Politecnico di Milano – Sede di Cremona
Anno Accademico 2022/2023

Architettura dei Calcolatori e Sistemi Operativi

Esame – 15.02.2024

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 sviluppi in codice assembly la funzione:

```
void bcdadd( unsigned char* r, unsigned char* a, unsigned char* b )
```

che prende in ingrsso due valori interi **a** e **b** e produce come risultato la somma di tali valori memorizzandola in **r**. I parametri **a**, **b** ed **r** sono vettori di byte di dimensione fissa e pari a 8, passati per alla funzione per indirizzo. Ogni vettore contiene le cifre decimali del valore, una per byte, ordinate dalla più significativa alla meno significativa.
Facendo riferimento al codice che segue, il vettore a rappresenta il valore 369857. Come si nota la cifra meno significativa è nella posizione più a destra, cioè l'elemento **a[7]** mentre la più significativa, il 3, è in posizione **a[2]**. Le prime due posizioni sono rimepite con degli zeri di padding poiché le cifre devono essere allineate alla destra del vettore. In modo simile, il vettore b rappresenta il valore 2574635
Nel seguito è riportato un codice di esempio che mostra come la funzione deve essere realizzata

```
        .data
a:      .byte 0, 0, 3, 6, 9, 8, 5, 7
b:      .byte 0, 2, 5, 7, 4, 6, 3, 5
r:      .space 8

        .text
main:   la    $a0, r
        la    $a1, a
        la    $a2, b
        jal   bcdadd
        li    $v0, 10
        syscall


bcdadd: add   $t0, $a0, 7       # &r[7]
        add   $t1, $a1, 7       # &a[7]
        add   $t2, $a2, 7       # &b[7]
        addi  $t3, $zero, 8     # n = 8
        add   $t4, $zero, $zero # carry = 0
        addi  $t7, $zero, 10    # 10

loop:   beq   $t3, $zero, end   # if n = 0 finish

        lb    $t5, ($t1)        # *a
        lb    $t6, ($t2)        # *b

        add   $t6, $t6, $t5     # t = *a + *b
        add   $t6, $t6, $t4     # t = t + carry
        div   $t6, $t7          # t div 10
        mflo  $t4               # carry = t / 10
        mfhi  $t6               # sum = t % 10

        sb $t6, ($t0)           # *r = sum

        addi $t0, $t0, -1       # r--
        addi $t1, $t1, -1       # a--
        addi $t2, $t2, -1       # b--
        addi $t3, $t3, -1       # n--
        j    loop
end:    jr $ra
```

Domanda B

Si sviluppi in C un modulo STACK che implementa uno stack di interi secondo le seguenti specifiche:

1. Lo stack deve poter essere usato in modo affidabile da più thread concorrenti
2. Lo stack deve poter essere usato in modo affidabileda più processi concorrenti
3. La dimensione massima dello stack è fissa e pari a 128 elementi
4. L'accesso al contenuto dello stack deve poter avvenire unicamente mediante le funzioni esposte dall'interfaccia del modulo, riportata di seguito.

File stack.h

```
#ifndef STACK_H
#define STACK_H

// Initializes the stack
void STACK_Init( void );

// Pushes a new value on the stack
// Returns 1 on success, 0 otherwise
int STACK_Push( int val );

// Pops a value from the stack
// Returns 1 on success, 0 otherwise
int STACK_Pop( int* val );

// Empties the stack
void STACK_Clear( void );

#endif
```

File stack.c

```
#include <pthread.h>

#include "stack.h"

static int data[128] = { 0 };
static int ptr       = 0;
static pthread_mutex_t mutex;

void STACK_Init( void )
{
    pthread_mutex_lock(&mutex);
    ptr = 0
    pthread_mutex_unlock(&mutex);
}
```

```
int STACK_Push( int val )
{
    int retval = 0;
    pthread_mutex_lock(&mutex);
    if( ptr < 127 )
    {
        data[ptr++] = val;
        retval      = 1;
    }
    pthread_mutex_unlock(&mutex);
    return retval;
}

int STACK_Pop( int* val )
{
    int retval = 0;
    pthread_mutex_lock(&mutex);
    if( ptr > 0 )
    {
        *val   = data[--ptr];
        retval = 1;
    }
    pthread_mutex_unlock(&mutex);
    return retval;
}

void STACK_Clear( void )
{
    STACK_Init();
}
```

Domanda C

Si consideri un piccolo sistema embedded con uno spazio di indirizzamento complessivo pari a 32 GByte ed due memorie cache con le caratteristiche seguenti:

	D-CACHE	**I-CACHE**
Associatività	Set associativa a 8 vie	Diretta
Dimensione totale	256 KB	64 K
Dimensione linea	256 B	128 B
Tempo di accesso	2 ns	1 ns
Hit rate	99 %	98 %

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

Struttura dell'indirizzo D-CACHE

Offset = log2 256 = 8 bit

Index = log2 [256 K / (8 * 256)] = log2 128 = 7 bit

Tag = log2 32G – 8 – 7 = 35 – 8 – 7 = 20 bit

Struttura dell'indirizzo I-CACHE

Offset = log2 128 = 7 bit

Index = log2 [64 K / 128] = log2 64K – log2 128 = 16 – 7 = 9 bit

Tag = log2 32G – 8 – 7 = 35 – 8 – 7 = 19 bit

Sapendo che:

- L'accesso alla memoria avviene a parole di 128 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 di accesso medio alla memoria.

Tempo medio di accesso ai dati

TD = 2ns * 0.99 + [2ns + 150 ns + (256 / 16 – 1) * 50ns] * 0.01 = 11 ns

Tempo medio di accesso alle istruzioni

TD = 1ns * 0.98 + [1ns + 150 ns + (128 / 16 – 1) * 50ns] * 0.02 = 11 ns

Domanda D

Si consideri un file system Linux organizzato come mostra la figura seguente.

I-node list:
<0,dir,6>, <1,dir,12> <2,dir,81> <3,dir,13> <5,norm,{94,95}> <7,dir,31> <11,dir,63> <111,norm,{200,201}> <113,norm, 250>

Blocco 006: …<1,dir1> <2, dir2 > …
Blocco 012: …<3,mydir > <5,config > …
Blocco 013: …<7,secret><10,report>
Blocco 031: …<111,analysis><11,plans>…
Blocco 063: …<113,action>
…
Blocco 094: …dati…
Blocco 095: …dati…
Blocco 200: …dati…
Blocco 201: …dati…
Blocco 250: …dati…
…

Si noti che l'i-node associato al alla root directory "/" ha i-number uguale 0. Valgono inoltre le seguenti specifiche:

- per tutte le operazioni su file la dimensione di un blocco è di 1KB ed il sistema deve garantire che il blocco contenente la posizione corrente sia in memoria principale.
- La scrittura dei dati su file avviene quando viene eseguita la richiesta, senza buffering.

Dato il seguente codice:

```
fd1 = open ("/dir1/config", O_RDWR);
fd2 = open ("/dir1/mydir/secret/analysis", O_RDWR);
read (fd2, buf2, 1500);
fd3 = open ("/dir1/mydir/secret/plans/action", O_RDWR);
write (fd1, buf1, 1200);
close (fd1);
lseek (fd2,-501L, SEEK_CUR);
read (fd3, buf3, 250);
close (fd2);
close (fd3);
```

Si completi la tabella seguente indicando l'accesso ai blocchi dati con B<n>, agli i-node con I<n>, aggiungendo per ogni accesso il suffisso M o D per indicare se l'accesso è in memoria o su disco

Chiamata di sistema	Pos. corr.	Sequenza accessi
`fd1 = open (”/dir1/config”, O_RDWR);`	0	I0D B6D I1D B12D I5D B94D
`fd2 = open (”/dir1/mydir/secret/analysis”, O_RDWR);`	0	I0D B6D I1D B12D I3D B13D I7D B31D I111D B200D
`read (fd2, buf2, 1500);`	1500	B200M I111M B201D
`fd3 = open (”/dir1/mydir/secret/plans/action”, O_RDWR);`	0	I0D B6D I1D B12D I3D B13D I7D B31D I11D B63D I113D B250D
`write (fd1, buf1, 1200);`	1200	B94D I50M B95D
`close (fd1);`	//	//
`lseek (fd2,-501L, SEEK_CUR);`	999	I111M B200D
`read (fd3, buf3, 250);`	250	B250M
`close (fd2);`	//	//
`close (fd3);`	//	//

Domanda E

Si consideri la sequenza di istruzioni sotto riportata:

# ISTR	ISTRUZIONE
1	`lw  $3, ($1)`
2	`sw  $2, ($3)`
3	`sub $1, $1, $2`
4	`add $1, $2, $3`
5	`sw  $3, 4($1)`

Si supponga che la microarchitettura MIPS destinata ad eseguire il codice in esame **non** sia dotata di nessun percorso di propagazione.

1. Si identifichino tutte le dipendenze dati e si completi la tabella seguente,

#ISTR	#ISTR da cui dipende	Registro coinvolto	Conflitto (S/N)	Numero stalli
2	1	$3	S	2
4	3	$1	N	0
5	3	$1	N	0
5	4	$1	S	2

2. Si completi il seguente diagramma riportando l'esecuzione effettiva del codice

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`lw  $3,($1)`	F	D	E	M	W										
`sw  $2, ($3)`		F	S	S	D	E	M	W							
`sub $1,$1,$2`					F	D	E	M	W						
`add $1,$2,$3`						F	D	E	M	W					
`sw  $3,4($1)`							F	S	S	D	E	M	W		

Si supponga ora che la microarchitettura MIPS destinata ad eseguire il codice in esame sia dotata dei percorsi di propagazione EX/EX, MEM/EX e MEM/MEM.

3. Si completi il seguente diagramma riportando l'esecuzione effettiva del codice

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`lw  $3,($1)`	F	D	E	M	W										
`sw  $2, ($3)`		F	S	D	E	M	W								
`sub $1,$1,$2`				F	D	E	M	W							
`add $1,$2,$3`					F	D	E	M	W						
`sw  $3,4($1)`						F	D	E	M	W					

Domanda F

Si spieghi il concetto di memoria segmentata descrivendo la struttura di massima un file elf e la corrispondente organizzazioned ella memoria quando caricato.

Instruction		Description	Operation	Type	Opcode	Funct
add	rd,rs,rt	Add	rd	R	000000	100000
sub	rd,rs,rt	Subtract	rd = rs - rt	R	000000	100010
addi	rt,rs,imm	Add Immediate	rt = rs + imm	I	001000	
addu	rd,rs,rt	Add Unsigned	rd = rs + rt	R	000000	100001
subu	rd,rs,rt	Subtract Unsigned	rd = rs - rt	R	000000	100011
addiu	rt,rs,imm	Add Immediate Unsigned	rt = rs + imm	I	001001	
mult	rs,rt	Multiply	{hi, lo} = rs * rt	R	000000	011000
div	rs,rt	Divide	lo = rs / rt; hi = rs % rt	R	000000	011010
multu	rs,rt	Multiply Unsigned	{hi, lo} = rs * rt	R	000000	011001
divu	rs,rt	Divide Unsigned	lo = rs / rt; hi = rs % rt	R	000000	011011
mfhi	rd	Move From Hi	rd = hi	R	000000	010000
mflo	rd	Move From Lo	rd = lo	R	000000	010010
and	rd,rs,rt	And	rd = rs & rt	R	000000	100100
or	rd,rs,rt	Or	rd = rs | rt	R	000000	100101
nor	rd,rs,rt	Nor	rd = ~(rs | rt)	R	000000	100111
xor	rd,rs,rt	Exclusive Or	rd = rs ^ rt	R	000000	100110
andi	rt,rs,imm	And Immediate	rt = rs & imm0	I	001100	
ori	rt,rs,imm	Or Immediate	rt = rs | imm0	I	001101	
xori	rt,rs,imm	Exclusive Or Immediate	rt = rs ^ imm0	I	001110	
sll	rd,rt,sh	Shift Left Logical	rd = rt << sh	R	000000	000000
srl	rd,rt,sh	Shift Right Logical	rd = rt >>> sh	R	000000	000010
sra	rd,rt,sh	Shift Right Arithmetic	rd = rt >> sh	R	000000	000011
sllv	rd,rt,rs	Shift Left Logical Variable	rd = rt << rs	R	000000	000100
srlv	rd,rt,rs	Shift Right Logical Variable	rd = rt >>> rs	R	000000	000110
srav	rd,rt,rs	Shift Right Arithmetic Variable	rd = rt >> rs	R	000000	000111
slt	rd,rs,rt	Set if Less Than	rd = rs < rt ? 1 : 0	R	000000	101010
sltu	rd,rs,rt	Set if Less Than Unsigned	rd = rs < rt ? 1 : 0	R	000000	101011
slti	rt,rs,imm	Set if Less Than Immediate	rt = rs < imm ? 1 : 0	I	001010	
sltiu	rt,rs,imm	Set if Less Than Immediate Unsigned	rt = rs < imm ? 1 : 0	I	001011	
j	addr	Jump	PC = PC & 0xF0000000 | (addr<< 2)	J	000010	
jal	addr	Jump And Link	ra = PC + 8; PC = PC & 0xF0000000 | (addr<<2)	J	000011	
jr	rs	Jump Register	PC = rs	R	000000	001000
jalr	rs	Jump And Link Register	ra = PC + 8; PC = rs	R	000000	001001
beq	rt,rs,imm	Branch if Equal	if (rs == rt) PC += 4 + (imm << 2)	I	000100	
bne	rt,rs,imm	Branch if Not Equal	if (rs != rt) PC += 4 + (imm << 2)	I	000101	
syscall		System Call		R	000000	001100
lui	rt,imm	Load Upper Immediate	rt = imm << 16	I	001111	
lb	rt,imm(rs)	Load Byte	rt = SignExt(MB[rs + imm])	I	100000	
lbu	rt,imm(rs)	Load Byte Unsigned	rt = MB[rs + imm] & 0xFF	I	100100	
lh	rt,imm(rs)	Load Half	rt = SignExt(MH[rs + imm])	I	100001	
lhu	rt,imm(rs)	Load Half Unsigned	rt = MH[rs + imm] & 0xFFFF	I	100101	
lw	rt,imm(rs)	Load Word	rt = MW[rs + imm]	I	100011	
sb	rt,imm(rs)	Store Byte	MB[rs + imm] = rt	I	101000	
sh	rt,imm(rs)	Store Half	MH[rs + imm] = rt	I	101001	
sw	rt,imm(rs)	Store Word	M4[rs + imm] = rt	I	101011	
ll	rt,imm(rs)	Load Linked	rt = MW[rs + imm]	I	110000	
Sc	rt,imm(rs)	Store Conditional	MW[rs + imm] = rt; rt = atomic ? 1 : 0	I	111000	

Pseudo instruction		Description	Operation	Type	Opcode	Funct
bge	rx,ry,imm	Branch if Greather Than or Equal	if (rs >= rt) PC += 4 + (imm << 2)			
bgt	rx,ry,imm	Branch if Greather Than	if (rs > rt) PC += 4 + (imm << 2)			
ble	rx,ry,imm	Branch if Less Than or Equal	if (rs <= rt) PC += 4 + (imm << 2)			
blt	rx,ry,imm	Branch if Less Than	if (rs < rt) PC += 4 + (imm << 2)			
la	rx,label	Load Address	rx = Address(label)			
li	rx,imm	Load 32-bit Immediate	rx = imm			
move	rx,ry	Move	rx = ry			
nop		No Operation				

Register Number	Register name
$0	$zero
$1	$at
$2 – $3	$v0–$v1
$4 – $7	$a0–$a3
$8 – $15	$t0–$t7
$16 – $23	$s0–$s7
$24 – $25	$t8–$t9
$26 – $27	$k0–$k1
$28	$gp
$29	$sp
$30	$fp
$31	$ra
	hi
	lo
	pc