Progetto Assembly RISC-V per il Corso di Architetture degli Elaboratori – A.A. 2024/2025 – Gestione di Liste Concatenate

---

Informazioni

- Autore: CHEN Xiong
- Email: xiong.chen@edu.unifi.it
- Matricola:7144793

- Data di consegna::17/5/2025
- Versione RIPES usata: 2.2.6

Descrizione del progetto:

Il programma è costruito attorno a una funzione principale chiamata **Main**, che riceve l'indirizzo di **listInput**, analizza i comandi e li delega alle subfunctions appropriate (**ADD**, **DEL**, **SORT**, **PRINT**, **REV**) in base al tipo di comando.
Oltre a queste funzioni principali, ci sono funzioni ausiliarie come **receive_Operator**, **clear_inputOperator**, **getWeight/cancelWeight**, **trim**, **next_operator** e **receive_parameter**, che supportano la logica principale del programma.
Il programma segue le convenzioni d'uso dei registri: i registri **'a'** fungono da variabili globali accessibili da tutte le funzioni, i registri **'s'** conservano valori locali a una funzione specifica e vengono inizializzati dopo il salvataggio nello stack, mentre i registri **'t'** vengono utilizzati per valori temporanei o caricamenti immediati.
Il programma include un rilevamento basilare degli errori: se **listInput** contiene più di 30 comandi, l'esecuzione viene interrotta e viene mostrato un messaggio di errore.

Definizioni delle costanti:

I seguenti nomi di comando sono predefiniti come stringhe:

- ADD: .string "ADD"
- DEL: .string "DEL"
- PRINT: .string "PRINT"
- SORT: .string "SORT"
- REV: .string "REV"

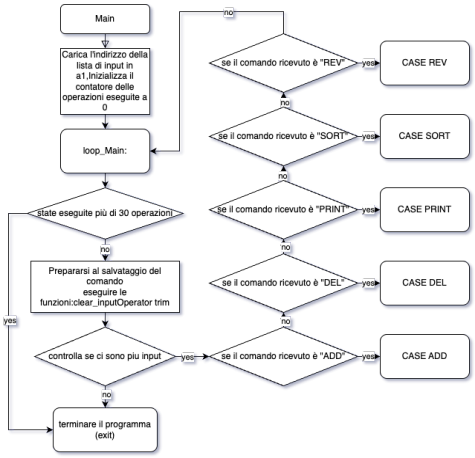
Definizione del Messaggio di Errore
Il messaggio di errore (una stringa) è definito nella RAM:

- MsgListInputError: .string "listInput non dovrà contenere più di 30 comandi!"

Altri indirizzi di memoria definiti:

- listInput: .string (utilizzato per memorizzare l'input della lista dei comandi)
- inputOperator: .string "AAAAAAAA" (ogni comando è limitato a 8 byte o meno)
- list:.string "" (spazio allocato per la lista concatenata)

Main:



La funzione **Main** inizializza l'indirizzo di **listInput** e configura i registri, incluso un contatore per i comandi eseguiti.
Successivamente entra nel ciclo **loop_Main**, dove legge ogni comando, lo confronta con la corrispondente subfunction (**ADD**, **DEL**, ecc.) e lo esegue.

Se un comando è invalido o è già stato eseguito, **Main** richiama **next_operator** per recuperare il comando successivo e ripete il processo di corrispondenza.

Input:

- Indirizzo di listInput
- Singolo comando

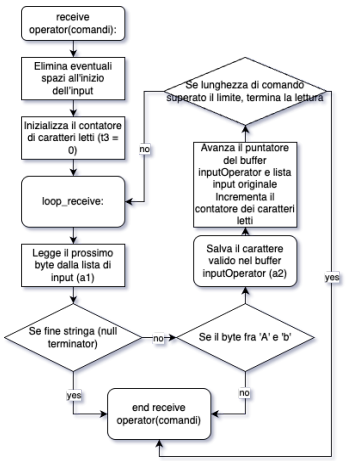
Output:

- lista concatenata modificata

Register & Stack Usage:

- a1: contiene l'indirizzo di **listInput**
- a2: memorizza ogni comando individuale
- a6: conta il numero di comandi eseguiti
- t registers: utilizzati per valori immediati, confronti e istruzioni di salto (branching)

receive_Operator:



Questa funzione legge **listInput** ed estrae un comando (operatore) separato dal carattere '~'.
Inizia richiamando la funzione **trim** per rimuovere eventuali spazi iniziali.
Successivamente, legge i byte da **listInput** uno alla volta. La lettura si interrompe quando viene incontrato un byte nullo oppure un carattere fuori dall'intervallo 'A'-'Z'.
La funzione poi ritorna alla **Main** per analizzare ed eseguire il comando.

Input:

- Indirizzo di **listInput**

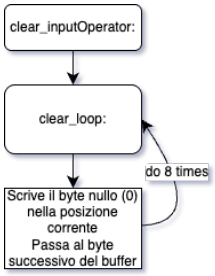
Output:

- Comando parsato

Register & Stack Usage:

- a1: contiene l'indirizzo di **listInput**
- a2: memorizza il comando parsato
- t3: conta il numero di caratteri letti
- Other t registers: utilizzati per il caricamento dei byte da **listInput** e per valori immediati

clear_inputOperator:



Questa funzione semplice azzera il contenuto di **inputOperator** dopo l'esecuzione di un comando, preparandolo per memorizzare il comando successivo.

Input:

- Indirizzo di inputOperator (utilizzato per memorizzare un singolo comando)

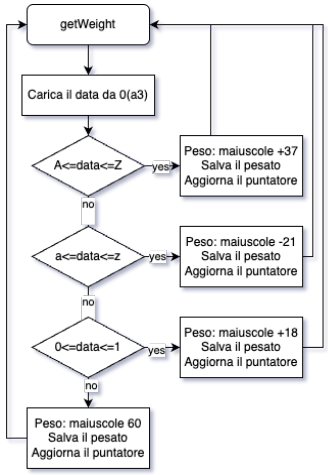
Output:

- Spazio in **inputOperator** azzerato

Register & Stack Usage:

- t0: contiene l'indirizzo di **inputOperator**
- t1: puntatore utilizzato per iterare su **inputOperator**
- t2: contatore per l'azzeramento dei byte

getWeight/cancelWeight:



Queste funzioni ausiliarie supportano la funzione **sort** ricorsiva modificando i pesi dei caratteri per influenzare l'ordine di ordinamento.
getWeight: aumenta il valore ASCII di alcuni tipi di caratteri (es. aggiunge 37 alle lettere maiuscole), in modo che vengano ordinati dopo le lettere minuscole.
cancelWeight: annulla la trasformazione, ripristinando i valori originali dei caratteri dopo l'ordinamento.
Le funzioni classificano i caratteri controllando i loro intervalli ASCII e applicano pesi differenti in base alla categoria.
Input:

- Indirizzo della lista concatenata

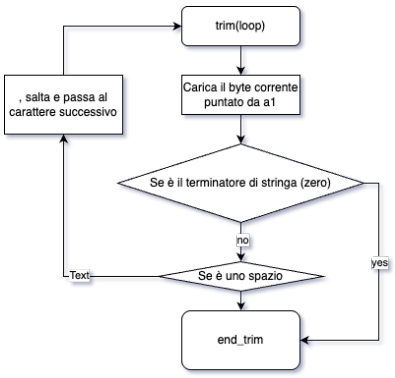
Output:

- Lista modificata dopo **getWeight** oppure ripristinata dopo **cancelWeight**

Register & Stack Usage:

- contengono valori immediati che definiscono i limiti degli intervalli ASCII
- t3: puntatore per attraversare la lista concatenata

trim:



Questa funzione salta gli spazi iniziali in **listInput**. Se il carattere corrente è uno spazio (' '), il puntatore avanza fino a trovare un carattere non spazio. Alla fine, restituisce l'indirizzo di quel primo carattere valido.
Input:

- Indirizzo di **listInput**

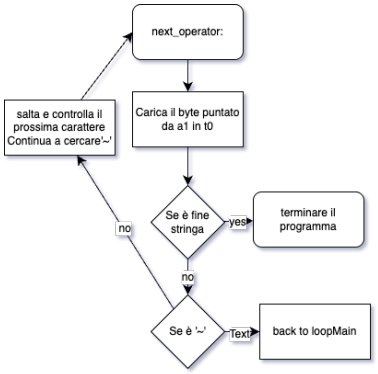
Output:

- Indirizzo del primo carattere non spazio

Register & Stack Usage:

- a3: puntatore a **listInput**
- t0: carica il byte attualmente puntato
- t1: contiene il codice ASCII dello spazio (' ', ovvero 0x20)

next_operator:



Questa funzione fa avanzare il puntatore al comando successivo in **listInput** scansionando i caratteri fino a trovare il delimitatore '~', che segnala l'inizio di un nuovo comando.
Input:

- Indirizzo di **listInput**

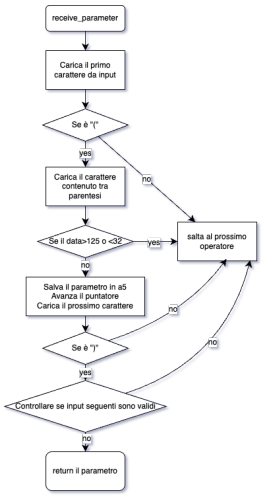
Output:

- Puntatore al comando successivo

Register & Stack Usage:

- a1: puntatore a **listInput**
- t0:carica il byte corrente durante la scansione
- t2:contiene il codice ASCII del carattere `'~'` (0x7E)

receive_parameter:



Questa funzione ausiliaria per **DEL** e **ADD** legge e valida i parametri che seguono un comando. Verifica se il parametro è racchiuso tra parentesi tonde e se rientra in un intervallo valido.

Input:

- Indirizzo di **listInput**

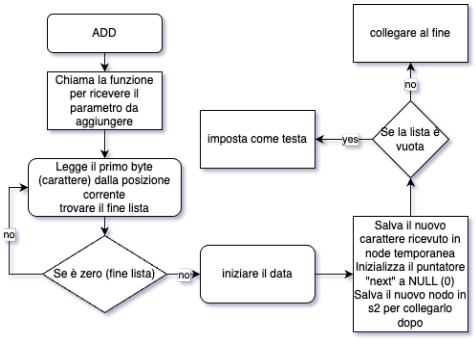
output:

- Parametro estratto dal comando

Register & Stack Usage:

- s0, s1: contengono i codici ASCII della parentesi sinistra '(' e destra ')'
- t0: carica i caratteri da **listInput**
- a5: memorizza il parametro estratto

ADD:



Questa funzione esegue il comando **ADD** ricevendo un parametro (tramite **receive_parameter**) e aggiungendolo alla lista concatenata. Se la lista concatenata è vuota, inizializza **a3** per puntare al nuovo nodo.

Input:

- Indirizzo di **listInput**
- Parametro da aggiungere
- Indirizzo della lista concatenata

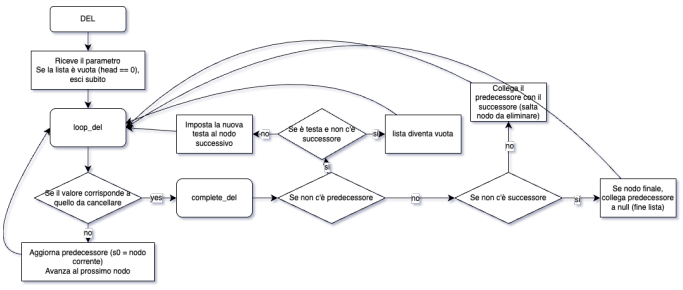
Output:

- lista concatenata aggiornata con il dato aggiunto

Register & Stack Usage:

- s0: indirizzo della lista concatenata
- s1: indirizzo della testa della lista concatenata
- t0:carica i byte dalla lista concatenata

DEL:



Questa funzione elimina tutti i nodi nella lista concatenata che corrispondono al parametro specificato.

Scorre la lista mantenendo traccia del nodo predecessore e del nodo successore. Quando trova un nodo corrispondente, lo rimuove e collega tra loro i nodi adiacenti, gestendo i casi con o senza predecessore e successore.

Il processo continua fino alla fine della lista.

Input:

- Indirizzo della lista concatenata
- Parametro da eliminare

Output:

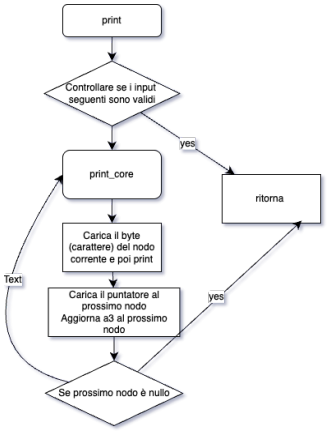
- lista concatenata aggiornata dopo la cancellazione

Register & Stack Usage:

- a5: valore del parametro

- s0: indirizzo del nodo predecessore
- s1: indirizzo del nodo corrente
- s2: indirizzo del nodo successore
- t registers: utilizzati per caricare i caratteri dalla lista concatenata

PRINT:



Questa funzione ricorsiva stampa i dati all'indirizzo puntato da **a3**, poi avanza **a3** al nodo successivo e richiama sé stessa.
L'esecuzione si interrompe quando **a3** è nulla (fine della lista).
Input:

- Indirizzo della lista concatenata

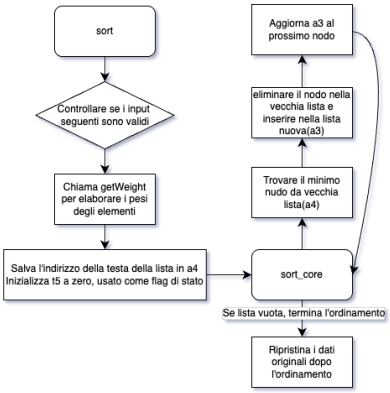
Output:

- Stampa tutti i dati presenti nella lista concatenata

Register & Stack Usage:

- a3:puntatore al nodo corrente
- t0: carica i dati dal nodo corrente
- t1: punta al nodo successivo

SORT:



Questa funzione ricorsiva implementa l'**insertion sort** su una lista concatenata.
Utilizza **a3** per costruire la nuova lista ordinata e **a4** per scansionare la lista vecchia.
La funzione trova ripetutamente il carattere minimo in **a4**, lo elimina da **a4** e lo aggiunge a **a3**.
Il processo continua finché **a4** non è vuota, quindi la funzione termina restituendo la lista ordinata.
Input:

- Indirizzo della lista concatenata

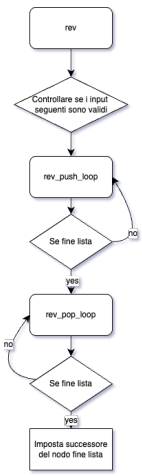
Output:

- lista concatenata ordinata

Register & Stack Usage:

- a3: indirizzo della nuova lista (ordinata)
- a4: indirizzo della vecchia lista (non ordinata)
- t5: flag che indica l'esistenza del nodo testa
- t registers: usati per caricare byte da **listInput** o valori immediati

REV:



Questa funzione inverte la lista concatenata utilizzando una pila (stack) e il suo principio LIFO. Durante la scansione della lista, spinge (push) ogni carattere sulla pila, quindi li estrae (pop) per ricostruire la lista in ordine inverso.
Input:

- Indirizzo della lista concatenata

Output:

- lista concatenata invertita

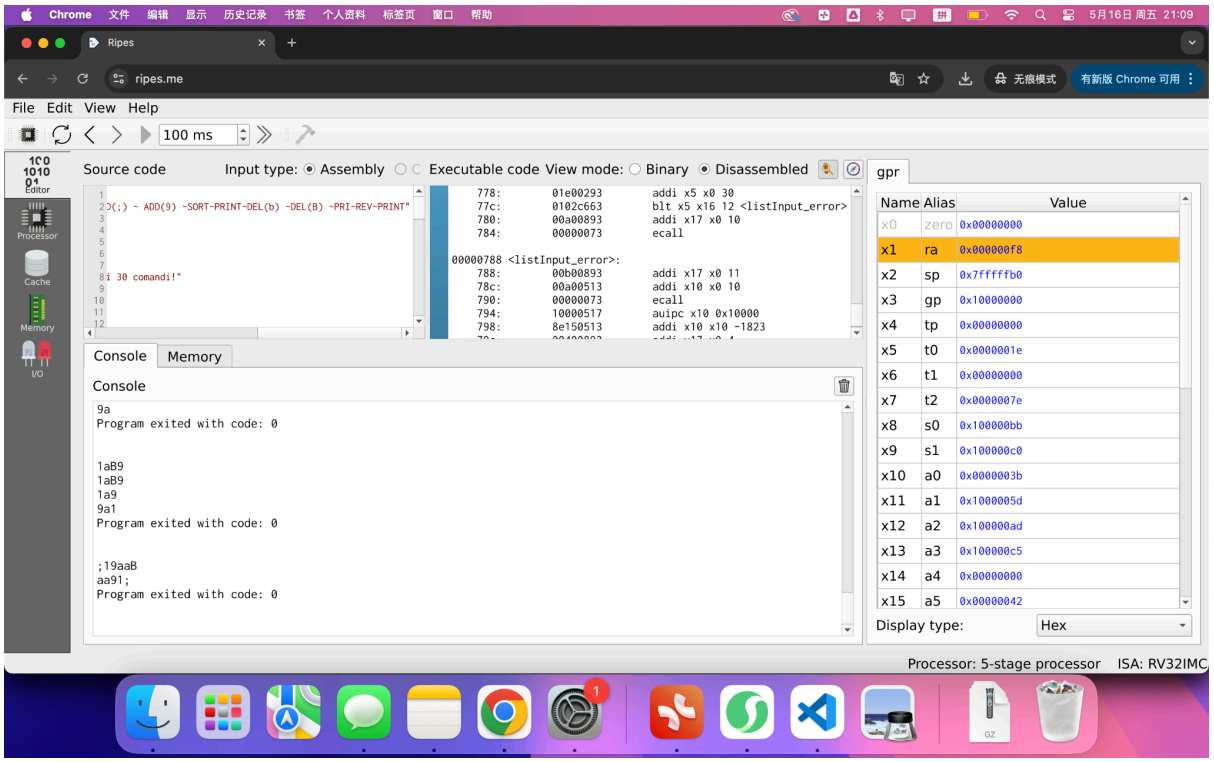
Register & Stack Usage:

- t registers: utilizzati come puntatori per attraversare la lista concatenata e gestire la pila

TEST:

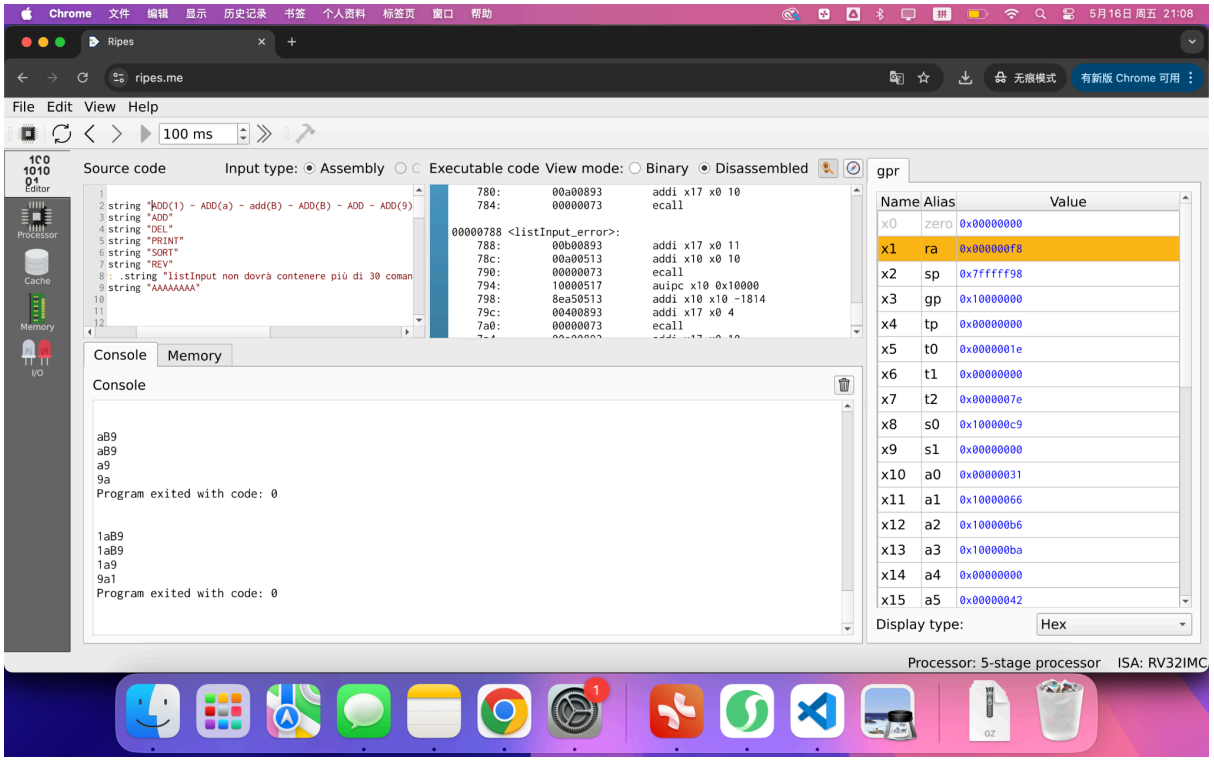
listInput = “ADD(1) ~ ADD(a) ~ ADD(a) ~ ADD(B) ~ ADD(;) ~ ADD(9) ~SORT~PRINT~DEL(b) ~DEL(B) ~PRI~REV~PRINT”

Comando Corrente	ADD(1)	ADD(a)	ADD(a)	ADD(B)	ADD(;)	ADD(9)	SORT	PRINT	DEL(b)	DEL(B)	PRI	REV	PRINT
Elementi in Lista	1	1a	1aa	1aaB	1aaB;	1aaB;9	;19aaB	;19aaB	;19aaB	;19aa	;19aa	aa91;	aa91;



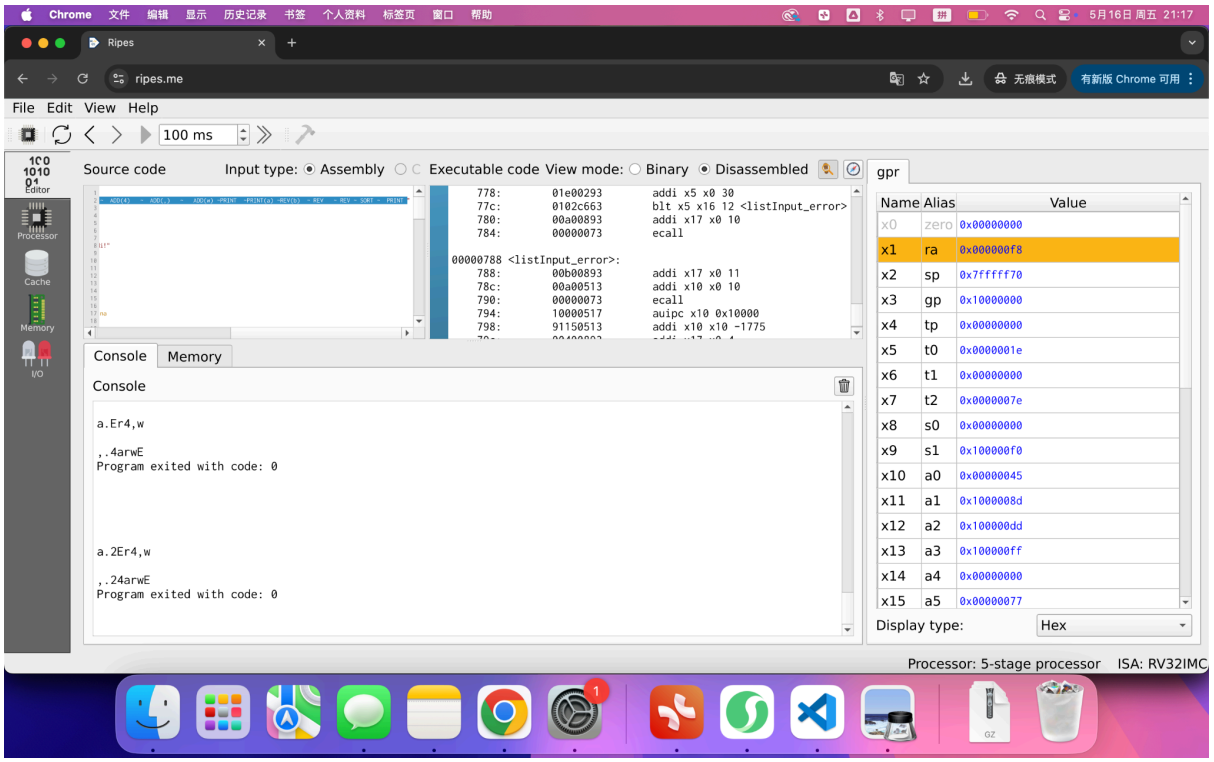
listInput = “ADD(1) ~ ADD(a) ~ add(B) ~ ADD(B) ~ ADD ~ ADD(9) ~PRINT~SORT(a)~PRINT~DEL(bb) ~DEL(B) ~PRINT~REV~PRINT”

Comando Corrente	ADD(1)	ADD(a)	add(B)	ADD(B)	ADD	ADD(9)	PRINT	SORT(a)	PRINT	DEL(bb)	DEL(B)	PRINT	REV	PRINT
Elementi in Lista	1	1a	1a	1aB	1aB	1aB9	1aB9	1aB9	1aB9	1aB9	1a9	1a9	9a1	9a1

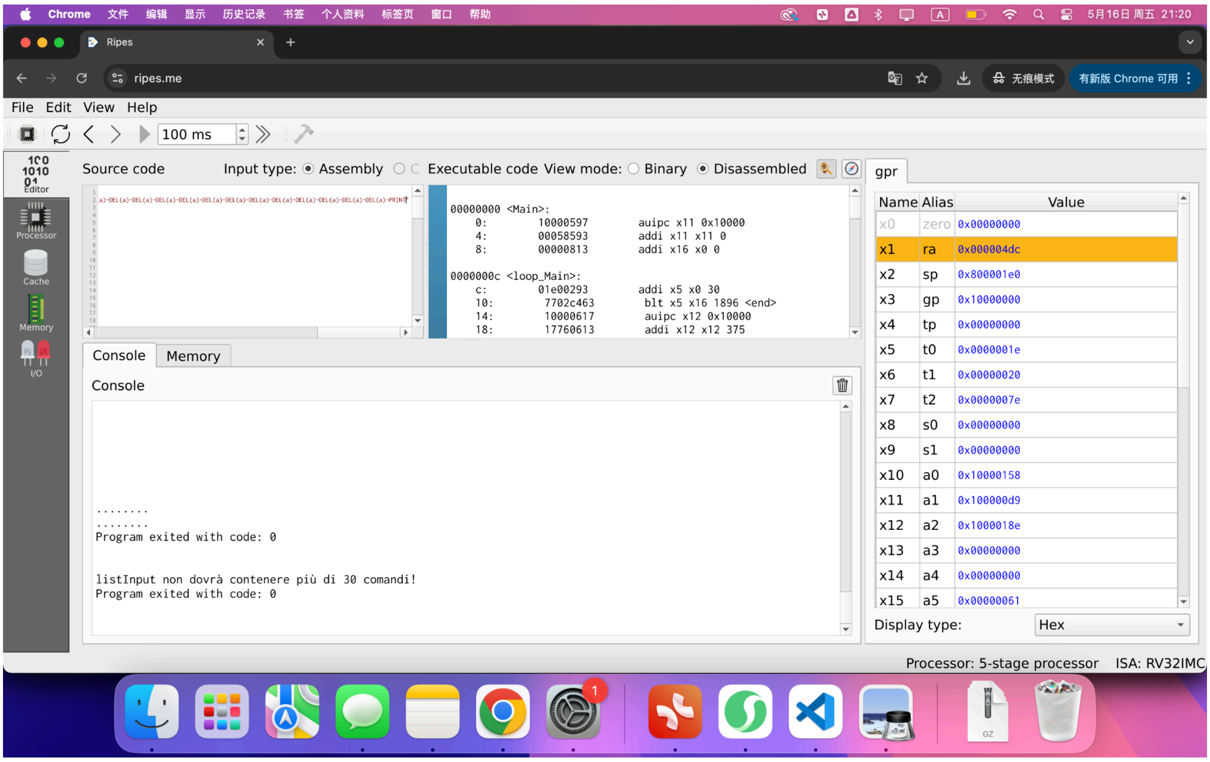


listInput = “ADD(a) ~ ADD(.) ~ ADD(2) ~ ADD(E) ~ ADD(r) ~ ADD(4) ~ ADD(,) ~ ADD(w) ~PRINT ~PRINT(a) ~REV(b) ~ REV ~ REV ~ SORT ~ PRINT”

ComandoCorrente	ADD(a)	ADD(.)	ADD(2)	ADD(E)	ADD(r)	ADD(4)	ADD(,)	ADD(w)	PRINT	PRINT(a)	REV(b)	REV	REV	SORT	PRINT
Elementi in Lista	a	a.	a.2	a.2E	a.2Er	a.2Er4	a.2Er4,	a.2Er4,w	a.2Er4,w	a.2Er4,w	a.2Er4,w	w,4rE2.a	w,4rE2.a	a.2Er4,w	,.24arwE

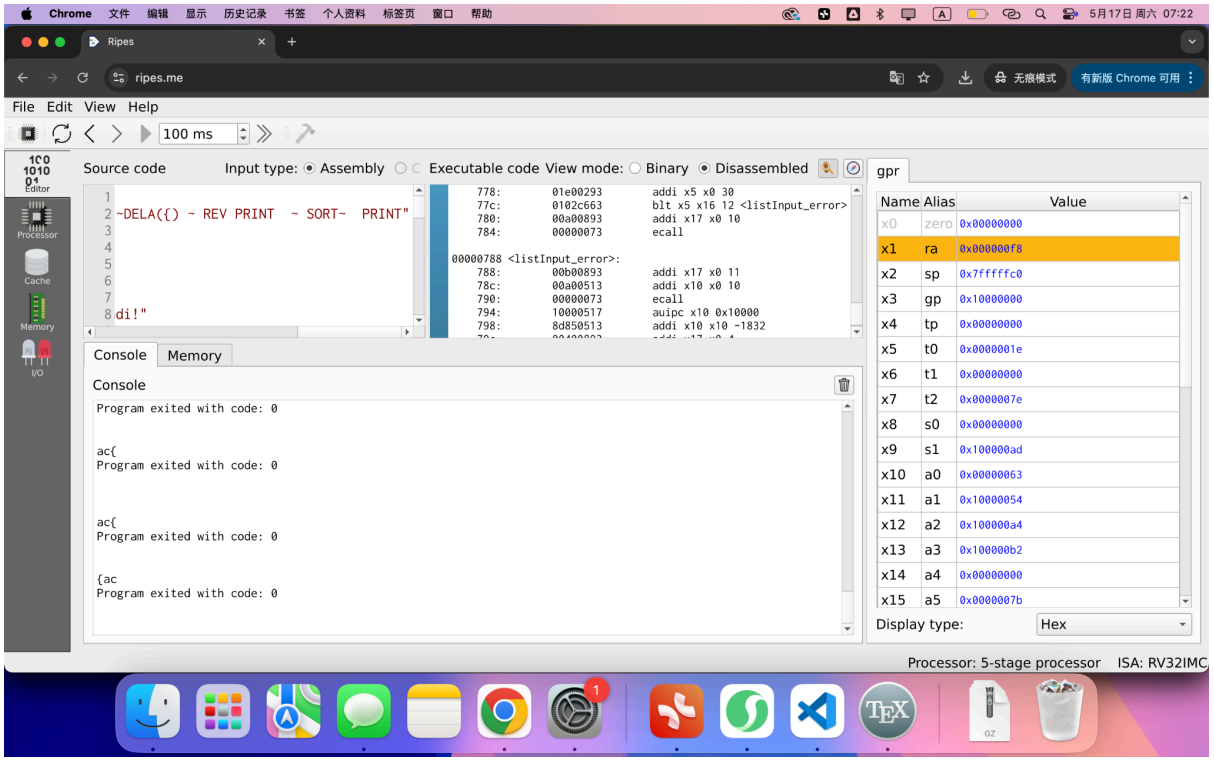


listInput =
“DEL(a)~PRINT”
Expected to display ErrorMessage



listInput = “ADD(a) ~ add(a) ~ ~ ADD(c)   ~ ADDS(v) ~ ADD({) ~DELA({) ~ REV PRINT ~ SORT~ PRINT”

Comando Corrente	**ADD(a)**	**add(a)**		**ADD(c)**	**ADD(v)**	**ADD({)**	**DELA({)**	**REV PRINT**	**SORT**	**PRINT**
Elementi in Lista	**a**	**a**	**a**	**ac**	**acv**	**ac{**	**ac{**	**ac{**	**{ac**	**{ac**


Chrome 文件 编辑 显示 历史记录 书签 个人资料 标签页 窗口 帮助
5月17日 周六 07:22
Ripes
ripes.me
无痕模式
有新版 Chrome 可用
File Edit View Help
100 ms
Editor
Processor
Cache
Memory
I/O
Source code
Input type: Assembly C
Executable code
View mode: Binary Disassembled
~DELA({) ~ REV PRINT ~ SORT~ PRINT"
di!"
778: 01e00293 addi x5 x0 30
77c: 0102c663 blt x5 x16 12 <listInput_error>
780: 00a00893 addi x17 x0 10
784: 00000073 ecall
00000788 <listInput_error>:
788: 00b00893 addi x17 x0 11
78c: 00a00513 addi x10 x0 10
790: 00000073 ecall
794: 10000517 auipc x10 0x10000
798: 8d850513 addi x10 x10 -1832
Console
Memory
Console
Program exited with code: 0
ac{
Program exited with code: 0
ac{
Program exited with code: 0
{ac
Program exited with code: 0
gpr
Name Alias Value
x0 zero 0x00000000
x1 ra 0x000000f8
x2 sp 0x7fffffc0
x3 gp 0x10000000
x4 tp 0x00000000
x5 t0 0x0000001e
x6 t1 0x00000000
x7 t2 0x0000007e
x8 s0 0x00000000
x9 s1 0x100000ad
x10 a0 0x00000063
x11 a1 0x10000054
x12 a2 0x100000a4
x13 a3 0x100000b2
x14 a4 0x00000000
x15 a5 0x0000007b
Display type: Hex
Processor: 5-stage processor ISA: RV32IMC