esercizio n. 1 – linguaggio macchina

prima parte – traduzione da C a linguaggio macchina RISC V

Si deve tradurre in linguaggio macchina simbolico (assemblatore) ***RISC-V*** il frammento di programma C riportato sotto. Il modello di memoria è quello **standard *RISC-V*** e le variabili intere sono da **64 bit**. Non si tenti di accorpare od ottimizzare insieme istruzioni C indipendenti. Si facciano le ipotesi seguenti:

- il registro "frame pointer" *fp* **non è in uso**
- le variabili locali sono allocate nei registri, se possibile
- vanno **salvati** (a cura del chiamante o del chiamato, secondo il caso) **solo i registri necessari**
- l'allocazione delle variabili in memoria non è allineata (non c'è frammentazione di memoria)

Si chiede di svolgere i quattro punti seguenti (usando le varie tabelle predisposte nel seguito):

1. **Si descriva** il segmento dei dati statici indicando gli indirizzi assoluti iniziali delle variabili globali e **si traducano** in linguaggio macchina le dichiarazioni delle variabili globali.
2. **Si descriva** l'area di attivazione della funzione `vsign`, secondo il modello RISC V, e l'allocazione dei parametri e delle variabili locali della funzione `vsign` usando le tabelle predisposte.
3. **Si traduca** in linguaggio macchina **il codice degli statement riquadrati nella funzione** `main`.
4. **Si traduca** in linguaggio macchina il codice **dell'intera funzione** `vsign` (vedi tab. 4 strutturata).

```
/* costanti e variabili globali                                    */
#define N 5
typedef long long int LONG
LONG VECTOR [N]
LONG signature = 0
/* testate funzioni ausiliarie – ambo sono funzioni foglia  */
LONG getstdin ( )       /* legge un intero da standard input  */
LONG checksum (LONG *)  /* calcola sommario (hash) di vettore */
/* funz. vsign – legge e firma vettore tramite una chiave   */
LONG vsign (LONG key, LONG * base)   {
   LONG count
   LONG * hash
   hash = base
   count = N
   do {
      count--
      VECTOR [count] = getstdin ( ) + count
   } while (count != 0)   /* do */
   *hash = checksum (base)
```

```
   return (*hash - key)
}  /* vsign */
/* programma principale                                          */
int main ( ) {
   signature = vsign (getstdin ( ), VECTOR)
}  /* main */
```

punto 1 – segmento dati statici (numero di righe non significativo)

contenuto simbolico	indirizzo assoluto iniziale (in hex)	
		indirizzi alti
...		
SIGNATURE		
VECTOR [4]		
...		
VECTOR [0]		indirizzi bassi

punto 1 – codice RISC V della sezione dichiarativa globale (numero di righe non significativo)
`.data 0x 0000 0000 1000 0000 // segm. dati statici standard`

punto 2 – area di attivazione della funzione **VSIGN**		
contenuto simbolico	**spiazz. rispetto a stack pointer**	
		indirizzi alti
		indirizzi bassi

punto 2 – allocazione dei parametri e delle variabili locali di **VSIGN** nei registri	
parametro o variabile locale	**registro**

punto 3 – codice RISC V dello statement riquadrato in **MAIN** (num. righe non significativo)
`// signature = vsign (getstdin ( ), VECTOR)`
`MAIN:`

punto 4 – codice RISC V della funzione **VSIGN** (numero di righe non significativo)
VSIGN: **addi** sp, sp, // COMPLETARE - *crea area attivazione*
// direttive EQU e salvataggio registri - NON VANNO RIPORTATI
// hash = base
// count = N
DO: // do
// count--
// VECTOR [count] = getstdin ( ) + count
// while (count != 0)
// *hash = checksum (base)
// return (*hash - key)

```
          // ripristino registri - NON VANNO RIPORTATI
```

seconda parte – assemblaggio e collegamento – RISC V

Dati i due moduli assemblatore seguenti, **si compilino** le tabelle relative a:

1. i due moduli oggetto MAIN e AUXILIARY (aggiungendo gli argomenti mancanti)
2. le basi di rilocazione del codice e dei dati di entrambi i moduli
3. la tabella globale dei simboli
4. la tabella di impostazione del calcolo delle costanti e degli spiazzamenti di istruzione e di dato
5. la tabella del codice eseguibile

modulo MAIN

```
            .data
BUF:        .space    56
            .text
            .globl    MAIN
MAIN:       mv        a2, zero
            la        t0, SUM
            ld        a3, (t0)
            jal       AUX
            bne       a0, zero, MAIN
            mv        t1, a0
            addi      t1, t1, 1
            la        t0, BUF
            sd        t1, (t0)
            j         MAIN
```

modulo AUXILIARY

```
            .data
            .eqv      CONST, 5
SUM:        .dword    20
            .text
            .globl    AUX
AUX:        beq       a2, a3, SKIP
            ret
SKIP:       addi      a2, a2, CONST
            la        t0, BUF
            sd        a2, (t0)
            ret
```

Regola generale per la compilazione di **tutte** le tabelle contenenti codice:

- espandere tutte le pseudo-istruzioni
- i codici operativi e i nomi dei registri vanno indicati in formato simbolico
- tutte le costanti numeriche all'interno del codice vanno indicate in esadecimale, con o senza prefisso 0x, e di lunghezza giusta per il codice che rappresentano

  esempio: un'istruzione come ***addi t0, t0, 15*** è rappresentata: ***addi t0, t0, 0x 00F***
- nei moduli oggetto i valori numerici che non possono essere indicati poiché dipendono dalla rilocazione successiva, vanno posti a zero e avranno un valore definitivo nel codice eseguibile

(1) – moduli oggetto

modulo `MAIN`

dimensione testo: 30 hex (48 dec)

dimensione dati: 38 hex (56 dec)

testo

indirizzo di parola	istruzione (COMPLETARE)
0	**addi** a2, zero, 0x 000
4	**auipc** t0,
8	**addi** t0,
C	**ld** a3, (t0)
10	**jal** ra,
14	**bne** a0, $zero,
18	**addi** t1,
1C	**addi** t1, t1, 0x 0001
20	**auipc** t0,
24	**addi** t0,
28	**sd** t1, (t0)
2C	**jal** zero,

dati

indirizzo di parola	contenuto

tabella dei simboli
tipo può essere *T* (testo) oppure *D* (dato)

simbolo	tipo	valore
BUF		
MAIN		

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo

modulo `AUXILIARY`

dimensione testo:

dimensione dati:

testo

indirizzo di parola	istruzione (COMPLETARE)
0	**beq** a2, a3,
4	**jalr** zero, 0(ra)
8	**addi** a2, a2,
C	**auipc** t0,
10	**addi** t0,
14	**sd** a2, (t0)
18	**jalr** zero, 0(ra)
1C	
20	
24	
28	
2C	

dati

indirizzo di parola	contenuto

tabella dei simboli
tipo può essere *T* (testo) oppure *D* (dato)

simbolo	tipo	valore
SUM		
AUX		
SKIP		

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo

(2) – posizione in memoria dei moduli			
modulo `MAIN`		**modulo** `AUXLIARY`	
base del testo:	`0x 0000 0000 0040 0000`	base del testo:	
base dei dati:	`0x 0000 0000 1000 0000`	base dei dati:	

(3) – tabella globale dei simboli				
simbolo	**valore finale**		**simbolo**	**valore finale**
`BUF`	`0x 0000 0000 1000 0000`		`SUM`	
`MAIN`	`0x 0000 0000 0040 0000`		`AUX`	
			`SKIP`	

(4) impostazione calcolo delle costanti e degli spiazzamenti di istruzione e di dato	
modulo `MAIN`	**modulo** `AUXILIARY`

NELLA TABELLA DEL CODICE ESEGUIBILE SI CHIEDONO SOLO LE ISTRUZIONI DEI MODULI MAIN E AUXILIARY CHE ANDRANNO COLLOCATE AGLI INDIRIZZI SPECIFICATI

(5) – codice eseguibile	
testo	
indirizzo	**codice** (con codici operativi e registri in forma simbolica)
. . .	
10	
. . .	
20	
24	
. . .	
2C	
. . .	
30	
3C	
40	
. . .	