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

Architettura dei Calcolatori e Sistemi Operativi

Esame – 22.01.2024

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 MIPS la funzione:

```
void compress( char* dst, char* src )
```

che prende in ingrsso il puntatore ad una stringa ASCII terminata **src** ed il puntatore ad uno spazio di memoria **dst**, che supponiamo essere di dimensioni sufficienti al problema (non sono cioè richiesti controlli). La funzione copia la stringa **src** nello spazio **dst** sostituendo le sequenze di più spazi con un singolo spazio. Nel seguito è riportato uno stralcio di codice che ne mostra lutilizzo.

```
        .data
dst:    .space  64
src:    .asciiz ”The     lazy  fox         jumped   over the   tree”

        .text
main:
        la      $a0, dst
        la      $a1, src
        jal     compress

print:  addi    $v0, $zero, 4
        syscall

exit:   addi    $v0, $zero, 4
        syscall


compress:
        # Initialization
        add     $t0, $a0, $zero         # t0 = dst
        add     $t1, $a1, $zero         # t1 = src
        addi    $t2, $zero, 32          # t2 = ' '

        # Reads one character
load:   lb      $t3, ($t1)              # t3 = src[i]

        # Saves it in the destination
store:  sb      $t3, ($t0)              # dst[i] = src[i]

        # Moves src/dst pointer forward by one byte
        addi    $t0, $t0, 1
        addi    $t1, $t1, 1

        # If '\0' is found, string is finished
        beq     $t3, $zero, end         # End of string

        # If this is a space enters the skip loop
        beq     $t3, $t2, skip
        b       load

skip:   addi    $t1, $t1, 1
        lb      $t3, ($t1)
        beq     $t3, $t2, skip

        # On the first non-space goes back to the main copy loop
        b       store

        # Return
end:    jr      $ra
```

Domanda B

Si scriva il codice C di un modulo "**counter**", costituito dai due file **counter.h** e **counter.c** secondo le seguenti specifiche:

- Il modulo non espone alcuna variabile globale
- Il modulo dispone di due funzioni pubbliche
  - **int get(void)**
  - **void step(void)**
- La funzione **get()** restituisce il valore corrente del contatore
- La funzione **step()** incrementa il contatore di una unità
- Il valore iniziale del contatore è 0

Il modulo deve essere implementato in modo da essere "thread safe" ovvero utilizzabile da un programma organizzato su più thrread.

File: counter.h

```
#ifndef COUNTER_H
#define COUNTER_H

int  get ( void );
void step( void );

#endif
```

File: counter.c

```
#include "counter.h"
#include <pthread.h>

static int                  counter = 0;
static pthread_mutex_t      mutex;

int get( void )
{
      return counter;
}

void step( void )
{
      pthread_mutex_lock( &mutex );
      counter++;
      pthread_mutex_unlock( &mutex );
}
```

Domanda C

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

	D-CACHE	**I-CACHE**
Associatività	Completamente associativa	Diretta
Dimensione totale	16 KB	32 KB
Dimensione linea	128 B	128 B
Tempo di accesso	2 ns	1 ns
Hit rate	99.9 %	98 %

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

Struttura dell'indirizzo D-CACHE

Memoria: 32 MB => log2 32MB = 25 bit

Offset: log2 128 B = 7 bit

Tag: 25 – 7 bit = 18 bit

Struttura dell'indirizzo I-CACHE

Memoria: 32 MB => log2 32MB = 25 bit

Offset: log2 128 B = 7 bit

Index: log2 (32 KB / 128 B) = 15 – 7 = 8

Tag: 25 – 8 – 7 bit = 10 bit

Sapendo che:

- Le istruzioni risiedono in una memoria FLASH
- I dati sono memorizzati in RAM
- L'accesso alla memoria sia RAM sia FLASH avviene a parole di 32 bit
- Il tempo di accesso alla RAM è di 20 ns
- Il tempo di acesso alla FLASH è di 40 ns

Si calcoli il tempo di accesso medio alla memoria.

Tempo medio di accesso ai dati

Td,miss = (128 / 4) * 20 ns = 640 ns

Td,hit = 2 ns

Td = 0.999 * 2ns + 0.001 (2ns + 640 ns) = 2.640 ns

Tempo medio di accesso alle istruzioni

Ti,miss = (128 / 4) * 40 ns = 1280 ns

Td,hit = 1 ns

Td = 0.98 * 1ns + 0.02 (1ns + 1280 ns) = 26.6 ns

Supponendo che il 10% delle istruzioni eseguite da un certo programma siano di load/store, si calcoli il tempo di accesso medio per istruzione con e senza cache ed il miglioramento delle prestazioni.

Tempo di accesso medio con cache:

T = (100 * 26.6 + 10 * 2.64) / 100 = (2660 + 26.4) / 100 = 26.86 ns

Tempo di accesso medio senza cache:

T = (100 * 40 + 10 * 20) / 100 = (4000+ 200) / 100 = 42.00 ns

Miglioramento delle prestazioni:

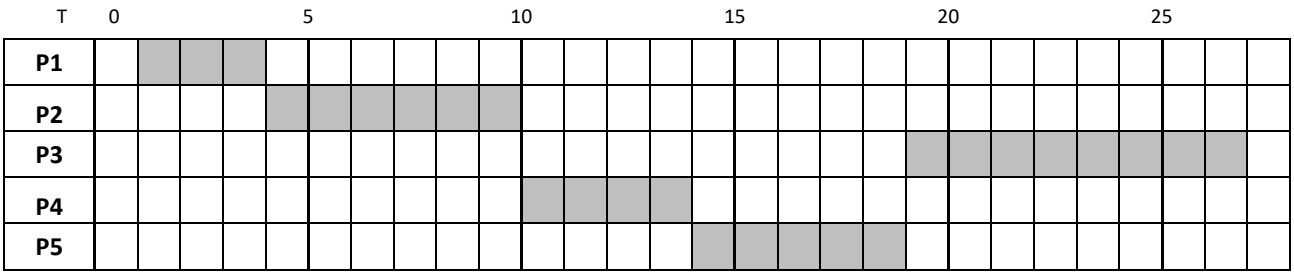
R = 1 - 26.86 / 42.00 = 1 – 0.6715 = 32.85 %

Domanda D

Dati i seguenti processi:

Processo	Arrival Time	Execution time
P1	1	3
P2	3	6
P3	5	8
P4	7	4
P5	8	5

Si esegua lo scheduling secondo l'algoritmo HRRN (non-preeemptive), indicando i valori del response ratio relativi a tutti i momenti in cui lo scheduler lo richiede.



Response ratio

T=0

Non ci sono processi, il processore è idle

T=1

Solo P1 è ready e viene schedulato. P1 termina a T=4

T=4

Solo P2 è ready e viene schedulato. P2 termina a T=10

T=10

Nella coda dei processi ready ci sono P3, P4 e P4. Per scegliere si deve calcolare response ratio (Wait+Exec/Exec) per ognuno di questi processi:

P3 = (5+8)/8 = 1.625

P4 = (3+4)/4 = 1.75

P5 = (2+5)/5 = 1.4

Il response ratio maggiore è quello di P4 che viene schedulato. P4 termina T=14

T=14

In coda sono rimasti P3 e P5. Si calcola il respone ratio

P3 = (9+8)/8 = 2.125

P5 = (6+5)/5 = 2.2

Il response ratio maggiore è quello di P5 che viene schedulato. P5 termina T=19

T=19

In coda è rimasto solo P3, che viene schedulato. P3 termina a T=27

Domanda E

Si consideri un sistema di memoria con uno spazio di indirizzamento virtuale di 16 MByte ed una dimensione di pagina pari a 8 KByte.

1. Si indichino le dimensioni in bit di:

   Indirizzo virtuale: 16 MB => 24 bit

   Numero di pagina virtuale: 16 MB / 8 KB = 2K = 11 bit

   Offset: 8 KB = 13 bit

2. Si completi la seguente tabella, riportanto numero di pagina virtuale e e spiazzamento sia in forma binaria, sia esadecimale.

Indirizzo virtuale	Numero di pagina virtuale		Offset	
	Hex	Bin	Hex	Bin
0x6ADAF6	356	011 0101 0110	1AF6	1 1010 1111 0110
0x8AF480	457	100 0101 0111	1480	1 0100 1000 0000
0x400882	200	010 0000 0000	0882	0 1000 1000 0010
0x9C34A4	4E1	100 1110 0001	14A4	1 0100 1010 0100
0x00ACC1	005	000 0000 0101	0CC1	0 1100 1100 0001

Domanda F

Si spieghi il concetto di branch prediction statica e si discuta la differenza tra di prestazioni tra una architettura priva di branch prediction ed una dotata di una branch prediction unit che la realizza la politica “always not-taken”. Per la valutazione consideri sia il caso di predizione corretta, sia di predizione errata.

Vedi testo

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