Progetto Assembly RISC-V per il Corso di Architetture degli Elaboratori – A.A. 2023/2024 – Valutatore di Espressioni

Versione 2 del documento, aggiornata il 25/03/2024.

Espressioni Aritmetiche

Un'espressione aritmetica è un insieme di numeri legati da segni di operazioni matematiche. Ad esempio, $23 + 4 * (5 + 3)$ è un'espressione aritmetica il cui risultato è 55. Per lo svolgimento del progetto, consideriamo solo **i numeri interi** e le quattro operazioni fondamentali:

- la somma ($+$);
- la sottrazione ($-$);
- la moltiplicazione ($*$);
- la divisione tra interi (\), che calcola il quoziente ignorando il resto:
  - $7/2 = 3, 16/3 = 5$.

Valutare un'espressione aritmetica significa calcolarne il suo risultato seguendo le regole di precedenza tra gli operatori: moltiplicazione e divisione hanno la precedenza su somma e sottrazione. Le regole di precedenza servono a rendere la valutazione "non ambigua" ($2 + 3 * 4$ è un'espressione ambigua perché può essere valutata sia come $5 * 4 = 20$ che come $2 + 12 = 14$). Un altro modo per rendere un'espressione non ambigua è usando le parantesi: ogni qual volta il risultato di un'espressione viene utilizzato all'interno di un'altra espressione, la sotto-espressione deve essere racchiusa tra parentesi. In questo modo, le espressioni sono sempre non ambigue e l'unica regola da seguire durante la valutazione è la seguente:

- le operazioni tra parantesi vengono svolte per prima;
  - in caso di più parentesi annidate, si parte da quelle più interne;

Esempi di espressioni non ambigue:

- $3 * \big(2 + \big(3 * (4 + 1)\big)\big) = 51$
- $3 * \big((2 + 3) * (4 + 1)\big) = 75$
- $(3 * 2) + \big((3 * 4) + 1\big) = 19$

Valutare Espressioni in RISC-V

Lo scopo del progetto di AE 23-24 è la realizzazione di un codice RISC-V che sia in grado di valutare un'espressione aritmetica non ambigua con parantesi ricevuta in input come stringa, dove ogni operando e risultato deve essere rappresentato come un intero con segno a 32 bit. Inoltre, il programma deve essere in grado di riconoscere eventuali errori nella stringa di input, eventuali overflow ed operazioni non ammesse (come dividere per 0) fermando l'esecuzione e mostrando all'utente un messaggio opportuno.

Parsing della Stringa di Input

Il programma deve accettare (e valutare) solamente le espressioni ottenute dal seguente linguaggio formale:

$$Expr := Op + Op \mid Op * Op \mid Op - Op \mid Op/\ Op$$
$$Op := (Expr) \mid Num$$
$$Num := Digit \mid DigitNum$$
$$Digit := 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Se la stringa non è valida, il programma deve terminare mostrando il messaggio "Espressione non valida!" senza stampare nessun risultato. Aggiungere dettagli al messaggio di errore (e.g. il motivo della non validità, posizione all'interno della stringa, ecc..) è opzionale ma può costituire merito addizionale.

Ecco alcuni esempi di stringhe non valide:

- tutte le stringhe contengono caratteri che non siano `{'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', '*', '/', '-', '(', ')', ' '}`, dove l'ultimo carattere rappresenta lo spazio. Eventuali spazi ripetuti o assenti non costituiscono errore;
- tutte le stringhe che non appartengono al linguaggio sopra definito:
  - "2 5", perché manca l'operazione;
  - "3 +∗ 2", perché ci sono due operazioni consecutive;
  - "2 + (6 ∗ 5", perché manca la parentesi di chiusura;
  - "2 + 6 ∗ 5", perché mancano le parantesi; un operando può essere o un numero o una espressione tra parantesi.
  - " − 5 + 6", perché manca il primo operando per la sottrazione
  - (2 + 5), perché le parantesi servono a specificare un operando

Gestione degli Errori Numerici

Il programma deve essere in grado di gestire eventuali errori numerici durante l'esecuzione, quali:

- **Overflow**, ovvero che il risultato di un'operazione non può essere rappresentato come intero con segno a 32 bit;
- **Divisione per 0**.

In caso di errore numerico, il programma deve terminare mostrando il messaggio "Overflow!" o "Divisone per 0!" a seconda dell'errore avvenuto senza stampare nessun risultato. Aggiungere dettagli al messaggio di errore (e.g. dettagli sull'operazione che ha generato l'errore) è opzionale ma può costituire merito addizionale.

Esempi

Qui di seguito sono riportati alcuni esempi di input:

- `"((1+2)*(3*2))-(1+(1024/3))"`
- `"((00000-2)*(1024+1024))  / 2"`
- `"1+(1+(1+(1+(1+(1+(1+0))))))"`
- `2*(2*(2*(2*(2*(2*(2*(2*(2*(2*(2*(1024*1024)))))))))))`
- `"2147483647+0"`
- `"2147483647+1"`
- `"(0-2147483647)-1"`
- `"(0-2147483647)-2"`

Struttura del Codice

Il codice assembly RISC-V del progetto deve contenere le seguenti funzioni:

- MAIN
- EVAL
- STRING_2_INT
- MUL
- DIV

Se ritenuto opportuno, altre funzioni possono essere definite.

Main

La funzione `main` è la funzione da cui inizia l'esecuzione e dovrà elaborare l'unico input utente del programma, ovvero l'espressione definita come una variabile `string` nel campo `.data` del codice RISCV, usando le altre funzioni opportunatamente definite. Il main si occuperà di stampare il risultato finale.

Eval

La funzione `eval` dovrà valutare l'espressione ricevuta in input e restituire in output il risultato dell'espressione. La funzione deve essere ricorsiva. Se ritenuto necessario, la funzione può ricevere ulteriori parametri e restituire ulteriori output.

String_2_int

La funzione `string_2_int` dovrà ricevere in input una stringa e restituire la rappresentazione del numero scritto nella stringa come intero con segno a 32 bit. Se ritenuto necessario, la funzione può ricevere ulteriori parametri e restituire ulteriori output.

Mul

La funzione `mul` riceve in input due numeri interi con segno a e b e restituisce il loro prodotto $a * b$. La funzione deve effettuare un numero di istruzioni lineare rispetto al numero di bit usati per rappresentare a e b. Se ritenuto necessario, la funzione può ricevere ulteriori parametri e restituire ulteriori output. L'implementazione dell'algoritmo di Booth costituisce merito addizionale.

Div

La funzione `div` riceve in input due numeri interi con segno a e b e restituisce il loro prodotto a / b. La funzione deve effettuare un numero di istruzioni lineare rispetto al numero di bit usati per rappresentare a e b. Se ritenuto necessario, la funzione può ricevere ulteriori parametri e restituire ulteriori output.

Relazione

La relazione deve essere strutturata come segue:

1. **Informazioni** su autore, indirizzo mail, matricola, data di consegna e versione RIPES usata.
2. **Descrizione** della soluzione adottata, contenente
    a. Una descrizione ad alto livello della soluzione adottata con discussione delle principali scelte implementative (ad esempio, definizioni di costanti in memoria o in opportuni registri)
    b. Una sotto-sezione per ogni funzione implementata contenente:
        i. Discussione di quali sono gli input e gli output della funzione
        ii. La descrizione dell'algoritmo implementato tramite pseudo-codice o flow-chart. Non bisogna ricopiare il codice assembly!
        iii. Descrizione della gestione dei registri e dello stack (cosa rappresenta ogni registro usato rispetto alla descrizione dell'algoritmo e cosa viene salvato nello stack)
3. **Test** per fornire evidenze del funzionamento del programma, anche in presenza di input errati o mal-formattati. Devono comparire **almeno** i test che usano gli input descritti nella parte di "Esempio"

Note e Modalità di Consegna

Note

- Seguire fedelmente tutte le specifiche dell'esercizio. In particolare, rendere il codice **modulare** utilizzando ove opportuno **chiamate a procedure** e **rispettando le convenzioni fra procedura chiamante/chiamata**. La modularità del codice ed il rispetto delle convenzioni saranno aspetti fondamentali per ottenere un'ottima valutazione del progetto
- Progetti con relazioni non strutturate come sopra specificato saranno valutati non sufficienti
- Commentare il codice in modo significativo (è poco utile commentare *addi s3, s3, 1* con "sommo uno ad s3"....).

Modalità di Esame

- Per sostenere l'esame è necessario consegnare preventivamente il codice e una relazione PDF sul progetto assegnato. Il progetto deve essere svolto dagli studenti singolarmente.
- Il codice consegnato deve essere funzionante sul simulatore RIPES in una versione uguale o maggiore alla 2.2.6, usato durante le lezioni.
- La scadenza esatta della consegna verrà resa nota di volta in volta, in base alle date dell'appello.
- Discussione e valutazione: la discussione degli elaborati avverrà contestualmente all'esame orale e **prevede anche domande su tutti gli argomenti di laboratorio trattati a lezione**.

Struttura della Consegna

La consegna dovrà consistere di un unico archivio contenente 3 componenti. L'archivio dovrà essere caricato sul sito moodle del corso di appello in appello, e dovrà contenere:

- Un unico file contenente il **codice** assembly
- la **relazione** in formato PDF
- un **breve video** (max 3 minuti) dove si registra lo schermo del dispositivo che avete utilizzato per l'implementazione durante l'esecuzione del programma, commentandone il funzionamento in base a 2-3 combinazioni di input diverse