registro EBX (BL). Per ottenere l'effettivo valore dell'operando in cifre viene sottratto il valore costante 48 al registro EBX e salvato poi in EBX stesso.

Il valore costante 10 viene salvato nel registro EDX attraverso un'istruzione movl e in seguito con un'istruzione "mull op" il registro EAX viene moltiplicato con il registro scelto come operando, quindi EDX in questo caso. La moltiplicazione per 10 del contenuto del registro EAX permette di ottenere il numero corretto secondo la notazione posizionale in base decimale. Il risultato viene salvato in EAX e, per la stessa ragione, quest'ultimo viene poi sommato al registro EBX che contiene la cifra corrente della stringa.

Viene impostata la variabile operazione a 0 per segnalare che si sta ancora gestendo un numero e non un operatore. Si prosegue in seguito con il successivo carattere.

Avviene poi un salto non condizionato all'etichetta *next_one* per permettere di proseguire con la valutazione del successivo carattere.

Etichetta next_one:

L'etichetta ha il compito di incrementare ESI per portare il puntatore al carattere successivo e il programma continua nuovamente con il controllo della stringa riprendendo dall'etichetta successiva control.

Abbiamo fino a qui spiegato cosa accade e quali parti del codice vengono visitate ed eseguite nel caso venga trovato un carattere che corrisponde ad una cifra positiva. Descriveremo in seguito cosa accade quando vengono trovate delle stringhe non valide, un operatore oppure uno spazio. La spiegazione seguirà l'ordine in cui sono descritte le etichette sul codice.

Etichetta segno:

Nel caso di simbolo "-" si presentano più casi:

- potrebbe essere il simbolo che indica il segno negativo dell'operando ed essere quindi valido
- potrebbe essere il simbolo di sottrazione se compare dopo almeno due operandi ● se fosse il primo carattere dell'espressione e subito dopo vi fosse uno spazio e non il numero, ci si troverebbe di fronte ad un'espressione non valida
- (non ci sono almeno due operandi, quindi invalid??)

Va quindi valutato il carattere subito successivo ad ESI con un indirizzamento indiretto a registro con spiazzamento. Viene nuovamente sfruttata l'istruzione compare per confrontare il contenuto del registro e delle costanti rappresentanti dei simboli.

Se si trova uno spazio va trattato come un simbolo di sottrazione ed è necessario eseguire un salto all'etichetta *sot*. Allo stesso modo, è necessario spostarsi nell'etichetta nel caso il carattere successivo sia di terminazione. In questo caso, infatti, l'espressione è finita ed è necessario completare gli ultimi calcoli.

Nel caso non ci si trovi in uno di questi due casi, va controllato se il carattere successivo sia un numero e non un altro carattere della tabella ASCII. Seguendo le stesse regole descritte precedentemente per l'individuazione della validità numerica dell'operando viene controllato che il carattere sia un numero (0-9). Se il carattere successivo non rappresenta una cifra avviene un salto all'etichetta *inv*.

L'ultima valutazione prevede di controllare se il carattere successivo è un "line feed" e quindi si tratta nuovamente di una sottrazione.

Se nessuna delle istruzioni di salto viene realizzata, ci si trova nella situazione in cui è stato individuato un numero negativo ed è quindi necessario settare a 1 la variabile *neg* per indicare che l'operando che seguirà il segno è negativo.

Segue poi un salto incondizionato all'etichetta *next_one.*

Etichetta som:

Vanno azzerati i precedenti valori dei registri EAX e EBX con un or esclusivo. Trovandosi nel caso in cui è possibile calcolare un'operazione vanno recuperati dallo stack gli ultimi due operandi caricati attraverso una pop. Questi ultimi vengono memorizzati sui registri EAX e EBX per poter poi essere sommati, il risultato verrà memorizzato nel registro EAX. Il risultato va caricato nuovamente sullo stack attraverso una push.

Viene decrementato il contatore delle push degli operandi caricati su stack e la variabile operazione viene risettata ad 1 per indicare che è stata eseguita un'operazione aritmetica. In seguito, vengono resettati i registri al valore iniziale zero ed è eseguita un'istruzione di salto all'etichetta next_one per passare al successivo carattere.

Etichetta sot:

La parte di codice che si occupa di sottrarre gli operandi funziona similmente alla somma. I registri vengono resettati, viene eseguita una doppia pop per recuperare gli ultimi due operandi, calcolata la sottrazione e salvato il risultato nel registro EAX e il valore viene nuovamente caricato su stack. Viene aggiornato il contatore delle push, aggiornata la variabile operazione e resettati i registri.

Etichetta mull:

La moltiplicazione segue le stesse modalità delle due operazioni precedenti. Viene utilizzata l'istruzione *imul operando* che moltiplica il contenuto del registro EAX per l'operando scelto, in questo caso EBX. L'istruzione permette di moltiplicare numeri con segno. Il risultato viene salvato nel registro EAX.

Etichetta div:

La divisione prevede l'azzeramento dei registri EAX, EBX e EDX. Viene eseguita la pop degli ultimi due valori memorizzati su stack ed eseguita la divisione. Secondo la convenzione assembly nel caso in cui l'operando sia rappresentato da 32-bit il valore ottenuto dalla concatenazione di EDX e EAX viene diviso per l'operando. Il risultato dovrebbe essere salvato nella combinazione dei registri EDX

e EAX, in cui nell'ordine vengono salvati il resto e il quoziente. Ciononostante, essendo

richiesta una divisione con risultato intero in 32 bit, viene considerato effettivamente solo il registro EAX che conterrà il risultato. Quest'ultimo, come negli altri casi, deve essere caricato su stack.

Etichetta spazio:

Vi sono alcuni casi in cui la stringa risulta non valida:

- Il carattere successivo è il simbolo di terminazione, la stringa non è terminata correttamente subito dopo l'operatore
- Il carattere successivo è un line feed
- Il carattere successivo allo spazio è un altro spazio

È necessario comprendere se è avvenuta un'operazione attraverso una comparazione tra la costante 1 e la variabile *operazione*. Nel caso sia avvenuta si prosegue alla lettura del carattere successivo, saltando all'etichetta *next_one.*

Si controlla poi se l'operando trovato precedentemente è negativo. In caso non lo sia viene eseguita si salta all'etichetta *push.* Altrimenti il registro EBX va azzerato, poi deve esservi caricato il valore -1e il registro EAX viene moltiplicato per EBX, ottenendo un numero negativo.

Etichetta push:

L'operando è positivo, quindi viene azzerata la variabile flag *neg* che segnala se si sta lavorando con un numero positivo o negativo. Il valore contenuto nel registro EAX, ovvero l'operando, viene caricato sullo stack e il registro viene azzerato. In seguito, viene incrementato il registro ECX utilizzato come contatore per determinare quanti operandi sono stati caricati sullo stack. Si prosegue poi con la valutazione del carattere successivo.

Etichetta inv:

Vanno controllate quante push degli operandi sono state realizzate. Nel caso in cui il contatore sia a 0 è necessario saltare all'etichetta *invalid2*, altrimenti si entra in un loop che permette di eseguire una pop dei valori contenuti nello stack, decrementando automaticamente il registro ECX. Quando il registro ECX è a zero lo stack è stato svuotato dai valori caricati precedentemente.

Etichetta invalid2:

Vengono resettati i registri EAX e EBX e viene caricata la stringa di errore "Invalid", rappresentata dalla variabile *error* sul registro EAX per la stampa del messaggio di errore. Si prosegue con la successiva etichetta.

Etichetta invalid_end:

Il contenuto del registro EAX va spostato carattere per carattere nel registro destinazione EDI e per realizzare questo compito viene utilizzato il registro EBX come registro temporaneo.

I registri EAX e EDI vengono poi incrementati di 1 per poter continuare il riempimento della

stringa.

È necessario controllare se il carattere successivo della stringa in EAX è il carattere di terminazione, se la stringa è terminata va inserito nella stringa a cui fa riferimento EDI il carattere di terminazione. Infine vengono ripristinati i registri ed effettuato il ritorno alla funzione chiamante. 8

Altrimenti, se non vi è il carattere di terminazione si ritorna all'inizio dell'etichetta *invalid_end.*

Etichetta end:

Si entra nell'etichetta end quando viene trovato il simbolo di terminazione alla fine della stringa. Vengono azzerati i registri general purpose.

Viene spostato il valore costante 10 sul registro EAX e si prende dallo stack l'ultimo valore caricato, corrispondente al risultato.

In seguito, si azzera la variabile *neg* per riportare la variabile allo stato iniziale.

Il valore di EAX viene comparato al valore 0 e se il risultato è un numero maggiore o uguale a zero si può passare alla fase successiva e quindi all'etichetta *result.* In caso contrario, stiamo gestendo un numero negativo, quindi azzeriamo EBX, si carica sul registro il valore -1 e il valore di EAX viene moltiplicato con EBX. La flag *neg* è settata ad 1 per indicare per il numero è negativo.

Etichetta result:

L'eventuale resto delle operazioni contenuto in EDX viene azzerato perché non è necessario riportarlo nel risultato. Viene presa una cifra per volta dividendo il valore di EAX per EBX, precedentemente settato a 10. Il resto, come da convenzione, viene salvato nel registro EDX e questo viene caricato sullo stack, incrementando il contatore delle push, ovvero ECX.

Si controlla poi se il quoziente è arrivato ad avere il valore zero, nel caso non valga ancora zero si continua a fare ritorno all'inizio dell'etichetta *result* creando un ciclo.

Viene salvato nel registro EDX il puntatore al primo carattere della stringa di output. In seguito si deve valutare se il numero è negativo o positivo, se positivo si passa all'etichetta *finish.* Se ciò non avviene il registro EAX può essere azzerato. Poi al valore di EAX viene aggiunto il valore costante 45 che determina il segno "-". Il carattere contenuto nella parte bassa del registro viene caricato all'inizio della locazione di memoria contenuta in EDI, ovvero all'inizio della stringa di output. Viene poi incrementato il registro per permettere di continuare con la creazione della stringa di output.

Etichetta finish:

Il registro ECX contiene l'informazione riguardante quante push sono state realizzate per sapere quante cifre devono essere recuperate.

Il registro EAX va azzerato per poter essere utilizzato come destinazione della pop dallo stack. Alla cifra recuperata va aggiunto il valore 48 per poter ottenere il valore corrispondente alla cifra nella tabella ASCII e trasformare quindi la cifra in un carattere. Il carattere viene spostato nella stringa indirizzata da EDI e lo stesso registro viene

incrementato per permetterne il completamento. È necessario poi realizzare un loop per assicurarsi di recuperare tutte le cifre, il loop si basa sul controllo del registro ECX e continuerà fino a quando il registro assume il valore zero. Completato il recupero di ogni carattere va inserito il carattere di terminazione.

Etichetta exit:

l'ultima parte del codice si occupa di togliere il valore dei registri precedente la chiamata alla funzione per ripristinare lo stato del codice corretto. Le pop, per via del funzionamento dello stack,

vengono eseguite seguendo l'ordine contrario rispetto alle push.
Infine, viene realizzata un'istruzione ret per riportare il codice all'istruzione subito successiva la chiamata di funzione.

SCELTE PROGETTUALI

- La divisione gestisce solamente i casi in cui il dividendo è sempre positivo e il divisore può essere un numero negativo. Si lavora con registri a 32 bit e quindi la divisione sfrutta i registri EDX e EAX per rappresentare il dividendo e il divisore è rappresentato dall'operando EBX. Il valore del registro EDX viene impostato a zero per permettere il corretto funzionamento della
- I risultati di divisione e moltiplicazione possono essere codificati solo su 32 bit per evitare di dover controllare se il risultato sia passato a 64 bit.
- Nel caso della divisione, il risultato ha forma intera quindi il registro EDX, contenente il resto della divisione, non viene controllato.
- Gli operandi hanno un valore massimo di 32 bit.
- Nel file di input si può presentare una stringa che inizi con il simbolo di "a capo" ("\n"), iniziando quindi dalla seconda o dalle successive righe. Non deve però iniziare con caratteri non numerici o con spazi.
- Sono necessari almeno due operandi per far si che l'espressione sia calcolabile. Abbiamo quindi effettuato un controllo sul registro ECX, contenente il numero di operandi caricati sullo stack tramite push. Nel caso in cui le push siano minori di 2, la stringa viene considerata non valida.

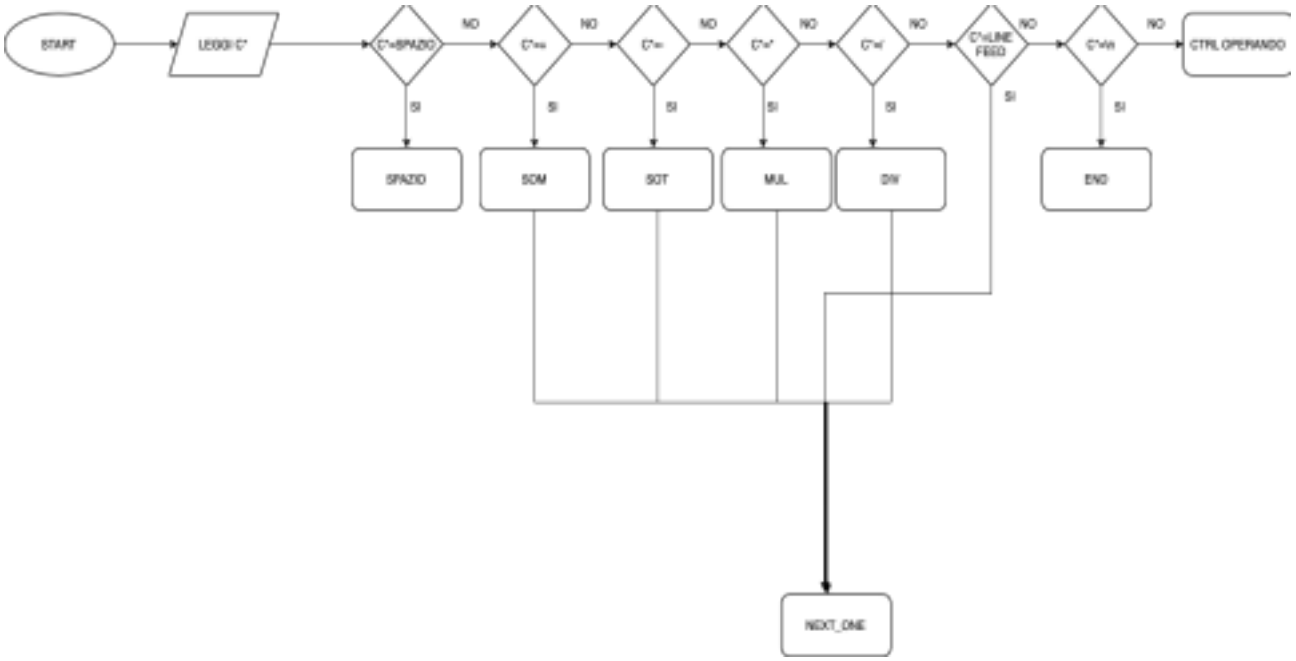
TEST DEI FILE DI INPUT

Abbiamo quindi testato da terminale le istanze di test delle stringhe sfruttando la printf all'interno del file main.c per poter avere una stampa del risultato anche a video.

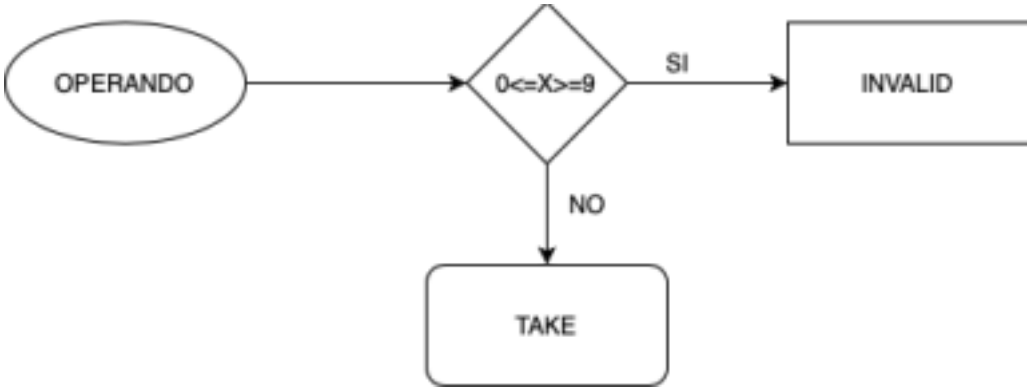
```
studente@debian:~/Scrivania/asm2/asm modificato/bin$ ls
in_1.txt  in_3.txt  in_5.txt   out_2.txt  out_4.txt  postfix
in_2.txt  in_4.txt  out_1.txt  out_3.txt  out_5.txt
studente@debian:~/Scrivania/asm2/asm modificato/bin$ ./postfix in_1.txt output1.txt
12
studente@debian:~/Scrivania/asm2/asm modificato/bin$ ./postfix in_2.txt output2.txt
30
studente@debian:~/Scrivania/asm2/asm modificato/bin$ ./postfix in_3.txt output3.txt
-16019741
studente@debian:~/Scrivania/asm2/asm modificato/bin$ ./postfix in_4.txt output4.txt
Invalid
studente@debian:~/Scrivania/asm2/asm modificato/bin$ ./postfix in_5.txt output5.txt
Invalid
```

DIAGRAMMA DI FLUSSO

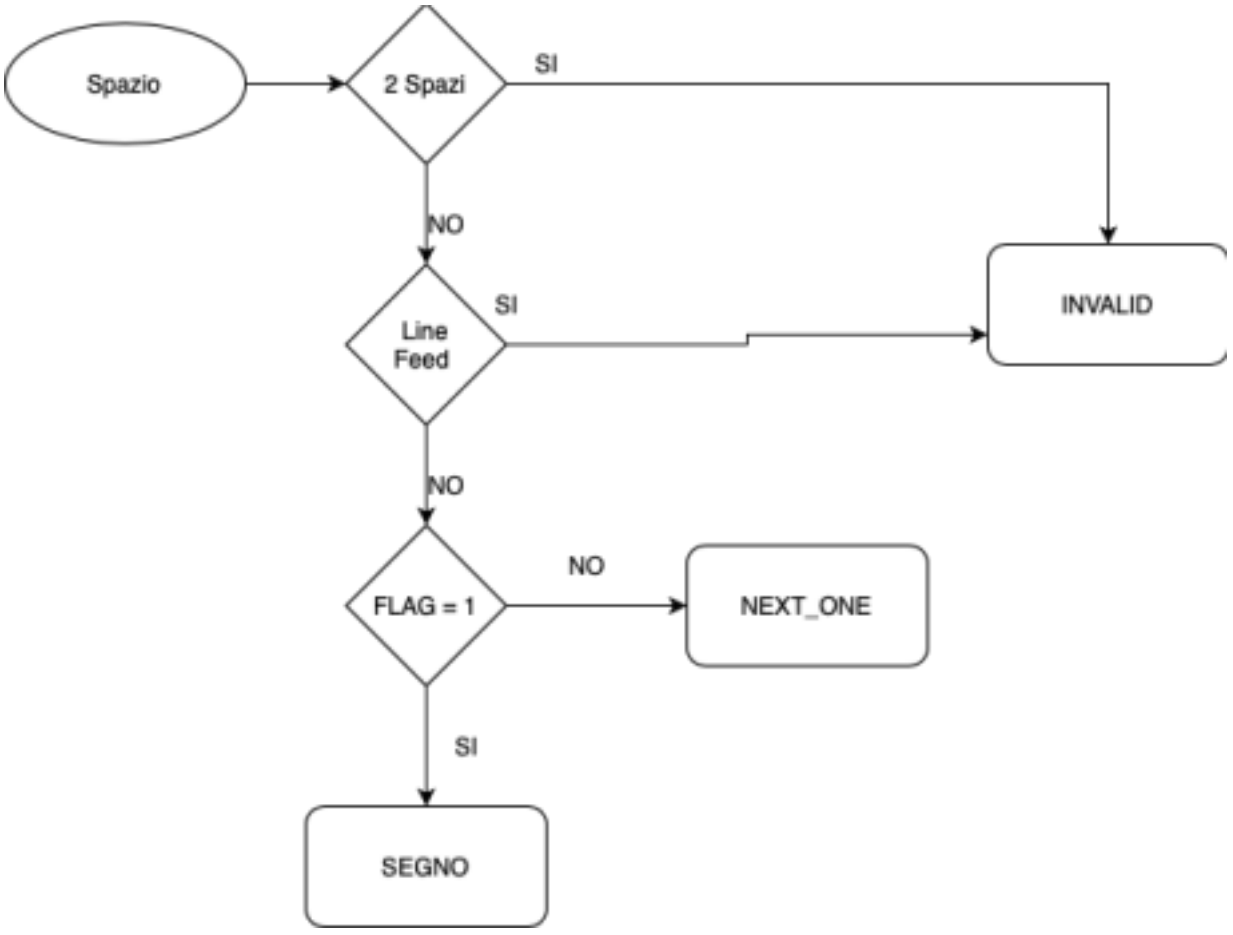
CONTROLLO:



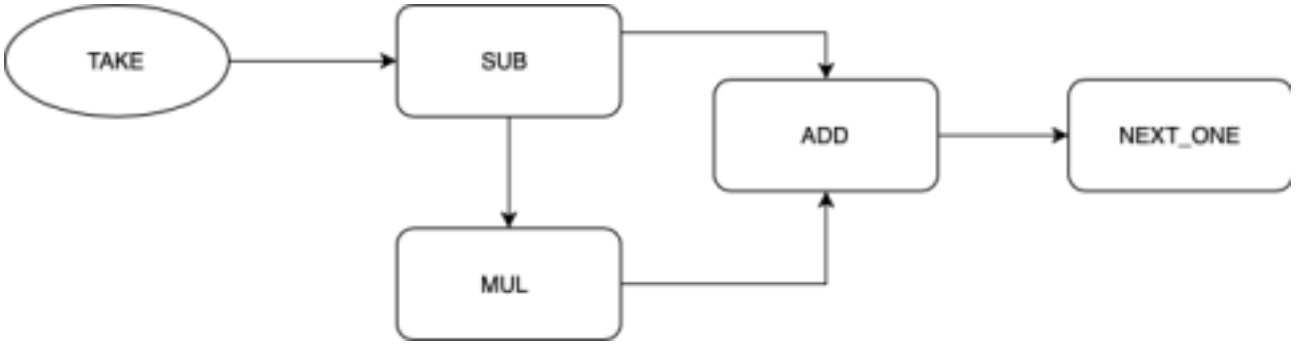
OPERANDO:


OPERANDO
0<=X>=9
SI
INVALID
NO
TAKE

SPAZIO:

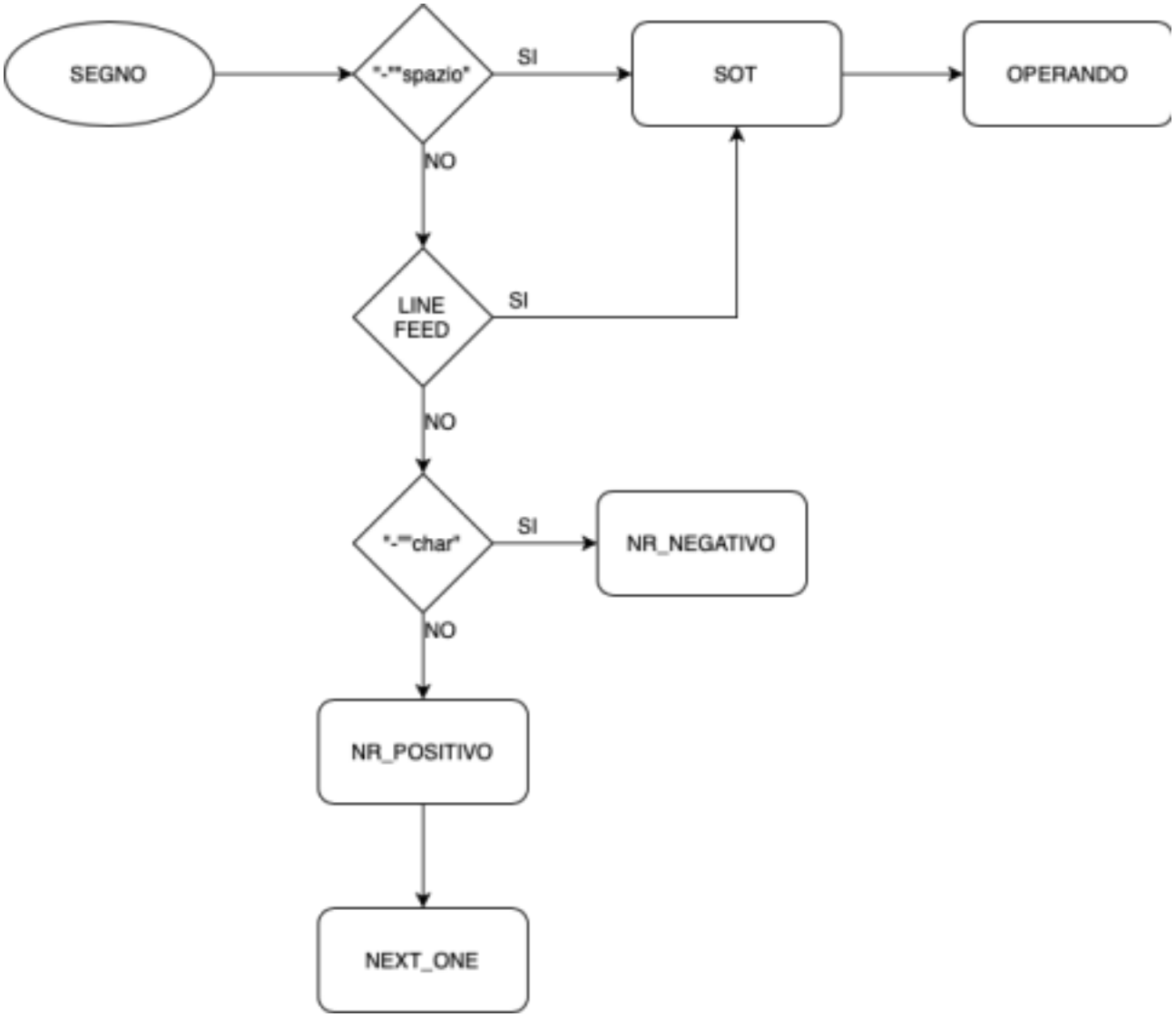

Spazio
2 Spazi
SI
NO
INVALID
Line Feed
SI
NO
FLAG = 1
NO
NEXT_ONE
SI
SEGNO

TAKE:

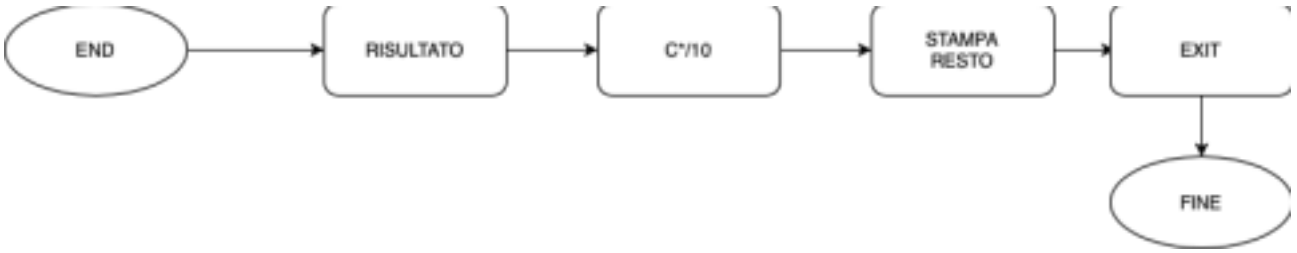

TAKE
SUB
ADD
NEXT_ONE
MUL

12

SEGNO:


SEGNO
"-""spazio"
SI
SOT
OPERANDO
NO
LINE FEED
SI
NO
"-""char"
SI
NR_NEGATIVO
NO
NR_POSITIVO
NEXT_ONE

END:


END
RISULTATO
C*/10
STAMPA
RESTO
EXIT
FINE