Gestione Linkedlist semplice Risc V

Giulio Carapelli — giulio.carapelli@edu.unifi.it — Matricola: 7169515

1. Introduzione

Specifiche con cui è stato realizzato il progetto:

- Versione Ripes: 2.2.6
- Processore scelto: Risc-V 32 bit single-cycle processor

2. Main

Il main nel dettaglio si occupa della scansione della stringa dei comandi, di identificare il fine comando rappresentato dalla tilde, scartare i comandi mal formattati o eseguire chiamando le relative funzioni di handle di quelli corretti.

Per fare ciò il programma usa un **parse_loop** che scorre la stringa di comandi, esce dal loop se scorrendo il **listInput** si raggiunge il terminatore di stringa. Nello scorrere la lista, viene letto carattere per carattere dal **listInput** e caricati all'interno di un **buffer** (definito nel .data come word) fino al raggiungimento della tilde. Una volta raggiunta la tilde, il programma salta all'analisi del buffer per identificare il comando ed eseguirlo o scartalo in caso di mal formattazione. Il parsing del buffer ignora tutti gli spazi prima del comando saltandoli. Per capire di quale comando si tratta, una volta trovato un carattere diverso dallo spazio controlla che sia iniziale di uno dei comandi, se lo è viene chiamato il check del comando specifico, in caso contrario il comando è invalido e il programma torna a scorrere la stringa dei comandi. Il check di ogni specifico comando si occupa di controllare se carattere per carattere il comando è ben formattato, se così fosse viene chiamata la procedura di handle del comando identificato. In caso il comando non superi la fase di check, il comando è invalido quindi il programma ritorna al **parse_loop** per continuare a scorrere la stringa dei comandi.

3. Funzioni

3.1 Add

La funzione di Add si occupa di creare in uno spazio libero della memoria un nodo con all'interno il carattere che viene passato tra parentesi e poi aggiungerlo alla lista. I casi dell'aggiunta sono 2 e dipendono dalla lista al momento dell'inserimento. Se la lista è vuota l'inserimento avviene in testa, se la lista è già popolata da altri elementi va scorsa fino a trovare l'ultimo elemento ed effettuare "logicamente" lì l'inserimento.

Il programma per gestire una chiamata ben formattata alla funzione ADD usa la procedura **handle add** a cui viene passato attraverso il registro **a1** il carattere che si desidera inserire. La procedura per prima cosa chiama **find_free_space**, una procedura che si occupa di ritornare un indirizzo di memoria con 5 byte contigui liberi. L'indirizzo viene trovato da **find_free_space** scorrendo la memoria verso indirizzi crescenti a partire da un indirizzo fisso (0x00000f54) ed esaminando se i 5 byte dopo quell'indirizzo sono vuoti. Se viene trovato un indirizzo che rispecchia questa specifica la procedura lo ritorna attraverso il registro **a1** ad **handle add** in caso l'indirizzo non venga trovato la procedura restituirà 0 come indirizzo, a quel punto **handle add** scriverà un messaggio di errore ritornando al main. Nel nuovo indirizzo ottenuto **handle add** scrivere l'elemento che gli è stato passato in **a1** e mette a 0 il puntatore al prossimo elemento. Una volta fatto ciò la procedura controlla l'indirizzo di testa per capire se la lista è vuota o meno. Se la lista è vuota modifica l'indirizzo di testa con quello del nodo appena creato. In caso contrario, il programma scorre la lista fino all'ultimo nodo (identificato dal puntatore al successivo uguale a 0), una volta li cambia il puntatore di tale nodo con quello dell'elemento appena creato facendo così diventare il nuovo nodo l'ultimo della lista.

3.2 Dell

La funzione di Dell si occupa di cancellare tutte le occorrenze di un carattere dalla lista. I casi di cancellamento sono due: Se la lista è vuota la funzione non fa nulla se la lista ha almeno un elemento la scorre cercando l'elemento desiderano eliminandolo e mantenendo l'integrità della lista.

La procedura che si occupa della cancellazione di un nodo nel programma è **handle_dell** alla quale viene passato attraverso il registro a1 carattere che si desidera eliminare. La procedura inizia a scorrere la lista dall'indirizzo di testa, il loop si interrompe se l'indirizzo del nodo da controllare è 0. Questo comportamento permette di gestire immediatamente il caso in cui la lista sia vuota e quindi la testa sia impostata a 0x0. La procedura si occupa dunque di scorrere la lista, ogni volta che trova un nodo da eliminare ricollega i puntatori tra l'elemento prima e l'elemento dopo il nodo target e poi procede a mettere la memoria all'indirizzo del nodo da eliminare tutta a 0 per essere ottenibile da **find_free_space**.

3.3 Print

La funzione di print si occupa di scorrere la lista e scrivere in console elemento per elemento.

La procedura che si occupa della funzione di print è **handle_print**. La procedura inizia caricando l'indirizzo di testa. La procedura termina quando l'indirizzo del nodo da stampare è 0x0. Una volta caricata la testa carica l'indirizzo al nodo stampa il carattere tramite la ecall e passa al prossimo.

3.4 Reverse

La funzione reverse si occupa di invertire logicamente l'ordine degli elementi all'interno della lista. Lo scambio deve avvenire attraverso l'ausilio dello stack. Per rispettare lo specifico la procedura scorre la lista caricando carattere per carattere all'interno della stack. Una volta fatto ciò la lista viene

attraversata nuovamente effettuando per ogni nodo un pop dalla stack e poi sostituendo l'elemento ottenuto nel nodo che si sta attraversando.

La procedura che si occupa di questa funzione è **handle_rev** che per prima cosa carica l'indirizzo head per iniziare a scorrere la lista. Per ogni nodo decrementando lo stack pointer di una word e carica l'elemento corrente nella stack finché non si raggiunge il terminatore della lista. Una volta fatto ciò ricarica nuovamente il puntatore all head della lista per scorrere la lista. Questa volta la procedura effettua un pop dallo stack poi incrementa lo stack pointer di una word.

3.5 Sort

La funzione di sort si occupa di ordinare logicamente la lista con le regole fornite nella specifica.

La funzione che nel programma si occupa del sort è chiamata **handle_sort** la quale implementa un ordinamento bubble sort di tipo ricorsivo. Per prima cosa la procedura salva nella stack il return address e l'indirizzo originale alla testa, setta poi il contenuto di t6 a 0, userà questo registro come flag per registrare se è stato effettuato uno swap tra elementi, in caso contrario questo servirà a effettuare un'uscita anticipata o capire se la lista è stata completamente ordinata. Questa implementazione purtroppo nel caso peggiore richiede di effettuare un ciclo in più non effettuando alcuna modifica. La parte principale della procedura è il ciclo **bubble_pass** il quale si occupa inizialmente di caricare nei registri temporanei t2 e t3 i valori contenuti dal nodo puntato e quello successivo, dopodichè di classificare a quale gruppo di caratteri, "char1" e "char2" appartengono. La classificazione avviene tramite confronti diretti con vari intervalli della tabella ascii. Una volta capita la categoria setta in t4 ad un numero tra 0 e 3 per il primo carattere e a un numero tra 0 a 3 il contenuto di t5 per il secondo carattere. I numeri contenuti nei registri t4, t5 rappresentano in ordine a quale categoria il carattere appartiene (0 extra, 1 numeri, 2 lettere minuscole e 3 lettere maiuscole). In base al contenuto di t4, t5 si sceglie se effettuare o meno lo scambio confrontandoli. Se lo scambio avviene t6 viene messo ad 1 ed il sort continuerà. Una volta fatto ciò la procedura viene richiamata ricorsivamente passando di nuovo a se stessa in a0 l'indirizzo della testa che era stato salvato nella stack. Una volta finita la procedura ricarica il return address e ritorna al check che la aveva chiamata.