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

PRIMA PARTE – martedì 30 agosto 2022

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 1 h : 30 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 `chksum`, secondo il modello RISC V, e l'allocazione dei parametri e delle variabili locali della funzione `chksum` 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** `chksum` (vedi tab. 4 strutturata).

```
/* costanti e variabili globali                                    */
#define N 7
typedef long long int LONG
LONG code = 18
char LETTERS [N] /* N.B: in C il char è un numero con segno */

/* funzione chksum                                                 */
LONG chksum (char * byte, LONG weight) {
   LONG idx   s0
   LONG * ptr  s1
   LONG partial   salvata sullo stack
   ptr = &partial
   *ptr = 0
   for (idx = N - 1; idx >= 0; idx--) {
      *ptr = *ptr + byte [idx] - weight
      weight++
   } /* for */
   return *ptr
}  /* chksum */
/* programma principale                                            */

void main ( ) {
   code = chksum (LETTERS, code)
}  /* main */
```

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

contenuto simbolico	indirizzo assoluto iniziale (in hex)	
...		indirizzi alti
LETTERS [N – 1]	0x 0000 0000 1000 000E	
...		
LETTERS [1]	0x 0000 0000 1000 0009	
LETTERS [0]	0x 0000 0000 1000 0008	
CODE	0x 0000 0000 1000 0000	indirizzi bassi

punto 1 – codice RISC V della sezione dichiarativa globale (numero di righe non significativo)
.data 0x 0000 0000 1000 0000 // segmento dati statici standard
CODE: .dword 18
LETTERS: .space 7
.eqv N 7

punto 2 – area di attivazione della funzione **CHKSUM**	
contenuto simbolico	**spiazz. rispetto a stack pointer**
s0	0x 10 (16 in dec)
s1	0x 8 (8 in dec)
partial	0x 0

indirizzi alti

indirizzi bassi

punto 2 – allocazione dei parametri e delle variabili locali di **CHKSUM** nei registri	
parametro o variabile locale	**registro**
byte	a2
weight	a3
idx	s0
ptr	s1

punto 3 – codice RISC V degli statement riquadrati in **MAIN** (num. righe non significativo)
`MAIN:  //  code = chksum (LETTERS, code)`
la a2, LETTERS
la t0, CODE
ld a3, 0(t0)
jal ra, CHKSUM
la t0, CODE
sd a0, 0(t0)

punto 4 – codice RISC V della funzione **CHKSUM** (numero di righe non significativo)
CHKSUM: **addi** sp, sp, -16 // **COMPLETARE** - crea area attivazione
// direttive EQV e salvataggio registri – **DA COMPLETARE**
.eqv S0 16
.eqv S1 8
.eqv PARTIAL 0
// ptr = &partial
add t0, sp, PARTIAL
mv s1, t0
// *ptr = 0
sd zero, 0(s1)
// for (idx = N - 1; idx >= 0; idx--)
li s0, N - 1
FOR:
blt s1, zero, ENDFOR
// *ptr = *ptr + byte [idx] - weight
ld t0, 0(s1)
add t1, a2, s0
ld t2, 0(t1)
add t0, t0, t2
sub t0, t0, a3
sd t0, 0(s1)
// weight++
addi a3, a3, 1
// idx--
addi s0, s0, -1
j FOR
ENDFOR: // return *ptr
sd a0, 0(s1)
// chiusura funzione – **NON RIPORTARE**

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 TASK
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

```
          .eqv     CONST, 0x 14A7B72D
          .data
WEIGHT:   .dword   30
REF:      .word    40
          .text
          .globl   MAIN
MAIN:     addi  a2, s0, 0
          li    a3, CONST
FUNCT:    jal   TASK
          beq   a0, zero, NEXT
          la    t0, WEIGHT
NEXT:     sd    a0, (t0)
          j     MAIN
```

modulo TASK

```
          .data
RESERVE:  .space   24
LOCAL:    .dword   0
          .text
          .globl   TASK
TASK:     bne   a2, zero, AFTER
          andi  a2, a3, 0x 123
          la    t0, REF
          sw    a2, (t0)
AFTER:    beq   s0, a2, NEXT
          ret
```

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

- espandere tutte le pseudo istruzioni
- l'allocazione delle variabili in memoria non è allineata (non c'è frammentazione di memoria)
- 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: 0x 24 (32 in dec)

dimensione dati: 0x 0C (12 in dec)

testo

indirizzo di parola	istruzione (COMPLETARE)
0	**addi** a2, s0, 0
4	**lui** a3, 0x 14A7B
8	**addi** a3, a3, 0x 72D
C	jal ra, 0x000
10	**beq** a0, zero, 0x 006
14	**auipc** t0, 0x 0000 0
18	**addi** t0, 0x 000
1C	**sd** a0, 0(t0)
20	jal zero, 0x F FFEA
24	
28	
2C	

dati

indirizzo di parola	contenuto
0	0x 0000 0000 0000 001E
8	0x 0000 0028

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

simbolo	tipo	valore (hex)
WEIGHT	D	0x 0000 0000 0000 0000
REF	D	0x 0000 0000 0000 0008
MAIN	T	0x 0000 0000 0000 0000
FUNCT	T	0x 0000 0000 0000 000C
NEXT	T	0x 0000 0000 0000 001C

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo
C	jal	TASK
14	auipc	WEIGHT
18	addi	WEIGHT

modulo TASK

dimensione testo: 0x 1C (28 in dec)

dimensione dati: 0x 20 (32 in dec)

testo

indirizzo di parola	istruzione (COMPLETARE)
0	**bne** a2, zero, 0x 00A
4	**andi** a2, a3, 0x 123
8	**auipc** t0, 0x 0000 0
C	**addi** t0, t0, 0x 000
10	**sw** a2, 0(t0)
14	**beq** s0, a2, 0x 000
18	jalr zero, 0(ra)
1C	
20	
24	
28	
2C	

dati

indirizzo di parola	contenuto
0	Non inizializzato
0x 18	0

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

simbolo	tipo	valore (hex)
RESERVE	D	0x 0000 0000 0000 0000
LOCAL	D	0x 0000 0000 0000 0018
TASK	T	0x 0000 0000 0000 0000
AFTER	T	0x 0000 0000 0000 0014

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo
8	auipc	REF
C	addi	REF
14	***beq***	NEXT

(2) – posizione in memoria dei moduli	
modulo `MAIN`	**modulo `TASK`**
base del testo: `0x 0000 0000 0040 0000`	base del testo: 0x 0000 0000 0040 0024
base dei dati: `0x 0000 0000 1000 0000`	base dei dati: 0x 0000 0000 1000 000C

(3) – tabella globale dei simboli				
simbolo	**valore finale (hex)**		**simbolo**	**valore finale (hex)**
`WEIGHT`	0x 0000 0000 1000 0000		`RESERVE`	0x 0000 0000 1000 000C
`REF`	0x 0000 0000 1000 0008		`LOCAL`	0x 0000 0000 1000 0024
`MAIN`	0x 0000 0000 0040 0000		`TASK`	0x 0000 0000 0040 0024
`FUNCT`	0x 0000 0000 0040 000C		`AFTER`	0x 0000 0000 0040 0038
`NEXT`	0x 0000 0000 0040 001C			

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

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

(5) – codice eseguibile	
testo	
indirizzo (hex)	**codice** (con codici operativi e registri in forma simbolica)
. . .	
C	jal ra, 0x 00C
. . .	
14	auipc t0, 0x 0 FC00
18	addi t0, t0, 0x FEC
20	jal ra, 0x F FFF0
. . .	
2C	auipc t0, 0x 0 FC00
30	addi t0, t0, 0x FD4
38	beq s0, a2, 0x FF2
. . .	

C: 0x 0000 0000 0040 0024 - 0x 0000 0000 0040 000C = 0x 0000 0000 0000 0018 => 0x 000C

14: 0x 0000 0000 1000 0000 - 0x 0000 0000 0040 0014 = 0x 0000 0000 0FBF FFEC => 0x 0 FBFF + 1 = 0x 0 FC00

18: 0x 0000 0000 0FBF FFEC => 0x FEC

20: 8* 2 = 16 => -16 => 0x F FFF0

2C: 0x 0000 0000 1000 0008 - 0x 0000 0000 0040 002C = 0x 0000 0000 0FBF FFD4 => 0x 0 FBFF + 1 = 0x 0 FC00

30: 0x 0000 0000 0FBF FFD4 => 0x FD4

38: 0x 0000 0000 0040 001C - 0x 0000 0000 0040 0038 = 0x FFFF FFFF FFFF FFE4 => 0x FF2