Progetto Assembly RISC-V per il Corso di Architetture degli Elaboratori – A.A. 2020/2021 – Gestione di Liste Concatenate

Versione 3 del documento, aggiornata il 28/4/2021. Modifiche rispetto alle precedenti versioni:

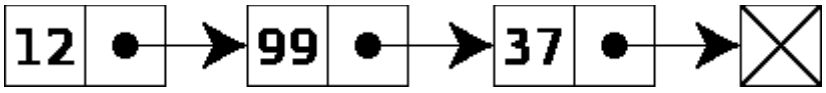
- Precisazione su comandi non ammissibili in listInput (sezione "Controllo Input")

Liste Concatenate

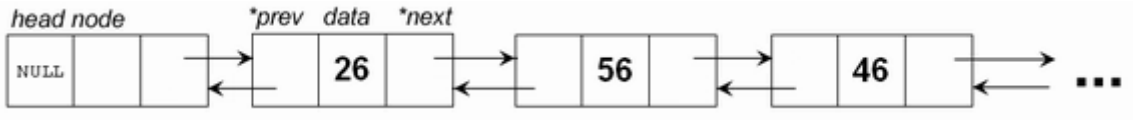
Una lista concatenata (o Linked List) è una struttura dati dinamica che consiste di una sequenza di nodi, ognuno contenente campi di dati arbitrari ed uno o due riferimenti ("link") che puntano al nodo successivo e/o precedente. Le liste concatenate permettono l'inserzione e la rimozione di nodi in ogni punto della lista in tempo costante, ma – diversamente dagli array – non permettono l'accesso casuale, solo quello sequenziale.

Esistono diversi tipi di liste concatenate (immagini da wikipedia):

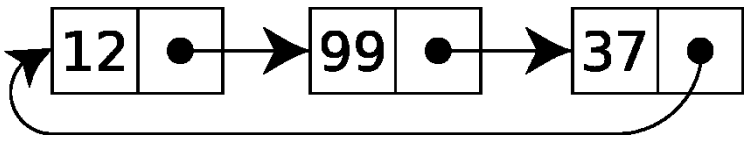
- liste concatenate semplici,



- liste concatenate doppie e



- liste circolari



Liste Concatenate Doppie in RISC-V

Il progetto di AE 20-21 mira all'implementazione di un codice RISC-V che gestisce alcune delle operazioni fondamentali per una lista concatenata doppia, tra le quali:

- ADD - Inserimento di un elemento
- DEL - Rimozione di un elemento
- PRINT - Stampa della lista

- SORT - Ordinamento della lista
- REV – Inversione degli elementi della lista

Ogni elemento della lista si deve supporre di dimensione 9 byte, così suddivisi:

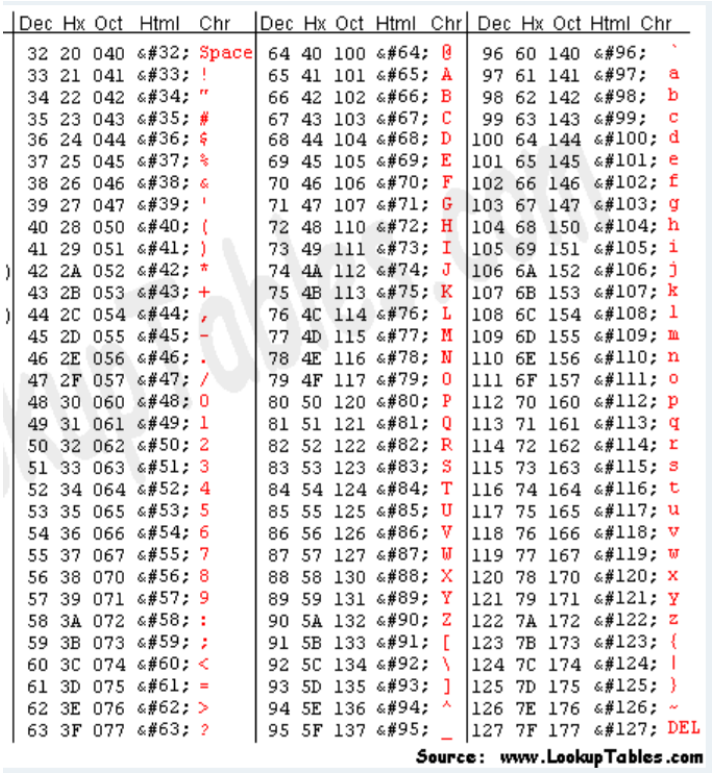
- PBACK (Byte 0-3): puntatore all'elemento precedente, o contenente 0xFFFFFFFF se primo elemento della lista
- DATA(Byte 4): contiene l'informazione
- PAHEAD (Byte 5-8): puntatore all'elemento successivo, o contenente 0xFFFFFFFF se ultimo elemento della lista

Si noti come i puntatori alla memoria abbiamo dimensione 32bit, ovvero una word RISC-V. Inoltre, si assume che il byte di informazione contenuto in ciascun elemento della lista rappresenti un carattere ASCII. Sui caratteri ascii viene stabilito il seguente ordinamento (transitivo):

- una lettera maiuscola (ASCII da 65 a 90 compresi) viene sempre ritenuta maggiore di una minuscola
- una lettera minuscola (ASCII da 97 a 122 compresi) viene sempre ritenuta maggiore di un numero
- un numero (ASCII da 48 a 57 compresi) viene sempre ritenuto maggiore di un carattere extra che non sia lettera o numero
- non si considerano accettabili caratteri extra con codice ASCII minore di 32.

Inoltre, all'interno di ogni categoria vige l'ordinamento dato dal codice ASCII. Per esempio, date due lettere maiuscole x e x', x < x' se e solo se ASCII(x) < ASCII(x'). Lo stesso vale per le lettere minuscole, per i numeri e per i caratteri extra. Ad esempio, la sequenza a.2Er4,w si ordinerà come ,.24arwE

Sotto si riassumono i **codici ASCII accettabili (da 32 a 125 compresi)** come informazioni negli elementi della lista, per questo progetto.



Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
32	20	040		Space	64	40	100	@	@	96	60	140	`	`
33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Dettaglio del Main

Lo studente dovrà implementare un codice assembly RISC-V strutturato con un main e relative funzioni, che dovranno essere definite al bisogno.

Il main dovrà elaborare l'unico input utente del programma, dichiarato come una variabile string *listInput* nel campo .data del codice RISCV. Tale *listInput* dovrà contenere una serie di comandi separati da ~ (ASCII 126), dove ogni comando contiene una operazione ed eventualmente dei parametri. listInput non dovrà contenere più di 30 comandi.

Nello specifico:

- ADD(*char*): crea un nuovo elemento della lista che contiene come informazione DATA=*char*, e viene aggiunto in coda alla lista esistente
- DEL(char): ricerca l'elemento con DATA=char esistente nella lista e, se esistente, lo elimina. Nel caso in cui più elementi con DATA=char siano presenti nella lista, rimuove solo il primo.
- PRINT: stampa tutti i DATA degli elementi della lista, in ordine di apparizione
- SORT: ordinamento crescente della lista
- REV: inverte gli elementi della lista

Controllo Input

Si dovrà quindi inserire un controllo degli input e della formattazione dei singoli comandi. Questo deve essere definito dallo studente e dettagliato nella relazione.

- Nel caso delle operazioni ADD e DEL si suppone di avere uno ed uno solo carattere tra parentesi; nel caso in cui compaiano zero o più di un carattere tra parentesi, l'operazione si considera mal formattata e da scartare.
- I caratteri dei comandi devono essere consecutivi: sarà ammissibile il comando "SORT" ma non il "SO RT". Allo stesso modo, è ammissibile "ADD(d)" ma non "AD D(d)" o "ADD (d)"
- Spazi vicini alle ~ sono ammessi e devono essere tllerati dal programma.
- Il comando deve essere espresso con lettere maiuscole. "PRINT" è ammissibile, "print" non lo è.

Esempi di Input ed Esecuzione

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

Comando Corrente	ADD(1)	ADD(a)	ADD(a)	ADD(B)	ADD(;)	ADD(9)	PRINT	SORT	PRINT	DEL(b)	DEL(B)	PRI	REV	PRINT
Elementi in Lista	1	2	3	4	5	6	6	6	6	6	5	5	5	5
PRINT	1	1a	1aa	1aaB	1aaB;	1aaB;9	1aaB;9	;19aaB	;19aaB	;19aaB	;19aa	;19aa	aa91;	aa91;

listInput = "ADD(1) ~ ADD(a) ~ ADD() ~ ADD(B) ~ ADD ~ ADD(9) ~PRINT~SORT(a)~PRINT~DEL(bb) ~DEL(B) ~PRINT~REV~PRINT"

Comando Corrente	ADD(1)	ADD(a)	ADD()	ADD(B)	ADD	ADD(9)	PRINT	SORT(a)	PRINT	DEL(bb)	DEL(B)	PRINT	REV	PRINT
Elementi in Lista	1	2	2	3	3	4	4	4	4	4	3	3	3	3
PRINT	1	1a	1a	1aB	1aB	1aB9	1aB9	1aB9	1aB9	1aB9	1a9	1a9	9a1	9a1

Dettaglio delle Singole Operazioni

Sotto si fornisce il dettaglio delle singole funzioni da implementare per il progetto. Le funzioni ADD e SORT richiederanno più lavoro, mentre le altre si riveleranno facilmente implementabili.

ADD - Inserimento di un Elemento

L'inserimento di un nuovo elemento nella lista comporta principalmente quattro operazioni:

1. identificazione di una porzione casuale di memoria da 9 byte che non si sovrappone con dati esistenti
2. la memorizzazione del nuovo elemento nella nuova area di memoria, con PAHEAD = 0xFFFFFFFF
3. l'aggiornamento del puntatore PBACK del nuovo elemento, che deve puntare all'elemento in coda alla lista
4. l'aggiornamento del puntatore PAHEAD dell'elemento in coda alla lista, che non deve più essere 0xFFFFFFFF (l'elemento non è più l'ultimo, ne stiamo aggiungendo uno nuovo), ma deve puntare all'elemento che stiamo inserendo

Generazione Pseudocasuale di Indirizzo: LFSR

Quando si inserisce un nuovo elemento in una lista a puntatori, tale elemento deve prima essere memorizzato in una posizione casuale della memoria (vedi punto 1 sopra). Il concetto di casualità e di ottenimento di numeri cauali (RNG – Random Number Generation) è un concetto complesso nell'informatica: spesso risulta difficile ottenere una casualità completa[1], e quindi si sceglie di approssimarla tramite Pseudo-RNG.

Uno dei metodi più semplici per derivare uno Pseudo-RNG in Assembly è l'utilizzo di un LFSR (Linear-Feedback Shift Register). Molto brevemente, un LFSR a n bit è una piccola rete combinatoria che a partire da un numero di partenza espresso su n bit (seed) genera un altro numero sempre espresso su n bit. Questo numero è ottenuto dal precedente nel modo seguente:

- si calcola il bit più significativo del nuovo numero calcolando un polinomio
- gli n-1 bit meno significativi si ottengono shiftando a destra il numero precedente

Questi numeri non sono veramente casuali, ma appartengono a delle successioni che hanno un periodo lungo (es. i numeri si ripetono ogni 10000 generazioni) e quindi vengono definiti numeri pseudo-casuali (Preudo-RNG).

PseudoCodice per LFSR a 16 Bit

Un possibile LFSR a 16 bit viene descritto da Wikipedia (https://en.wikipedia.org/wiki/Linear-feedback_shift_register). Tale LFSR usa il polinomio x^16 + x^14 + x^13 + x^11 + 1; tale polinomio viene calcolato tramite shifts (a destra) successivi, che vengono poi messi in XOR. Si devono calcolare tanti shifts quanti sono i termini x^k (k>0) non nulli del polinomio, shiftando ogni volta di 16 – k bit. Per il polinomio sopra si avranno quindi 4 shifts a destra: il primo di 0 (=16-16) bit, il secondo di 2 (=16-14) bit, il terzo di 3 (=16-13) bit, ed il quarto di 5 (=16-11) bit. Tale LFSR si può riassumere come segue in uno pseudo-linguaggio:

[1] https://engineering.mit.edu/engage/ask-an-engineer/can-a-computer-generate-a-truly-random-number/

```
function pseudo-rng()
    if first invocation of function
        lfsr = default_seed                 # any 16-bit string greater than 0x0000 fits
    newBit = (lfsr >> 0) XOR (lfsr >> 2)    # we calculate polynomial by XOR of
        XOR (lfsr >> 3) XOR (lfsr >> 5)     # different right shifts of the old lfsr value
    lfsr = (newBit << 15) OR (lfsr >> 1)
    return lfsr                             # new pseudo-rng value
```

Area di Memoria per Inserimento Elementi Lista

RIPES usa 4 byte (da 0x00000000 a 0xFFFFFFFF) per indirizzare la memoria. Per evitare potenziali overflow e/o sovrascritture di dati esistenti, si suppone di memorizzare gli elementi della lista unicamente nella porzione di memoria che va dall'indirizzo 0x00010000 all'indirizzo 0x0001FFFF.

Quindi, l'indirizzo a 16 bit (2 byte) derivato tramite LFSR quando si inserisce un nuovo elemento rappresenta i 2 byte meno significativi dell'indirizzo, ai quali si deve sempre sommare 0x00010000 per posizionarsi nell'area corretta di memoria. Riassumendo:

indirizzo nuovo elemento lista = 0x00010000 OR 16-bit-LFSR

DEL - Rimozione di un Elemento

Questa operazione porta ad eliminare un dato elemento da una lista.

- Si deve identifcare l'elemento della lista che corrisponde all'elemento da eliminare (se presente)
- Si deve ricavare l'indirizzo dell'elemento precedente e del successivo rispetto a quello da rimuovere
- Modificare il PAHEAD dell'elemento precedente sovrascrivendolo con l'indirizzo all'elemento successivo rispetto a quello da rimuovere
- Modificare il PBACK dell'elemento successivo sovrascrivendolo con l'indirizzo all'elemento precedente rispetto a quello da rimuovere

PRINT - Stampa della Lista

Questa funzionalità stampa il contenuto della lista, in ordine di apparizione

SORT - Ordinamento della Lista

Questa funzionalità ordina gli elementi della lista in base ad un dato algoritmo di ordinamento. Costituirà merito addizionale – anche se non è obbligatoria - l'implementazione ricorsiva di un algoritmo di rdinamento.

REV – Inversione degli Elementi della Lista

Questa funzionalità inverte la lista esistente. Tale operazione può essere fatta sia modificando i puntatori degli elementi della lista sia cambiando l'informazione contenuta dagli elementi.

Note e Modalità di Consegna

Note

- Seguire fedelmente tutte le specifiche dell'esercizio (incluse quelle relative ai nomi delle variabili e al formato del loro contenuto).
- 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. Si richiede in particolare di *implementare ogni operazione (ciascun algoritmo ADD-DEL-PRINT-SORT-REV) come una procedura separata.*
- 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, 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 achivio 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
- un **breve video** (max 5 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
- la **relazione** in formato PDF, strutturata come segue.
    1. **Informazioni** su autori, indirizzo mail, matricola e data di consegna
    2. **Descrizione** della soluzione adottata, trattando principalmente i seguenti punti:
        a. Descrizione ad alto livello di ciascun algoritmo di cifratura/decifratura, di altre eventuali procedure e del main, in linguaggio naturale, con flow-chart, in pseudo-linguaggio, etc
        b. Uso dei registri e memoria (stack, piuttosto che memoria statica o dinamica)
    3. **Test** per fornire evidenze del corretto funzionamento del programma, eventualmente anche in presenza di input errati. Devono comparire **almeno** i test che usano gli input descritti nella parte di "Esempio"