**Politecnico di Milano**
**Dip. di Elettronica, Informazione e Bioingegneria**

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

# AXO RISC V – Architettura dei Calcolatori e Sistemi Operativi

## Prova di lunedì 9 novembre 2020

**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 **È ALLINEATA** (ci può essere 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 `ping`, secondo il modello RISC-V, e l'allocazione dei parametri e delle variabili locali della funzione `ping` usando le tabelle predisposte
3. **Si traduca** in linguaggio macchina **il codice dello statement riquadrato nella funzione** `main`.
4. **Si traduca** in linguaggio macchina il codice **dell'intera funzione** `ping` (vedi tab. 4 strutturata).

```
/* variabili globali */
/* stringa inizializzata, termina con il carattere NULL */
typedef long long int LONG
char animal [ ] = "RAT"
LONG n = 1
LONG off [ ] = { 13, -14, 14 }

LONG strlen (char *)          /* funzione che modifica arg */

/* Nota Bene: in C il char è un numero con segno */
/* funzione ricorsiva */
LONG ping (LONG * ptr, char * string) {
   LONG tmp = 0
   if (strlen (string) > 0) {
      string [0] = string [0] - *ptr
      tmp = ping (ptr - 1, &string [1]) + tmp - 65
   }  /* if */
   return tmp
}  /* ping */

int main ( ) {                    /* programma principale */
   n = ping (&off [2], animal)
   printf ("\n num = %d, new animal: %s", n, animal)
   return 0
}  /* main */
```

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

| contenuto simbolico | indirizzo assoluto iniziale (in hex) | |
|---|---|---|
| OFF [2] | 0x 0000 0000 1000 0020 | indirizzi alti |
| OFF [1] | 0x 0000 0000 1000 0018 | |
| OFF [0] | 0x 0000 0000 1000 0010 | |
| N | 0x 0000 0000 1000 0008 | |
| ANIMAL [3] | 0x 0000 0000 1000 0003 | |
| … | … | |
| ANIMAL [0] | x0 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  // seg. dati statici standard
ANIMAL: .asciiz  "RAT"          // varglob array stringa ANIMAL (4 char)
        .align   3              // allineamento a parola doppia
```

N: .dword 1

OFF: .dword 13, -14, 14

| **punto 2** – area di attivazione della funzione **PING** | | |
|---|---|---|
| **contenuto simbolico** | **spiazz. rispetto a stack pointer** | |
| ra | +8 | indirizzi alti |
| tmp | 0 | |
| | | |
| | | |
| | | indirizzi bassi |

| **punto 2** – allocazione dei parametri e delle variabili locali di **PING** nei registri | |
|---|---|
| **parametro o variabile locale** | **registro** |
| *ptr | a2 |
| *string | a3 |
| tmp | s0 |
| | |

| **punto 3** – codice RISC-V dello statement riquadrato in **MAIN** (num. righe non significativo) |
|---|
| `//  n = ping (&off [2], animal)` |
| `MAIN:` |
| la t0, OFF |
| li t1, 2 |
| slli t1, t1, 3 |
| add t0, t0, t1 |
| mv a2, t0 |
| la a3, ANIMAL |
| jal ra, PING |
| la t0, N |
| sd a0, 0(t0) |
| |

| **punto 4** – codice RISC-V della funzione **PING** (numero di righe non significativo) | |
|---|---|
| PING: | **addi** sp, sp, -16 // COMPLETARE |
| | // direttive EQU e salvataggio registri - NON VANNO RIPORTATI |
| | // tmp = 0 |
| | mv s0, zero |
| | |
| IF: | // if (strlen (string) > 0) |
| | addi sp, sp, -16 |
| | sd a2, 8(sp) |
| | sd a3, 0(sp) |
| | mv a2, a3 |
| | jal ra, STRLEN |
| | ld a2, -8(sp) |
| | ld a3, 0(sp) |
| | addi sp, sp, 16 |
| | ble a0, zero, ENDIF |
| | |
| THEN: | // string [0] = string [0] - *ptr |
| | lb t0, 0(a3) |
| | ld t1, 0(a2) |
| | sub t0, t0, t1 |
| | sb t0, 0(a3) |
| | |
| | |
| | // tmp = ping (ptr - 1, &string [1]) + tmp - 65 |
| | addi a2, a2, -8 |
| | addi a3, a3, 1 |
| | jal ra, PING |
| | add t0, a0, s0 |
| | li t1, -65 |
| | add t0, t0, t1 |
| | mv s0, t0 |
| ENDIF: | // return tmp |
| | mv a0, s0 |
| | |
| | // rientro |
| | addi sp sp, 16 |
| | ret |
| | |

**seconda parte – assemblaggio e collegamento – RISC V**

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

1. il modulo oggetto ROUTINE (il modulo MAIN è già interamente dato)
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
STR:        .asciiz   "WALLABY"
// N.B.: la direttiva asciiz aggiunge
il char NULL al termine della stringa,
per terminarla
            .text
            .globl    MAIN
MAIN:       la    a2, STR
            jal   ROUTINE
            ld    a2, (a2)
            li    a0, 1
            ecall
            li    a0, 2048
            ecall
```

**modulo ROUTINE**

```
            .data
SUMFL:      .dword    0
            .eqv      LEN, 5
            .text
            .globl    ROUTINE
ROUTINE:    ld    s0, (a2)
            addi  s1, zero, LEN
            beq   s0, zero, SKIP
            add   s1, a2, s1
SKIP:       ld    s2, (s1)
            bne   s0, s1, MAIN
            la    t0, SUMFL
            sd    s0, (t0)
            ret   ra
```

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:** 28 hex (40 dec)

**dimensione dati:** 08 hex (8 dec)

**testo**

| indirizzo di parola | istruzione |
|---|---|
| 00 | **auipc** a2, **0x 0000 0** |
| 04 | **addi** a2, a2, **0x 000** |
| 08 | **jal** ra, **0x 0 0000** |
| 0C | **ld** a2, (a2) |
| 10 | **lui** a0, **0x 0000 0** |
| 14 | **addi** a0, a0, **0x 001** |
| 18 | **ecall** |
| 1C | **lui** a0, **0x 0000 1** |
| 20 | **addi** a0, a0, **0x 800** |
| 24 | **ecall** |
| 28 | |
| 2C | |

**dati**

| indirizzo di parola | contenuto |
|---|---|
| 0 | 'W' 'A' 'L' 'L' |
| 4 | 'A' 'B' 'Y' 0x 00 |
| 8 | |

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

| simbolo | tipo | valore |
|---|---|---|
| **MAIN** | T | 0x 0000 0000 0000 0000 |
| **STR** | D | 0x 0000 0000 0000 0000 |
| | | |
| | | |

**tabella di rilocazione**

| indirizzo di parola | cod. operativo | simbolo |
|---|---|---|
| 00 | **auipc** | STR |
| 04 | **addi** | STR |
| 08 | **jal** | ROUTINE |
| | | |

### modulo `ROUTINE`

**dimensione testo:** 0x 28 (40 dec)

**dimensione dati:** 0x 8 (8 dec)

**testo**

| indirizzo di parola | istruzione |
|---|---|
| 00 | ld s0, 0(a2) |
| 04 | addi s0, zero, 0x 005 |
| 08 | beq s0, zero, 0x 001 |
| 0C | add s1, a2, s1 |
| 10 | ld s2, 0(s1) |
| 14 | bne s0, s1, 0x 000 |
| 18 | auipc t0, 0x 0 0000 |
| 1C | addi t0, t0, 0x 000 |
| 20 | sd s0, 0(t0) |
| 24 | jalr zero, ra, 0x 000 |
| 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 |
|---|---|---|
| **ROUTINE** | T | 0x 0000 0000 0000 0000 |
| **SKIP** | T | 0x 0000 0000 0000 0010 |
| **SUMFL** | D | 0x 0000 0000 0000 0000 |
| | | |

**tabella di rilocazione**

| indirizzo di parola | cod. operativo | simbolo |
|---|---|---|
| 14 | bne | MAIN |
| 18 | auipc | SUMFL |
| 1C | addi | SUMFL |
| | | |

| (2) – posizione in memoria dei moduli | |
|---|---|
| **modulo MAIN** | **modulo ROUTINE** |
| base del testo: 0x 0000 0000 0040 0000 | base del testo: 0x 0000 0000 0040 0028 |
| base dei dati: 0x 0000 0000 1000 0000 | base dei dati: 0x 0000 0000 1000 0008 |

| (3) – tabella globale dei simboli | | | | |
|---|---|---|---|---|
| **simbolo** | **valore finale** | | **simbolo** | **valore finale** |
| **MAIN** | 0x 0000 0000 0040 0000 | | **ROUTINE** | 0x 0000 0000 0040 0028 |
| **STR** | 0x 0000 0000 1000 0000 | | **SKIP** | 0x 0000 0000 0040 0038 |
| | | | **SUMFL** | 0x 0000 0000 1000 0008 |

| (4) impostazione calcolo delle costanti e degli spiazzamenti di istruzione e di dato | |
|---|---|
| **modulo MAIN** | **modulo ROUTINE** |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

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) |
| ... | |
| 0x 0000 0000 0040 0000 | auipc a2, 0x 0 FC00 |
| 0x 0000 0000 0040 0004 | addi a2, a2, 0x 000 |
| 0x 0000 0000 0040 0008 | jal ra, 0x 010 |
| ... | |
| 0x 0000 0000 0040 003C | bne s0, zero, 0x FE2 |
| 0x 0000 0000 0040 0040 | auipc t0, 0x F FC00 |
| 0x 0000 0000 0040 0044 | addi t0, t0, 0x FC8 |
| ... | |

0: 0x 0000 0000 1000 0000 - 0x 0000 0000 0040 0000 = 0x 0000 0000 0FC0 0000 => 0x 0 FC00

4: 0x 0000 0000 0FC0 0000 => 0x 000

8: 0x 0000 0000 0040 0028 - 0x 0000 0000 0040 0008 = 0x 0000 0000 0000 0020 => 0x 0000 0000 0000 0010

3C: 0x 0000 0000 0040 0000 - 0x 0000 0000 0040 003C = 0x FFFF FFFF FFFF FFC4 => 0x FE2

40: 0x 0000 00000 1000 0008 - 0x 0000 0000 0040 0040 = 0x 0000 0000 0FBF FFC8 => 0x 0 FBFF + 1 = 0x F FC00

44: 0x 0000 0000 0FBF FFC8 => 0x FC8