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

Architettura dei Calcolatori e Sistemi Operativi

Esame – 26.06.2023

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 linguaggio assembly MIPS la funzione:

```
void itob( unsigned int n, char* s )
```

che prende in ingresso un intero n ed un puntatore s ad una stringa, allocata a carico del chiamante di dimensione fissa e pari 33 byte. La funzione converte il valore intero senza segno n nella sua rappresentazione sotto forma di stringa binaria. La stringa di uscita ha lunghezza fissa paria 33 caratteri e deve essere terminata dal carattere '\0'. Il codice seguente mostra un possibile utilizzo della funzione.

```
        .data
N:      .word   0x12345678
S:      .space  33

        .text
main:   la      $a0, N
        lw      $a0, ($a0)
        la      $a1, S
        jal     itob
        move    $a0, $a1
        li      $v0, 4
        syscall                     # Print string
        li      $v0, 10
        syscall                     # Exit

itob:   add     $t0, $a0, $zero         # t0 = N
        add     $t1, $a1, 32        # t1 = &S[32]
        sb      $zero, ($t1)        # S[32] = '\0'
        addi    $t1, $t1, -1        # t1--
        addi    $t2, $zero, 32          # i = 32
loop:   andi    $t3, $t0, 1         # t3 = N & 0x0000 0001
        addi    $t3, $t3, 48        # t3 = t3 + '0'
        sb      $t3, ($t1)          # *t1 = t3
        addi    $t1, $t1, -1        # i--
        addi    $t2, $t2, -1        # i--
        srl     $t0, $t0, 1         # N <<= 1
        bne     $t2, $zero, loop
        jr      $ra
```

Domanda B

Si consideri il seguente programma in esecuzione su una macchina a 32 bit con organizzazione della memoria secondo il modello big-endian.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
      char a;
      char b;
      int  c;
      char d;
      char e[3];
} record_t;

int main()
{
      record_t r = { 10, 20, 30, 2, { 1, 0, 0 } };
      char*   pc = (char*)&r;
      int*    pi = (int*) &r;

      printf( "1: %d\n", *(pc)   );
      printf( "2: %d\n", *(pc+1) );
      printf( "3: %d\n", *(pc+4) );
      printf( "4: %d\n", *(pc+7) );

      printf( "5: %d\n",   pi[0] );
      printf( "6: %d\n", *(pi+1) );
      printf( "7: %d\n", *(pi+2) );

      printf( "8: %d\n", (int)((long int)&r.d - (long int)&r.b) );

      return 0;
}
```

Si riporti qui di seguito quanto stampato dal programma, qualora univocamente determinabile. In caso contrario si riporti la dicitura “N/A”.

La struttura dati in memoria è la seguente:

Offset	0	1	2	3	4	5	6	7	8	9	10	11
Campo	a	b	padding		c				d	e[0]	e[1]	e[2]
Valore	0x0A	0x14	0x??	0x??	0x1E	0x00	0x00	0x00	0x02	0x01	0x00	0x00
Accesso con pc	*(pc)	*(pc+1)			*(pc+4)			*(pc+7)				
Accesso con pi	pi[0]				*(pi+1)				*(pi+2)			

Ne consegue che i valori stampati sono:

Passo / Risultato
1: 10
2: 20
3: 30
4: 0
5: N/A (in quanto i byte di paading 2 e 3 non sono noti
6: 30
7: 258
8: 7 in quanto è l’offset tra d (cioè 8) e b (cioè 1)

Domanda C

Si consideri un sistema con uno spazio di indirizzamento di 256 MByte ed due memorie cache con le caratteristiche seguenti:

	D-CACHE	**I-CACHE**
Associatività	Set associativa a 8 vie	Diretta
Dimensione totale	32 KB	128 K
Dimensione linea	128 B (per ogni via)	256 B
Tempo di accesso	2 ns	1 ns
Hit rate	99.5 %	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(128) = 7

Index = log2(32K / (8 *128)) = log2(32) = 5

Tag = log2(256M) – 7 – 5 = 28 – 12 = 16

Struttura dell'indirizzo I-CACHE

Offset = log2(256) = 8

Index = log2(128K / 256) = log2(128k) – 8 = 17 – 8 = 9

Tag = log2(256M) – 8 – 9 = 28 – 17 = 11

Sapendo che:

- L'accesso alla memoria RAM avviene a parole di 128 bit
- Il tempo di accesso alla RAM in modalità normale è di 90 ns
- Il tempo di accesso alla RAM in modalità burst è
  - 180 ns per la prima parola
  - 40 ns per le parole successive

Si calcoli il tempo di accesso medio alla memoria.

Tempo medio di accesso ai dati

Td = 0.995 x 2ns + 0.005 x (2ns + 180ns + (128/16 – 1) x 40ns) = 4.30 ns

Tempo medio di accesso alle istruzioni

Ti = 0.98 x 1ns + 0.02 x (1ns + 180ns + (256/16 – 1) x 40ns) = 16.60 ns

Supponendo che il 15% 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:

$Tcc = (100 \times Ti + 15 \times Td) / 100 = 17.245$ n

Tempo di accesso medio senza cache:

$Tm = 90$
$Tsc = (100 \times Tm + 15 \times Tm) / 100 = 103.5$ ns

Miglioramento delle prestazioni:

$r = 1 - Tcc/Tsc = 83.3$ %

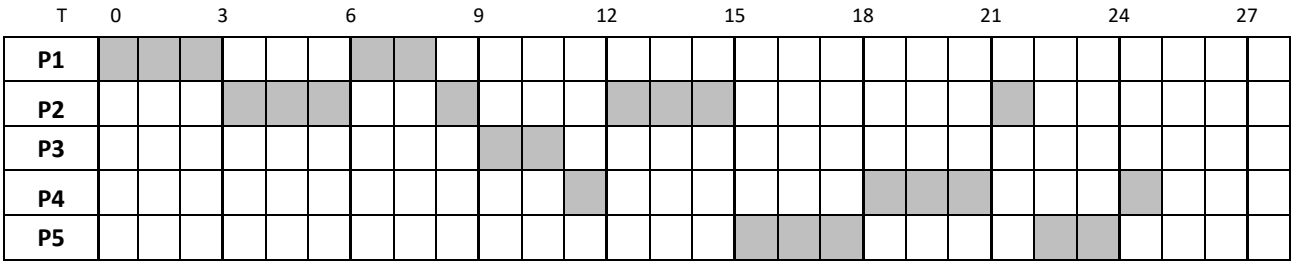
Domanda D

Dati i seguenti processi:

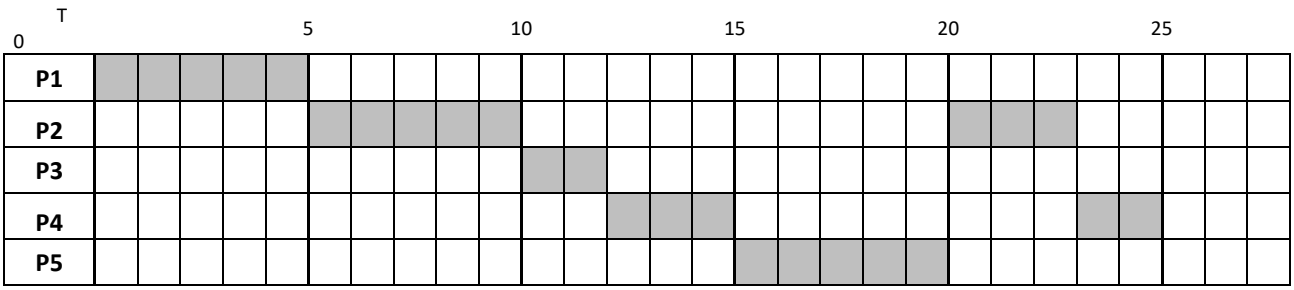
Processo	Arrival Time	Execution time
P1	0	5
P2	1	8
P3	7	2
P4	8	5
P5	10	5

Si esegua lo scheduling con algoritmo round-robin nei due casi descritti di seguito. Si tenga presente che nel caso in cui quando un processo esce dallo stato di running ve ne sia uno che diventa ready, quest'ultimo entra per primo nella coda.

Round-robin con quanto di tempo pari a 3 unità



Round-robin con quanto di tempo pari a 5 unità



Domanda E

Si consideri la sequenza di istruzioni MIPS riportata qui di seguito.

```
A:  lw    $t0, 8($a0)
    addi  $t0, $t0, 4
    ori   $t1, $zero, 1
    bne   $t1, $zero, B
    lw    $t2, 0($a1)
    sw    $t2, 4($a1)
B:  addi  $a0, $a1, 32
    sw    $t1, ($a0)
```

Si simuli l'esecuzione su una architettura pipelined senza alcuna ottimizzazione.

In primo luogo si nota che prima del branch $t1 vale 1, quindi la condizione del salto è vera.

	1				5					10					15					20					25
A:lw $t0,8($a0)	F	D	E	M	W																				
addi $t0,$t0,4		F	S	S	D	M	W																		
ori $t1,$zero,1					F	D	E	M	W																
bne $t1,$zero,B						F	S	S	D	E	M	W														
lw $t2,0($a1)									F	S	S	D	E	M	W			Diventa una NOP							
sw $t2,4($a1)																		Non eseguita							
B:addi $a0,$a1,32												F	D	E	M	W									
sw $t1,($a0)													F	S	S	D	M	W							

In questo caso la destinazione del salto è disponibile alla fine dello stadio di M, cioè al ciclo 12. L'istruzione lw entra nella pipeline e viene stallata fino al ciclo 11, dopo di che viene invalidata (trasformata in NOP) ed il controllo passa alla etichetta B.

Si simuli l'esecuzione su una architettura con register forwarding e branch prediction statica di tipo always-not-taken.

	1				5					10					15					20					25
A:lw $t0,8($a0)	F	D	E	M	W																				
addi $t0,$t0,4		F	S	D	E	M	W																		
ori $t1,$zero,1				F	D	E	M	W																	
bne $t1,$zero,B					F	D	E	M	W																
lw $t2,0($a1)						F	D	E	M	W								Diventa una NOP							
sw $t2,4($a1)																		Non eseguita							
B:addi $a0,$a1,32							F	D	E	M	W														
sw $t1,($a0)								F	D	E	M	W													

Con le ottimizzazioni la destinazione del salto è disponibile alla fine dello stadio di D cioè al ciclo 8. L'istruzione lw entra nella pipeline e viene invalidata (trasformata in NOP) ed il controllo passa alla etichetta B.

Si calcoli il miglioramento delle prestazioni dell'architettura ottimizzata rispetto a quella priva di ottimizzazione.

Domanda F

Nel contesto della gestione della memoria virtuale si spieghino chiaramente i concetti di TLB miss e di page fault.

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