Politecnico di Milano
Dip. di Elettronica, Informazione e Bioingegneria

prof. Luca Breveglieri prof.ssa Donatella Sciuto
prof. Gerardo Pelosi prof.ssa Cristina Silvano

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di lunedì 8 novembre 2021

Cognome_____ **Nome** _____

Matricola_____ **Firma**_____

Istruzioni

Si scriva solo negli spazi previsti nel testo della prova e non si separino i fogli.

Per la minuta si utilizzino le pagine bianche inserite in fondo al fascicolo distribuito con il testo della prova. I fogli di minuta, se staccati, vanno consegnati intestandoli con nome e cognome.

È vietato portare con sé libri, eserciziari e appunti, nonché cellulari e altri dispositivi mobili di calcolo o comunicazione. Chiunque fosse trovato in possesso di documentazione relativa al corso – anche se non strettamente attinente alle domande proposte – vedrà annullata la propria prova.

Non è possibile lasciare l'aula conservando il tema della prova in corso.

Tempo a disposizione 2 h : 00 m

Valore indicativo di domande ed esercizi, voti parziali e voto finale:

esercizio 1 (6 punti) _____

esercizio 2 (2 punti) _____

esercizio 3 (6 punti) _____

esercizio 4 (2 punti) _____

voto finale: (16 punti) _____

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   */
/* la funzione getstdin non altera i registri di argomento   */
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	0x 0000 0000 1000 0028	
VECTOR [4]	0x 0000 0000 1000 0020	
...		
VECTOR [0]	0x 0000 0000 1000 0000	indirizzi bassi

punto 1 – codice RISC V della sezione dichiarativa globale (numero di righe non significativo)
`.eqv N, 5 // costante numerica`
`.data 0x 0000 0000 1000 0000 // segm. dati statici standard`
VECTOR: .space 40
SIGNATURE: .dword 0

punto 2 – area di attivazione della funzione **`VSIGN`**		
contenuto simbolico	**spiazz. rispetto a stack pointer**	
ra	16	indirizzi alti
count	8	
hash	0	
		indirizzi bassi

punto 2 – allocazione dei parametri e delle variabili locali di **`VSIGN`** nei registri	
parametro o variabile locale	**registro**
count	s0
hash	s1
key	a2
base	a3

punto 3 – codice RISC V dello statement riquadrato in **`MAIN`** (num. righe non significativo)
`// signature = vsign (getstdin ( ), VECTOR)`
`MAIN:`
jal ra, GETSTDIN
mv a2, a0
la a3, VECTOR
jal ra, VSIGN
la t0, SIGNATURE
sd a0, 0(t0)

punto 4 – codice RISC V della funzione **VSIGN** (numero di righe non significativo)
VSIGN: **addi** sp, sp, -24 // COMPLETARE - *crea area attivazione*
// direttive EQU e salvataggio registri - NON VANNO RIPORTATI
// hash = base
mv s1, a3
// count = N
la t0, N
ld s0, 0(t0)
DO: // do
// count--
addi s0, s0, -1
// VECTOR [count] = getstdin ( ) + count
jal ra, GETSTDIN
add t0, a0, s0
la t1, VECTOR
slli t2, s0, 3
add t1, t1, t2
sd t0, 0(t1)
// while (count != 0)
bne s0, zero, DO
// *hash = checksum (base)
addi sp, sp, -8
sd a2, 0(sp)
mv a2, a3
jal ra, CHECKSUM
sd a0, 0(s1)
ld a2, 0(sp)
addi sp, sp, 8
// return (*hash - key)
ld t0, 0(s1)
sub t0, t0, a2
mv a0, t0
// ripristino registri - NON VANNO RIPORTATI
addi sp, sp, 24
jalr zero, ra

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, 0x 0000 0
8	**addi** t0, t0, 0x 000
C	**ld** a3, 0(t0)
10	**jal** ra, 0x 0 0000
14	**bne** a0, $zero, 0x FF6
18	**addi** t1, a0, 0x 000
1C	**addi** t1, t1, 0x 0001
20	**auipc** t0, 0x 0000 0
24	**addi** t0, t0, 0x 000
28	**sd** t1, 0(t0)
2C	**jal** zero, 0x F FFEA

dati

indirizzo di parola	contenuto
BUF	Non inizializzato

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

simbolo	tipo	valore
BUF	D	0x 0000 0000 0000 0000
MAIN	T	0x 0000 0000 0000 0000

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo
4	aiupc	SUM
8	addi	SUM
10	jal	AUX
20	aiupc	BUF
24	addi	BUF

modulo AUXILIARY

dimensione testo: 0x1C (28 dec)

dimensione dati: 0x 8 (8 dec)

testo

indirizzo di parola	istruzione (COMPLETARE)
0	**beq** a2, a3, 0x 004
4	**jalr** zero, 0(ra)
8	**addi** a2, a2, 0x 005
C	**auipc** t0, 0x 0 0000
10	**addi** t0, t0, 0x 000
14	**sd** a2, 0(t0)
18	**jalr** zero, 0(ra)
1C	
20	
24	
28	
2C	

dati

indirizzo di parola	contenuto
SUM	0x 0000 0000 0000 0014

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

simbolo	tipo	valore
SUM	D	0x 0000 0000 0000 0000
AUX	T	0x 0000 0000 0000 0000
SKIP	T	0x 0000 0000 0000 0008

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo
C	aiupc	BUF
10	addi	BUF

(2) – posizione in memoria dei moduli			
modulo `MAIN`		**modulo `AUXLIARY`**	
base del testo:	`0x 0000 0000 0040 0000`	base del testo:	0x 0000 0000 0040 0030
base dei dati:	`0x 0000 0000 1000 0000`	base dei dati:	0x 0000 0000 1000 0038

(3) – tabella globale dei simboli				
simbolo	**valore finale**		**simbolo**	**valore finale**
`BUF`	`0x 0000 0000 1000 0000`		`SUM`	0x 0000 0000 1000 0038
`MAIN`	`0x 0000 0000 0040 0000`		`AUX`	0x 0000 0000 0040 0030
			`SKIP`	0x 0000 0000 0040 0038

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

10. 0x 0000 0000 0040 0030 - 0x 0000 0000 0040 0010 = 0x 000 0000 0040 0020 => 0x 0000 0000 0020 0010

20: 0x 0000 0000 1000 0000 - 0x 0000 0000 0040 0020 = 0x 0000 0000 0FBF FFE0 => 0x 0 FBFF + 1 = 0x 0FC00

24: 0x 0000 0000 0FBF FFE0 => 0x FE0

2C: 0x 0000 0000 0040 0000 - 0x 0000 0000 0040 002C = 0x FFFF FFFF FFFF FFD4 => 0x FFFF FFFF FFFF FFEA

30: 2 => 0x 004

3C: 0x 0000 0000 0040 0000 - 0x 0000 0000 0040 003C = 0x 0000 0000 0FBF FFC4 => 0x 0 FBFF + 1 = 0x 0 FC00

40: 0x 0000 0000 0FBF FFC4 => 0x FC4

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	jal ra, 0x 0 0010
. . .	
20	auipc t0, 0x 0FC0 0
24	addi t0, t0, 0x FE0
. . .	
2C	jal zero, 0x F FFEA
. . .	
30	beq a2, a3, 0x 004
. . .	
3C	auipc t0, 0x 0 FC00
40	addi t0, t0, 0x FC4
. . .	