Politecnico di Milano – Sede di Cremona
Anno Accademico 2023/2024

Architettura dei Calcolatori e Sistemi Operativi

Esame – 14.02.2025

Prof. Carlo Brandolese

Cognome _____ **Nome** _____

Matricola _____ **Firma** _____

Istruzioni

1. Scrivere con cura, negli spazi sopra segnati, il proprio cognome, nome, numero di matricola e apporre la firma.
2. È vietato consultare libri, eserciziari, appunti ed utilizzare la calcolatrice e qualunque strumento elettronico (inclusi i cellulari), pena l'invalidazione del compito.
3. Il testo, debitamente compilato, deve essere riconsegnato in ogni caso.
4. Il tempo della prova è di 3 ore

Valutazione

Domanda	Voto	Note
A		
B		
C		
D		
E		
F		

Domanda A

Si implementi in codice assembly RISC-V la funzione:

```
int dotp( int* a, int* b, int n )
```

che calcola il prodotto scalare dei due vettori di interi a e b, cioè:

$$s = \sum_{i=0}^{n} a_i \, b_i$$

Si scriva quindi il codice del programma **`main`** che chiama la funzione passando a questa i vettori **`A`** e **`B`** come definiti dalla sezione **`.data`** riportata qui di seguito, la loro lunghezza, e che salva il risultato nella variabile **`S`**. Qui di seguito è fornita una traccia della sezione dati.

```
        .data
A:      .word   1 2 3 4
B:      .word    10 20 30 40
S:      .space  4

        .text
main:   la      a0, A
        la      a1, B
        li      a2, 4
        jal     dotp
        la      t0, S
        li      a7, 10
        ecall

dotp:   mv      t0, a0
        mv      t1, a1
        mv      t2, a2
        mv      t3, zero
loop:   beq     t2, zero, end
        lw      t4, (t0)
        lw      t5, (t1)
        mul     t4, t4, t5
        add     t3, t3, t4
        addi    t0, t0, 4
        addi    t1, t1, 4
        addi    t2, t2, -1
        b       loop
end:    mv      a0, t3
        jr      ra
```

Domanda B

Si sviluppi in C un programma secondo le seguenti specifiche:

1. Il programma è costituito da due thread produttori **gen_a** e **gen_b** ed un thread consumatore **add**
2. Il thread **gen_a** esegue un ciclo di 10 iterazioni. Ad ogni iterazione genera un numero casuale $a_i \in [0; 9]$ e lo stampa su standard output
3. Il thread **gen_b** esegue un ciclo di 10 iterazioni. Ad ogni iterazione genera un numero casuale $b_i \in [0; 99]$ e lo stampa su standard output
4. Il thread **add** esegue un ciclo in cui, ad ogni iterazione, calcola la somma $s_i = a_i + b_i$ dei valori prodotti dai thread **gen_a** e **gen_b** e stampa il risultato s_i su standard output

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

sem_t   sem_a_generated, sem_b_generated, sem_a_used, sem_b_used;
int     a, b;

void* gen_a( void* arg )
{
      for( int i = 0; i < 10; i++ ) {
            sem_wait( &sem_a_used );
            a = rand() % 10;
            printf( "A: %d\t", a  );
            sem_post( &sem_a_generated );
      }
    pthread_exit( NULL );
}

void* gen_b( void* arg )
{
      for( int i = 0; i < 10; i++ ) {
            sem_wait( &sem_b_used );
            b = rand() % 100;
            printf( "B: %d\t", b  );
            sem_post( &sem_b_generated );
      }
    pthread_exit( NULL );
}

void* add( void* arg )
{
      while( 1 ) {
            sem_wait( &sem_a_generated );
            sem_wait( &sem_b_generated );
            printf( "S: %d\n", a + b );
            sem_post( &sem_a_used );
            sem_post( &sem_b_used );
      }
      pthread_exit( NULL );
}
```

```
int main( int argc, char *argv[] )
{
    pthread_t tid_a;
    pthread_t tid_b;
    pthread_t tid_s;

      sem_init( &sem_a_generated, 0, 0 );
      sem_init( &sem_b_generated, 0, 0 );
      sem_init( &sem_s_used, 0, 0 );
      sem_init( &sem_s_used, 0, 0 );

      pthread_create( &tid_a, NULL, gen_a, NULL );
      pthread_create( &tid_b, NULL, gen_b, NULL );
      pthread_create( &tid_s, NULL, add,   NULL );

      sem_post( &sem_a_used );
      sem_post( &sem_b_used );

    pthread_join( tid_a, NULL );
    pthread_join( tid_b, NULL );

    exit(0);
}
```

Domanda C

Si consideri un sistema con uno spazio di indirizzamento complessivo pari a 1 GByte ed una struttura di cache organizzata su due livelli L0 ed L1, con le caratteristiche seguenti:

	Cache L0	**Cache L1**
Associatività	Diretta	Set associativa a 4 vie
Dimensione totale	32 KB	128 KB
Dimensione linea	128 B	256 B (per ogni set)
Tempo di accesso	1 ns	2 ns
Hit rate	98 %	99 %

Si indichi la struttura dell'indirizzo visto dalle cache, descrivendo i vari campi e il loro significato.

Struttura dell'indirizzo della cache L0

Offset = log2(128) = 7 bit

Index = log2(32K/128) = 8 bit

Tag = log2(1G) – 7 – 8 = 30 – 7 – 8 = 15 bit

Struttura dell'indirizzo della cache L1

Offset = log2(256) = 8 bit

Index = log2(128K/(256*4)) = 7 bit

Tag = log2(1G) – 8 – 7 = 30 – 8 – 7 = 15 bit

Sapendo che:

- L'accesso alla memoria avviene a parole di 64 bit
- Il tempo di accesso alla RAM in modalità normale è di 100 ns
- Il tempo di accesso alla RAM in modalità burst è di
  - 150 ns per il primo accesso
  - 50 ns per gli accessi successivi

Si calcoli il tempo medio di accesso alle due cache.

Tempo medio di accesso alla cache L1

T = 0.99 * 2ns + 0.01 * [2ns + 150ns + (256/8-1) * 50ns]

= 1.98ns + 0.01 * [2ns + 150ns + 1550n s]

= 1.98ns + 17.02ns = 19 ns

Tempo medio di accesso alla cache L0

Quando si ha un miss in L1 necessariamente si è avuto prima un miss in L0 per cui si può ipotizzare che la lettura da RAM a L1 e da L1 a L0 avvenga contemporaneamente, in una sorta di modalità "read-through". S ha pertanto:

T = 0.98 * 1ns + 0.02 * [1ns + 19ns]

= 0.98 + 0.40 = 1.38 ns

Se si ipotizza invece che la lettura avvenga in modo indipendente, il tempo di caricamento di una linea in L0 è uguale al tempo medio di accesso in L1 per la prima parola (64 bit) più il tempo di hit in cache L1 per le restanti parole, cioè:

T = 0.98 * 1ns + 0.02 * [1ns + 19ns + (128/8-1) * 2ns]

= 0.98 + 0.02 * (50 ns) = 1.98 ns

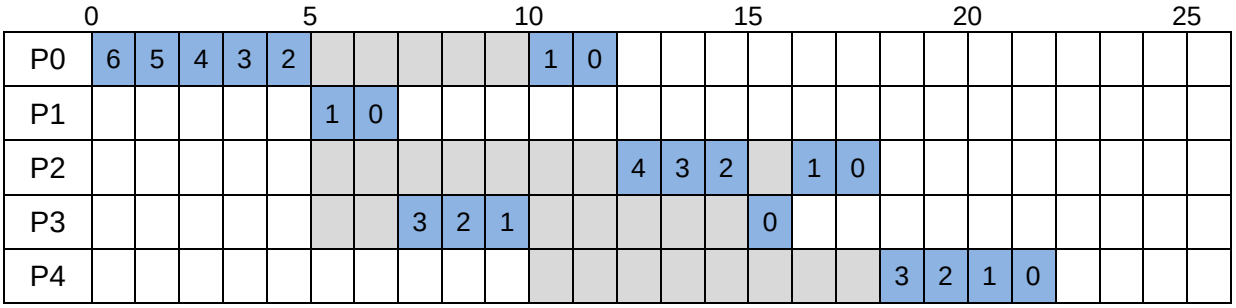
Domanda D

Si considerino i seguenti processi:

Process	Arrival Time	Execution time
P0	0	7
P1	2	2
P2	5	5
P3	5	4
P4	10	4

Si disegni sul diagramma seguente il risultato dello scheduling HRRN non preemptive con quanto di tempo pari a 5 unità.

La soluzione seguente considera come execution time sempre il **tempo nominale** iniziale di esecuzione dei processi



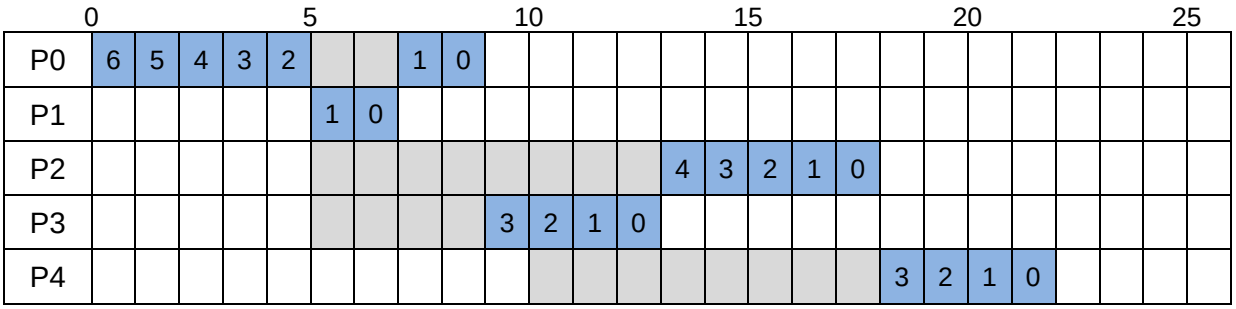
Per ogni processo si calcoli quindi l'overhead in percentuale, cioè il rapporto tra il ritardo totale ed il tempo nominale di esecuzione. Infine, si calcoli il ritardo medio di tutti i processi.

Overhead

P0 = 5 / 7 = 71 %
P1 = 0 / 2 = 0 %
P2 = 8 / 5 = 160 %
P3 = 7 / 4 = 175 %
P4 = 8 / 4 = 200 %

Ritardo medio
T = (5 + 0 + 8 + 7 + 8) / 5 = 5.6

In alternativa, si può utilizzare il tempo di esecuzione **rimanente**, ottenendo la soluzione seguente.



Per ogni processo si calcoli quindi l'overhead in percentuale, cioè il rapporto tra il ritardo totale ed il tempo nominale di esecuzione. Infine, si calcoli il ritardo medio di tutti i processi.

Overhead

P0 = 2 / 7 = 28 %
P1 = 0 / 2 = 0 %
P2 = 8 / 5 = 160 %
P3 = 4 / 4 = 100 %
P4 = 8 / 4 = 200 %

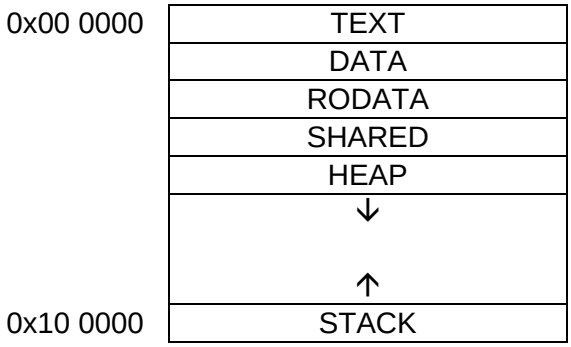
Ritardo medio
T = (2 + 0 + 8 + 4 + 8) / 5 = 4.4

Domanda E

Si consideri un sistema dotato di memoria virtuale con paginazione e segmentazione di tipo UNIX caratterizzato dai parametri seguenti:

- La memoria fisica ha capacità di 128 Kbyte
- La memoria logica ha capacità di 1MB Kbyte
- La pagina di memoria ha dimensione di 4 Kbyte

Il modello di riferimento per la segmentazione è il seguente:



Si svolgano i seguenti punti.

1. Si indichino le dimensioni in bit di:

   Indirizzo virtuale: _____20_____

   Indirizzo fisico: _____17_____

   Numero di pagina virtuale: _____8_____

   Numero di pagina virtuale: _____5_____

   Offset: _____12_____

2. Nel sistema vengono creati tre processi, indicati nel seguito con P, Q ed R. I programmi eseguiti da tali processi sono due: X e Y. La dimensione iniziale dei segmenti dei programmi è la seguente:

CX: 8K	DX: 4K	PX: 4K	Entry point: 0x00 1C30
CY: 12K	DY: 8K	PY: 4K	Entry point: 0x00 2800

   Si inserisca in tabella sottostante la struttura in pagine della memoria virtuale.

NPV	Pagine Programma X
0	CX0
1	CX1 (entry point)
2	DX0
255	PX0

NPV	Pagine Programma Y
0	CY0
1	CY1
2	CY2 (entry point)
3	DY0
4	DY1
255	PY0

3. Sapendo che:
   - L'esecuzione di un programma avviene caricando inizialmente:
     - La pagina di codice con l'istruzione di partenza
     - La pagina di stack corrente
     - La prima pagina di dati
   - Il sistema utilizza la politica di lazy-loading
   - Il caricamento di pagine ulteriori è in demand paging

   L'esecuzione avviene secondo questa sequenza:

   1. La shell crea (fork) il processo P che esegue (exec) il programma X
   2. Il processo P crea (fork) il processo Q che esegue (exec) il programma Y
   3. Il processo Q crea (fork) il processo R

   Sapendo che:

   - Tutte le istruzioni necessarie alla creazione dei vari processi si trovaino nella pagina che contiene l'entry point
   - Non vengono effettuate scritture nelle pagine dati
   - Il programma Y, per creare un nuovo processo, richiede l'allocazione di una nuova pagina di stack.

   Si riporti nella tabella seguente lo stato della memoria fisica al termine della sequenza di operazioni di cui sopra.

NPF	Pagine dei processi
0	P:CX1
1	P:DX0
2	P:PX0
3	Q:CY2 / R:CY2
4	Q:DY0 / R:DY0
5	Q:PY0
6	Q:PY1
7	R:PY1

Infine, si riporti nella tabella seguente lo stato della MMU al termine della sequenza di operazioni di cui sopra

Processo	NPV	NPF
P	1	0
P	2	1
P	255	2
Q	2	3
Q	3	4
Q	255	5
Q	254	6
R	2	3
R	3	4
R	254	7

Domanda F

Si descriva il significato e l'utilizzo dei diversi registri e gruppi di registri del processore RISC-V.

Registri	Descrizione
zero	Si veda la teoria
ra	
sp	
t0-t6	
s0-a7	
s0-s11	