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

RISC V - PRIMA PARTE – giovedì 23 giugno 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

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

```
/* costanti e variabili globali                                    */
#define N 28
typedef long long int LONG
char word [N]
LONG maius
/* testata funzione ausiliaria - è una funzione foglia             */
/* se c è carattere maiuscolo restituisce 1, altrimenti 0          */
LONG ucase (char c) /* in C il tipo char è un intero con segno */
/* funz. verify - verifica se la stringa è maiuscola               */
LONG verify (char * string, LONG dim) {
   char * ptr
   LONG yes
   LONG cnt
   ptr = string
   yes = 1
   for (cnt = dim - 1, cnt >= 0, cnt--) {
      yes = ucase (*ptr) && yes
      ptr++
   } /* for */
   return yes
}  /* verify */
/* programma principale                                            */
void main ( ) {
   maius = verify (word, N)
}  /* main */
```

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

contenuto simbolico	indirizzo assoluto iniziale (in hex)	
		indirizzi alti
...		
MAIUS	0x 0000 0000 1000 001C	
WORD [N – 1]	0x 0000 0000 1000 001B	
...		
WORD [0]	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
.eqv N, 28
WORD: .space N
MAIUS: .space 8

punto 2 – area di attivazione della funzione **`VERIFY`**		
contenuto simbolico	**spiazz. rispetto a stack pointer**	
ra	24	indirizzi alti
ptr	16	
yes	8	
cnt	0	
		indirizzi bassi

punto 2 – allocazione dei parametri e delle variabili locali di **`VERIFY`** nei registri	
parametro o variabile locale	**registro**
ptr	s0
yes	s1
cnt	s2
string	a2
dim	a3

punto 3 – codice RISC V dello statement riquadrato in **`MAIN`** (num. righe non significativo)
`// maius = verify (word, N)`
`MAIN:`
la a2, WORD
add a3, zero, N
jal ra, VERIFY
la t0, MAIUS
sd a0, 0(t0)

punto 4 – codice RISC V della funzione **`VERIFY`** (numero di righe non significativo)
`VERIFY:` **`addi`** `sp, sp, -32  // COMPLETARE - crea area attivazione`
`// direttive EQU e salvataggio registri - NON VANNO RIPORTATI`
`// ptr = str`
`mv s0, a2`
`// yes = 1`
`li s1, 1`
`// for (cnt = dim - 1, cnt >= 0, cnt--)`
`addi s2, a3, -1`
`FOR:`
`blt s2, zero, ENDFOR`
`// yes = ucase (*ptr) && yes`
`lb a2, 0(s0)`
`jal ra, UCASE`
`and s1, a0, s1`
`// ptr++`
`addi s0, s0, 1`
`// cnt--`
`addi s2, s2, -1`
`j FOR`
`ENDFOR:  // ripristino registri - NON VANNO RIPORTATI`
`// restituisci valore, elimina area e rientra`
`mv a0, s1`
`addi sp, sp, 32`
`ret`

seconda parte – assemblaggio e collegamento

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

1. i due moduli oggetto MAIN e SECONDARY (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
BLOCK:    .space   30
          .text
          .globl   MAIN
MAIN:     mv    a2, zero
          la    t0, BLOCK
FUN:      jal   SEC
          bne   a2, zero, OVER
          mv    t0, a2
OVER:     addi  t0, t0, 1
          la    t1, SUB
          sd    t0, (t1)
          j     FUN
```

modulo SECONDARY

```
          .data
SUB:      .dword   0
          .text
          .globl   SEC
SEC:      bne   a2, t0, FUN
          mv    a2, zero
LOOP:     addi  a2, a2, 1
          la    t0, BLOCK
          sd    a2, (t0)
          beq   t0, a2, LOOP
          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: 0x 2C (44 in dec.)

dimensione dati: 0x 1E (30 in dec)

testo

indirizzo di parola	istruzione (COMPLETARE)
0	**addi** a2, zero, zero
4	**auipc** t0, 0x 0 0000
8	**addi** t0, t0, 0x 000
C	**jal** ra, 0x 0 0000
10	**bne** a2, zero, 0x 004
14	**addi** t0, a2, 0x 000
18	**addi** t0, t0, 0x 001
1C	**auipc** t1, 0x 0 0000
20	**addi** t1, t1, 0x 000
24	**sd** t0, 0(t1)
28	**jal** zero, 0x F FFF2
2C	

dati

indirizzo di parola	contenuto
0	30 Byte non specificati

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

simbolo	tipo	valore
BLOCK	D	0x 0000 0000 0000 0000
MAIN	T	0x 0000 0000 0000 0000
FUN	T	0x 0000 0000 0000 000C
OVER	T	0x 0000 0000 0000 0018

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo
4	auipc	BLOCK
8	addi	BLOCK
C	jal	SEC
1C	auipc	SUB
20	addi	SUB

modulo SECONDARY

dimensione testo: 0x 20 (32 in dec.)

dimensione dati: 0x 8 (8 in dec.)

testo

indirizzo di parola	istruzione (COMPLETARE)
0	**bne** a2, t0, 0x 000
4	**addi** a2, zero, zero
8	**addi** a2, a2, 0x 001
C	**auipc** t0, 0x 0 0000
10	**addi** t0, t0, 0x 000
14	**sd** a2, 0(t0)
18	**beq** t0, a2, 0x FFC
1C	**jalr** zero, 0x 000(ra)
20	
24	
28	
2C	

dati

indirizzo di parola	contenuto
0	0x 0000 0000 0000 0000

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

simbolo	tipo	valore
SUB	D	0x 0000 0000 0000 0000
SEC	T	0x 0000 0000 0000 0000
LOOP	T	0x 0000 0000 0000 0008

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo
0	bne	FUN
C	auipc	BLOCK
10	addi	BLOCK

(2) – posizione in memoria dei moduli	
modulo `MAIN`	**modulo `SECONDARY`**
base del testo: `0x 0000 0000 0040 0000`	base del testo: 0x 0000 0000 0040 002C
base dei dati: `0x 0000 0000 1000 0000`	base dei dati: 0x 0000 0000 1000 001E

(3) – tabella globale dei simboli				
simbolo	**valore finale**		**simbolo**	**valore finale**
`BLOCK`	0x 0000 0000 1000 0000		`SUB`	0x 0000 0000 1000 001E
`MAIN`	0x 0000 0000 0040 0000		`SEC`	0x 0000 0000 0040 002C
`FUN`	0x 0000 0000 0040 000C		`LOOP`	0x 0000 0000 0040 0034
`OVER`	0x 0000 0000 0040 0018			

(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 ROUTINE CHE ANDRANNO COLLOCATE AGLI INDIRIZZI SPECIFICATI

(5) – codice eseguibile	
testo	
indirizzo	**codice** (con codici operativi e registri in forma simbolica)
. . .	
4	auipc t0, 0x 0 FC00
8	addi t0, t0, 0x FFC
C	jal ra, 0x 0 0010
. . .	
1C	auipc t1, 0x 0 FC00
20	addi t1, t1, 0x 002
. . .	
2C	bne a2, t0, 0x FF0
. . .	

4: 0x 0000 0000 1000 0000 - 0x 0000 0040 0004 = 0x 0000 0000 0FBF FFFC => 0x 0 FC00

8: 0x 0000 0000 0FBF FFFC => 0x FFC

C: 0x 0000 0000 0040 002C - 0x 0000 0000 0040 000C = 0x 0000 0000 0000 0020 => 0x 0000 0000 0000 0010

1C: 0x 0000 0000 1000 001E - 0x 0000 0000 0040 001C = 0x 0000 0000 0FC0 0002 => 0x 0 FC00

20: 0x 0000 0000 0FC0 0002 => 0x 002

2C: 0x 0000 0000 0040 000C - 0x 0000 0000 0040 002C = 0x FFFF FFFF FFFF FFE0 => 0x FF0