



Politecnico di Milano – Sede di Cremona  
Anno Accademico 2020/2021

**Architettura dei Calcolatori e Sistemi Operativi**

**Esame – 02.09.2021**

**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 sviluppino in C il programma principale main() e due funzioni step() ed action() secondo le seguenti specifiche:

1. Le due funzioni devono essere eseguite in due thread separati.
  - La funzione step() svolge le seguenti azioni:
  - Esegue un ciclo in cui incrementa il valore di un contatore CNT ad ogni iterazione
  - Quando il valore corrente del contatore è un multiplo di 10, viene eseguita la funzione action()
  - Il ciclo continua finché il contatore rimane minore di 1000.
2. La funzione action()
  - Esegue un ciclo in cui si sospende in attesa di essere attivata dalla funzione step()
  - Quando attivata somma al contatore CNT un valore casuale compreso tra 0 e 5, quindi si sospende nuovamente.
  - La funzione termina quando il contatore raggiunge il valore 1000.
3. Il programma principale attende la terminazione dei due thread e stampa il numero di iterazioni che sono state eseguite da ognuna delle due funzioni.

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

int    count        = 0;
int    loop_step    = 0;
int    loop_action  = 0;

sem_t  sem_action;

sem_t  sem_step;

void*  step( void* arg )
{
    while( count <= 1000 )
    {
        if( (count % 10) == 0 )
        {
            sem_post( &sem_action );
            sem_wait( &sem_step );
        }
        count      += 1;
        loop_step += 1;
    }
    pthread_exit( NULL );
}
```

```
void* action( void* arg )
{
    while( count <= 1000 )
    {
        sem_wait( &sem_action );
        count      += rand() % 6;
        loop_action += 1;
        sem_post( &sem_step );
    }
    pthread_exit( NULL );
}

int main( int argc, char** argv )
{
    pthread_t t1, t2;

    pthread_create( &t1, NULL, step,  NULL );
    pthread_create( &t2, NULL, action, NULL );

    pthread_join( t1, NULL );
    pthread_join( t2, NULL );

    printf( "step:  %d\n", loop_step  );
    printf( "action %d\n", loop_action );

    exit(0);
}
```

## Domanda B

Si implementi in codice assembly MIPS un programma che prende in ingresso un vettore  $\mathbf{x}$  composto da 1024 elementi interi e produce in uscita un vettore  $\mathbf{y}$  che contiene l'indice  $\mathbf{n}$  degli elementi  $X[\mathbf{n}]$  per cui vale la seguente relazione:

$$X[\mathbf{n}] - X[\mathbf{n}-1] > \mathbf{n}$$

Se il numero di tali elementi è maggiore di 16, il programma ignora la parte successiva dell'array  $\mathbf{x}$  e termina. A tale scopo si considerino le definizioni dei vettori indicate nel seguito.

```
.data
X:   .word 1 10 11 20 21 30 35 50
Y:   .space 64

.text

main:
    la $t0, X           # Points to X
    la $t1, Y           # Points to Y
    li $t2, 0           # N = 0
    li $t3, 16          # K = 16

loop:
    addi $t2, $t2, 1    # N++

    slti $t4, $t2, 1024 # if !(N < 1024) go to end
    beq  $t4, $zero, end

    addi $t0, $t0, 4    # Pointer to X += 4
    lw   $t6, 0($t0)    # X[N]
    lw   $t7, -4($t0)   # X[N-1]

    sub  $t6, $t6, $t7  # t6 = X[N] - X[N-1]
    slt  $t6, $t2, $t6  # if !(N < t6) go to loop
    beq  $t6, $zero, loop

save:
    sw   $t2, 0($t1)    # Y[K] = N
    addi $t1, $t1, 4    # Y += 4
    addi $t3, $t3, -1   # K = K - 1
    bne  $t3, $zero, loop # if K != 0 go to loop

end:
```

## Domanda C

Si consideri architettura a 32 bit dotata di un sistema operativo con memoria paginata e segmentata, sul quale venga compilato il seguente programma:

```
#define N 64

int D1 = 0;
int D2 = 3;
char C1 = '\n';
char C2;
int X1[N];
int* X2;
static const struct {
    int A;
    char B;
    int C;
} S1 = { 10, 20, 30 };

int main()
{
    int V1, V2 = 11;
    X2 = (int*)malloc( 2 * N * sizeof(int) );
    return 0;
}
```

Si supponga inoltre che i segmenti siano così strutturati:

Segmento	Base	Note
TEXT	0x0000	Il codice viene allocato a partire dall'indirizzo di base
DATA	0x4000	
BSS	0x5000	
HEAP	0x6000	Lo heap cresce verso l'alto
STACK	0xFFFF	Lo stack cresce verso il basso

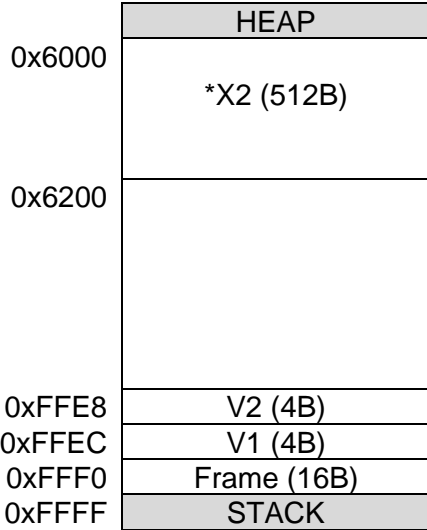
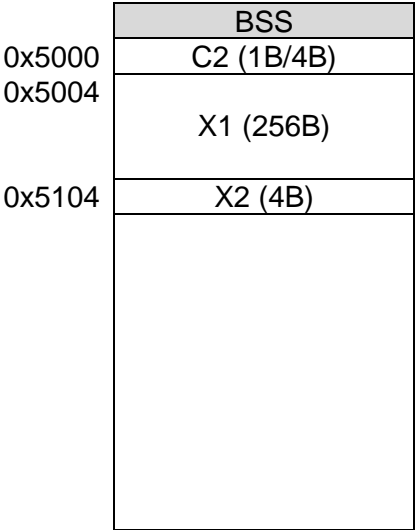
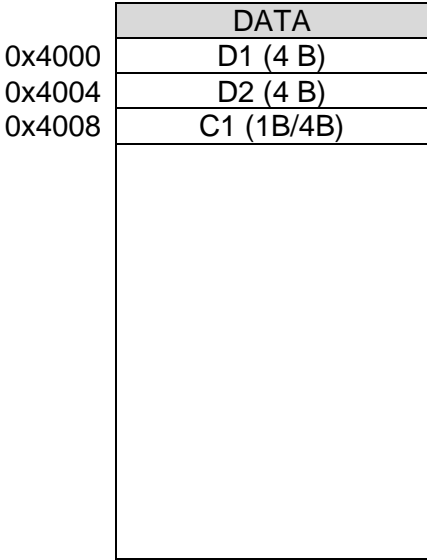
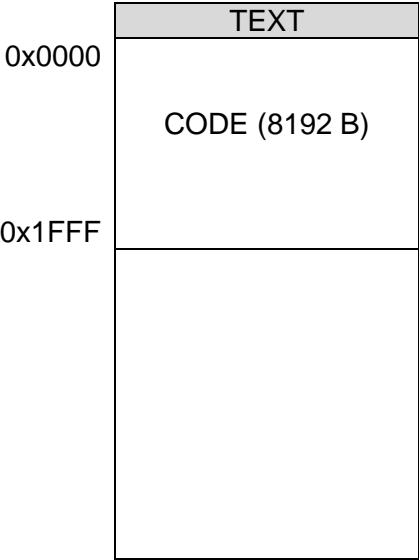
Si riporti nelle figure seguenti lo stato della memoria, indicando in particolare, per ogni zona di memoria:

- Una descrizione del suo contenuto (nome di variabile, valore costante, codice...)
- L'indirizzo di partenza in esadecimale
- La dimensione in bytes

A tal fine si considerino le seguenti ipotesi:

- La memoria è indirizzabile a parole e pertanto allineata a multipli di 4 byte.
- Il codice viene allocato a partire dalla base del segmento di pertinenza.
- Si supponga per semplicità che il punto d'ingresso del programma sia la funzione `main()` del codice riportato, ignorando pertanto la parte di inizializzazione eseguita prima di `main()`.
- Si supponga che il frame di attivazione di una funzione (SP, BP, ...) occupi 16 byte e sia allocato sullo stack prima di ogni altro dato.
- La dimensione del codice binario è pari 8K

Si completi la figura seguente indicando lo stato della memoria immediatamente dopo il caricamento del programma compilato, prima di iniziarne l'esecuzione. Nella figura sono riportate, a titolo di esempio, le descrizioni di parte delle informazioni richieste dal problema.



## Domanda D

---

Si consideri una gerarchia di memoria composta da:

- Memoria centrale: 512MB, indirizzabile a parole di 32 bit
- Cache istruzioni: 64K, set associativa a 8 vie con linee di 64 bytes per ogni via
- Cache dati: 16K, completamente associativa, linee da 256B

Tali memorie presentano i seguenti tempi di accesso:

- Accesso alle cache: 1 ciclo di clock
- Accesso alla memoria centrale in modalità normale: 100 cicli di clock
- Accesso alla memoria centrale in modalità burst: 200 cicli di clock per la prima parola, 25 cicli di clock per le parole a indirizzi successive (modalità burst).

Si consideri inoltre la situazione in cui:

- Hit rate della memoria istruzioni pari al 98%
- Hit rate della memoria dati pari al 99%

Sulla base di queste informazioni:

1. Indicare la struttura degli indirizzi di memoria per le memorie cache

Memoria 512 MB -> 29 bit

ICACHE

#index:  $\log_2(64 \text{ KB} / (64\text{B} \times 8)) = \log_2(128) \rightarrow 7 \text{ bit}$

#offset:  $\log_2(64) \rightarrow 6 \text{ bit}$

#tag:  $29 - 7 - 6 \rightarrow 16 \text{ bit}$

DCACHE

#index:  $\log_2(256) \rightarrow 8 \text{ bit}$

#tag:  $29 - 8 \rightarrow 21 \text{ bit}$

2. Calcolare il tempo medio di accesso alle due cache

ICACHE

$T_{ave,I} = 0.98 \times 1cc + 0.02 [ 1cc + 200cc + 25cc \times (64 / 4 - 1) ] = 12.5cc$

DCACHE

$T_{ave,D} = 0.99 \times 1cc + 0.01 [ 1cc + 200cc + 25cc \times (256 / 4 - 1) ] = 18.75cc$

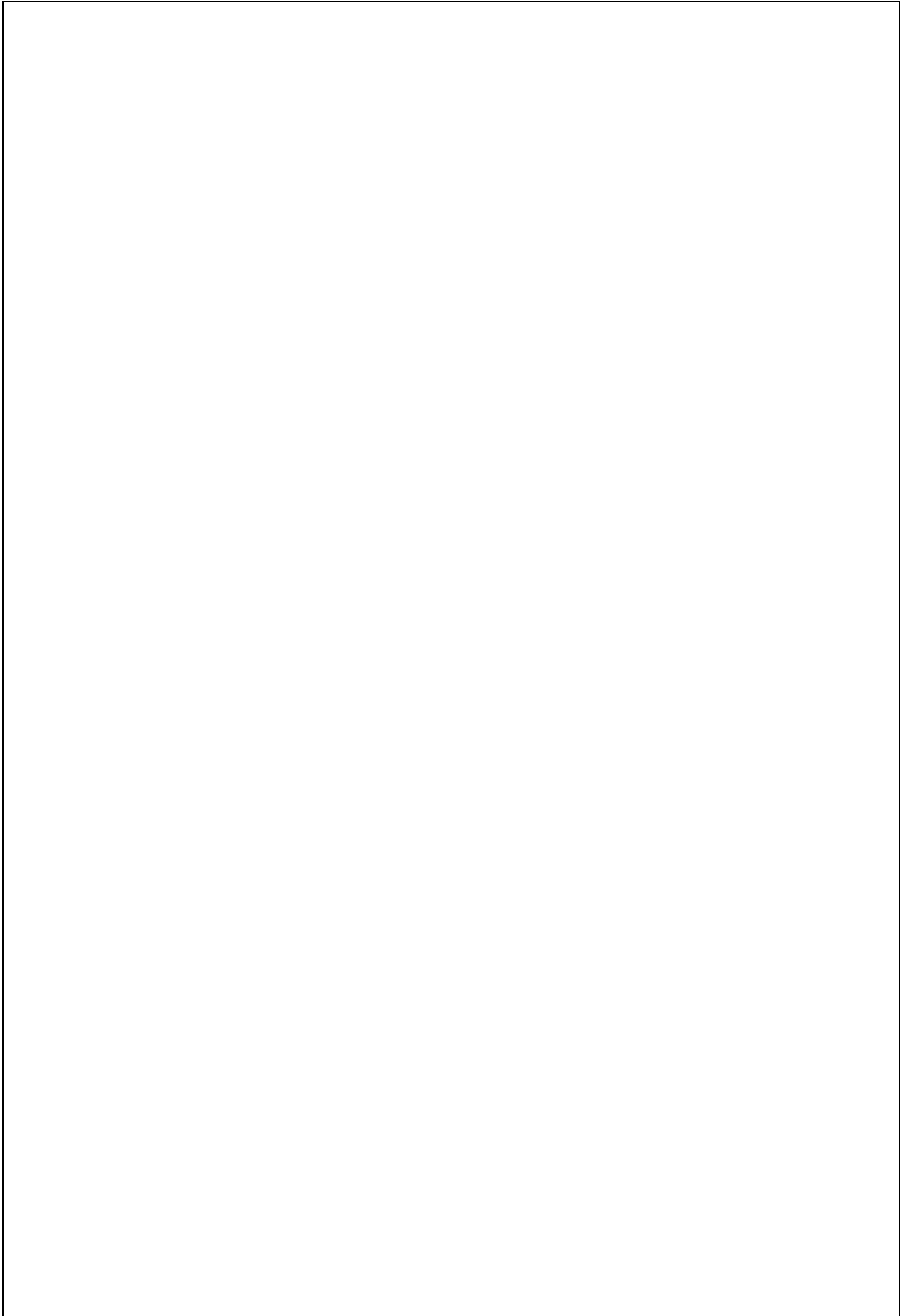




## Domanda F

---

Si descriva il funzionamento di un protocollo di bus basato su handshake sincrono e si riporti il diagramma relativo ad una operazione di lettura, opportunamente commentato.



Instruction	Description	Operation	Type	Opcode	Funct
add rd,rs,rt	Add	rd = rs + rt	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 Greater Than or Equal	if (rs >= rt) PC += 4 + (imm << 2)			
bgt rx,ry,imm	Branch if Greater 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

