Politecnico di Milano – Sede di Cremona
Anno Accademico 2017/2018

Architettura dei Calcolatori e Sistemi Operativi

Esame – 17.07.2018

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 consideri la funzione delle libreria standard del C:

```
void* memcmp( const void* m1, const void* m2, size_t n )
```

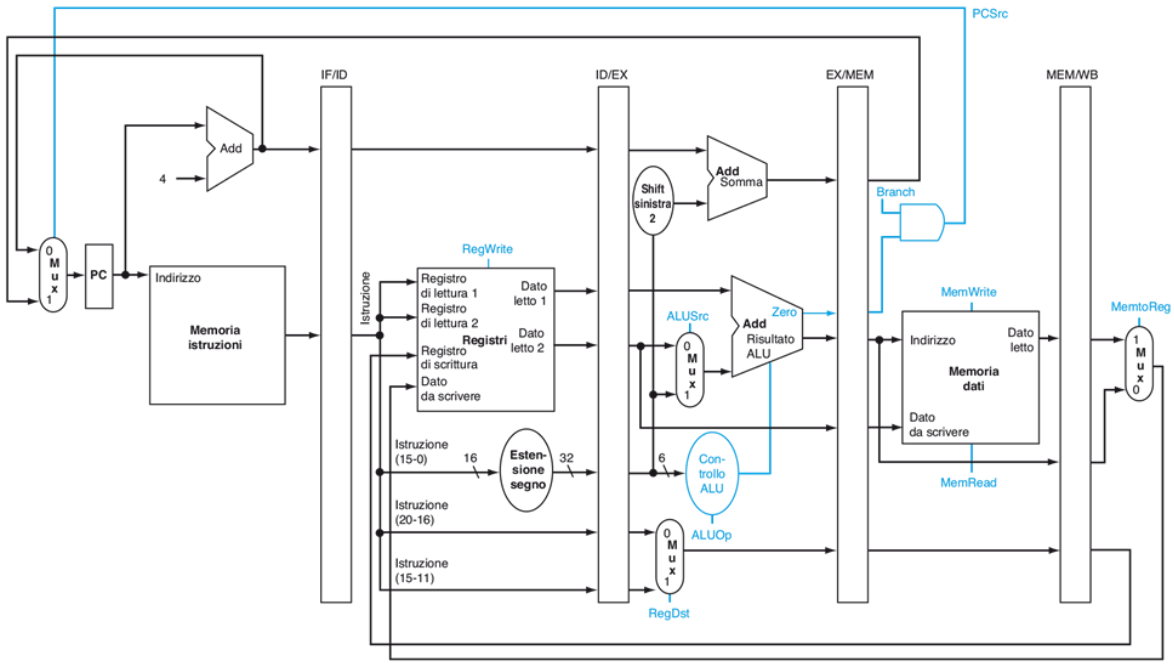
che confronta le due zone di memoria puntate da **m1** ed **m2**, per un massimo di **n** byte. La funzione restituisce 0 se le due zone di memoria sono identiche, un valore maggiore di zero la prima è maggiore della seconda ed un valore minore di zero nel caso contrario. Si dice che la prima è maggiore della seconda se nella prima posizione in cui differiscono, il byte nella prima area è maggiore del byte corrispondente nella seconda area. La definizione di minore è analoga. Come ulteriore indicazione si consideri il codice seguente che mostra un utilizzo corretto della funzione in esame.

```
        .data
mem0:   .byte    15, 24, 19, ...
mem1:   .byte    15, 24, 19, ...
        .text
main:
        la    $a0,mem0
        la    $a1,mem1
        li    $a2,37
        jal   memcmp
        bne   $v0, $zero, diff
        ...
```

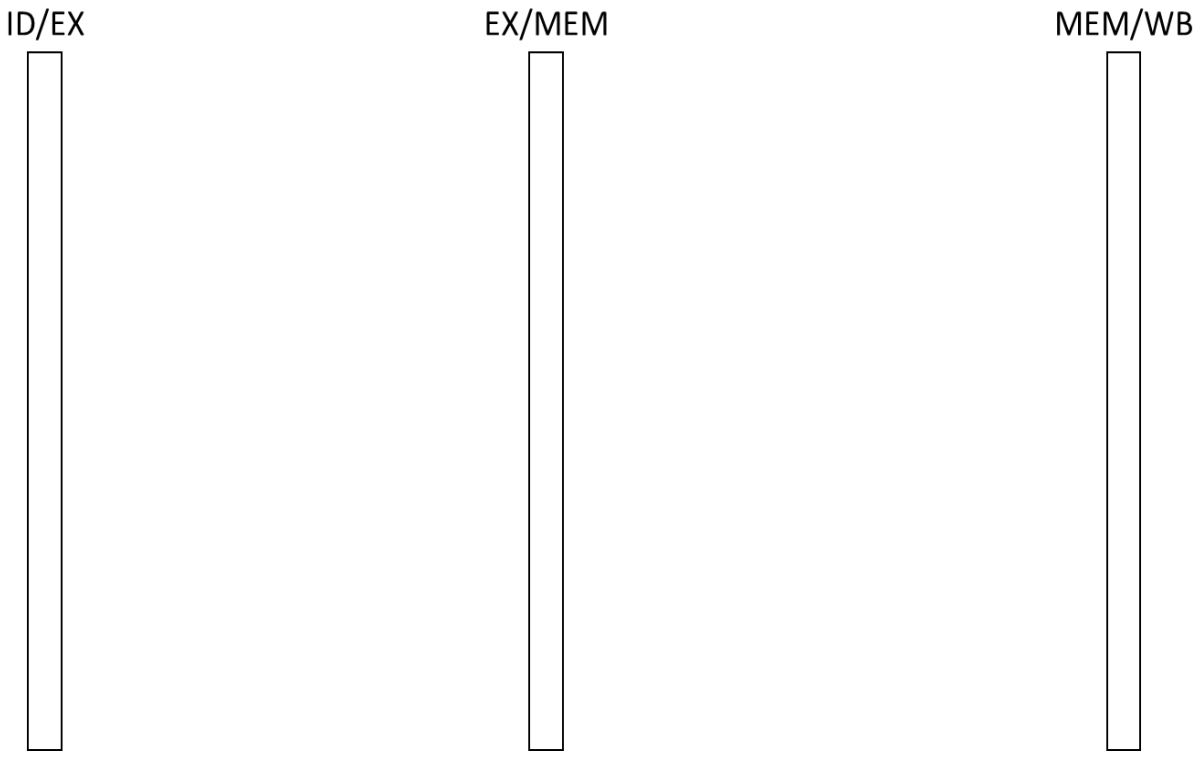
Si traduca la funzione **memcmp()** in assembly.

Domanda B

Dato lo schema di un processore MIPS con pipeline a 5 stadi:



si modifichi opportunamente l'architettura in modo da inserire l'ottimizzazione di forwarding, e cioè nello specifico i percorsi di forwarding EX/EX, MEM/EX e MEM/MEM. Si riportino in particolare gli stadi EX e MEM modificati.



Con riferimento alla nuova architettura modificata, sia dato il seguente codice assembly. Si evidenzi nella colonna FW quali percorsi di forwarding sono utilizzati tenendo conto di annotare solo le istruzioni che propagano un operando tramite tale tecnica e inserendo una o più tra le seguenti 4 diciture: N/A, EX/EX, MEM/EX, MEM/MEM.

Istruzione	**Forwarding**
`LW   $s0, 0($t0)`	
`ADDI $t0, $t0, 4`	
`ADDI $s1, $s0, 4`	
`SW   $s1, 0($t0)`	

Domanda C

Si consideri un sistema con uno spazio di indirizzamento di 1MByte ed una cache con le caratteristiche seguenti:

Associatività	Set associativa a 2 vie
Dimensione totale	8 KB
Dimensione linea	256 B (per ogni set)
Tempo di accesso	1 ns
Hit rate	98%

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

Struttura dell'indirizzo

Sapendo che:

- L'accesso alla memoria RAM avviene a parole di 16 bit
- Il tempo di accesso alla RAM in modalità normale è di 20 ns
- Il tempo di accesso alla RAM in modalità burst è
    - 40 ns per la prima parola
    - 10 ns per le parole successive

Si calcoli il tempo di accesso medio alla memoria.

Tempo medio di accesso

Considerando un programma in cui vengono effettuati gli accessi indicati nella tabella seguente, e supponendo di partire da una cache vuota, si completino le colonne mancanti indicando:

- Il tag el'index relativi all'accesso, riportandoli nel set scelto
- Lo stato dei bit di valid e dirty dopo l'operazione
- L'esito dell'operazione (hit/miss)

OP	ADDR	INDEX	SET 0			SET 1			HIT/MISS
			TAG	V	D	TAG	V	D	
R	0xAC330								
W	0xB0520								
R	0xB8342								
R	0xA0240								
W	0xB0512								
W	0xB8360								
R	0x00300								

Domanda D

Si consideri il seguente programma C:

```
#include <pthread.h>
#include <semaphore.h>

sem_t semaphore;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int shared = 10;

void* thread1( void* arg )
{
  int local1 = *((int*)arg) + shared;
  shared = shared / 2; // BREAKPOINT 1
  sem_post( &semaphore );
}

void* thread2( void* arg )
{
  int local2 = *((int*)arg);
  sem_wait( &semaphore );
  pthread_mutex_lock( &mutex );
  local2 = local2 + shared;
  shared = shared + 2; // BREAKPOINT 2
  pthread_mutex_unlock( &mutex );
  sem_post( &semaphore );
}

void* thread3( void* arg )
{
  int local3 = *((int*)arg);
  sem_wait( &semaphore );
  pthread_mutex_lock( &mutex );
  shared = shared + local3; // BREAKPOINT 3
  pthread_mutex_unlock( &mutex );
  sem_post( &semaphore );
}

int main( int argc, char* argv[] )
{
  pthread_t th1, th2, th3;
  int par1 = 1, par2 = 2, par3 = 3;

  sem_init( &semaphore, 0, 0 );

  pthread_create( &th1, NULL, thread1, (void*) &par1 );
  pthread_create( &th2, NULL, thread2, (void*) &par2 );
  pthread_create( &th3, NULL, thread3, (void*) &par3 );

  pthread_join( th1, NULL );
  pthread_join( th2, NULL );
  pthread_join( th3, NULL ); // BREAKPOINT 4

  return 0;
}
```

Si svolgano i seguenti punti:

1. Si completi la seguente tabella, riportando lo stato delle variabili **immediatamente dopo** il breakpoint indicato. Nel caso si passi più di una volta da un breakpoint considerare ogni sua esecuzione nel rispondere. Si utilizzi la seguente classificazione:

   - "ESISTE" se la variabile certamente esiste
   - "NON ESISTE" se sicuramente la variabile non esiste
   - "PUO' ESISTERE" se potrebbe esistere oppure non esistere

Breakpoint	**local1**	**local2**	**local3**	**shared**
Breakpoint 1				
Breakpoint 2				
Breakpoint 3				
Breakpoint 4				

2. Si completi la tabella seguente indicando se la variabile specificata può assumere nessuno, uno o più valori subito dopo i breakpoint specificati. Si utilizzi la seguente classificazione:

   - "N/A" se la variabile non può assumere nessun valore
   - "UN VALORE" se la variabile può assumere uno e un solo valore
   - "PIÙ VALORI" se la variabile può assumere più di un valore

Breakpoint	**local1**	**local2**	**local3**	**shared**
Breakpoint 1				
Breakpoint 2				
Breakpoint 3				
Breakpoint 4				

Domanda E

Si consideri un calcolatore dotato di sistema operativo Linux in cui valgono le seguenti specifiche:

- Le dimensioni dei blocchi sono di 512 byte
- Per l'apertura dei file è sempre necessario accedere a:
  - I-node di ogni cartella o file presente nel percorso
  - Blocco per il contenuto di ogni cartella presente nel percorso
  - Primo blocco dati del file

Dato il contenuto del seguente volume:

I-lista:	<0,dir,25> <1,dir,26> <2,dir,27> <3,norm,{100,101,102,103}> <4,dir,23> <5,norm,{200,201,202,203,204,205}> <6,dir,24> <7,dir,28> …
Blocco 23:	… <11,bash> …
Blocco 24:	… <20,pippo> <21,pluto> <22,paperino>
Blocco 25:	… <1,home> <8,bin> <9,tmp>
Blocco 26:	… <2,canid> <10,brandole> …
Blocco 27:	… <7,polimi> <12,work> <13,private>
Blocco 28:	… <14,secret.dat> <3,acso.dat> <5,rl.dat> …

1. Per ciascuna delle chiamate di sistema sotto riportate, si indichi la sequenza di accessi agli I-Node e ai blocchi (del tipo: I-Node X oppure Blocco Y).

Chiamata di sistema	Sequenza di accessi
fd = **open**("/home/canid/polimi/acso.dat", O_RDWR)	
fd2 = **open**("/home/canid/polimi/rl.dat", O_RDWR)	
read(fd, buffer, 555)	

2. Si supponga ora di creare un nuovo file di dimensione 1KB con la seguente istruzione:

fd3 = **open**(“/home/canid/polimi/exam.dat”, O_RDWR | O_CREAT, S_IRUSR | S_IRGRP | S_IROTH)

Riportare qui sotto le modifiche effettuate al contenuto del volume dopo la creazione del file stesso.

Inoltre, data la seguente struttura dei blocchi liberi specificare quali blocchi saranno occupati dal file *exam.dat* utilizzando le politiche di allocazione:

- **Contigua – First Fit**

- **Contigua – Best Fit**

- **Contigua – Worst Fit**

Spazio fisico disponibile

Blocco	…	400	401	402	403	…	500	501	…	700	701	702	703	704	705	…
Stato																

Domanda F

Si consideri il seguente codice C, relativo al file **somecode.c**.

```
#define MAX_LEN   32

typedef struct
{
    float t[3][3];
} Tensor_t;

Tensor_t         T1;
static Tensor_t  T2;
static const int order = 17;

float tensor_trace(Tensor_t* t);

int process(int size)
{
    static int initval = 12;
    char* progname = "Tensor";
    int*    data;

    data = (int*)malloc(128*sizeof(int));
    ...
}
```

Si completi la seguente tabella, indicando per ogni simbolo:

- Il tipo
- Il segmento di memoria in cui è allocato, oppure N/A se il simbolo non richiede allocazione di memoria
- La sua dimensione in byte, supponendo di riferirsi ad un sistema a 32bit
- La visibilità da parte di un file diverso da **somecode.c**

Simbolo	Tipo	Segmento	Dimensione	Visibilità