Matricola: _____

Nome: _____

Cognome: _____

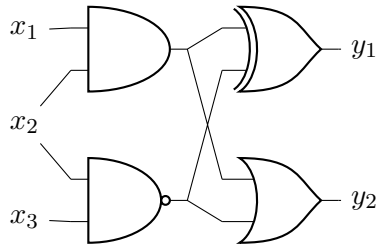
ESAME - Programmazione e Architetture (Modulo B)

20 Maggio 2021 (Simulazione)

L'esame consiste di 15 domande a risposta multipla sugli argomenti del corso. Affinché una risposta sia considerata valida la scelta *deve essere motivata*.

Domanda 1

Si consideri il seguente circuito:



Supponendo che gli input siano $x_1 = 0$, $x_2 = 1$ e $x_3 = 0$, i valori degli output y_1 e y_2 saranno:

☐ $y_1 = 0$, $y_2 = 0$

☐ $y_1 = 0$, $y_2 = 1$

☐ $y_1 = 1$, $y_2 = 0$

☐ $y_1 = 1$, $y_2 = 1$

Domanda 2

Si consideri la seguente sequenza di bit `10010011`. Interpretando questa sequenza come numero in complemento a due otteniamo come valore:

☐ 19

☐ -109

☐ 75

☐ Non è interpretabile come un numero in complemento a due

Domanda 3

Dato un insieme di n registri, per poter selezionare uno specifico registro del quale leggere l'output possiamo usare:

☐ una ALU

☐ un demultiplexer

☐ un latch SR

☐ un multiplexer

Domanda 4

Ricordando che ARM-v7a ha 13 registri di 32 bit `R0`, ..., `R12`, e supponendo che il registro `R0` contenga il valore 12, quale è il valore contenuto nel registro `R2` al termine dell'esecuzione del seguente programma?

```
      MOV R1, R0
      B end
      ADD R1, R1, R0
end   ADD R2, R1, R0
```

☐ 48

☐ 24

☐ Il programma non termina

☐ Non è ben definito dato che il valore di `R1` non è noto

Domanda 5

Se possiamo effettuare il *memory-mapped I/O*, quale delle seguenti affermazioni è necessariamente vera?

☐ La CPU deve fornire delle istruzione apposite per l'I/O

☐ È possibile usare solamente il polling

☐ Possiamo accedere ai dispositivi con le stesse istruzioni usate per l'accesso alla memoria

☐ È presente il DMA (Direct Memory Access)

Domanda 6

Si considerino due processi P_1 e P_2 in un sistema con scheduling round-robin e con *quanto* di tempo pari a 100ms. Supponendo che P_1 abbia un tempo di esecuzione di 300ms e P_2 di 600ms, che P_1 sia schedulato prima di P_2 e ignorando i tempi necessari al context-switching, dopo quanto tempo termineranno i processi P_1 e P_2?

☐ P_1: 500ms, P_2: 900ms

☐ P_1: 300ms, P_2: 600ms

☐ P_1: 800ms, P_2: 900ms

☐ P_1: 900ms, P_2: 900ms

Domanda 7

Nel filesystem standard unix (quello visto a lezione), supponendo di avere l'inode di un file già disponibile in memoria, per accedere all'i-esimo blocco del file quanti accessi al disco dobbiamo fare?

☐ Al massimo 1

☐ Da 1 a 4

☐ Al massimo i accessi

☐ L'inode non ci permette di accedere ai blocchi di cui è composto un file

Domanda 8

Si considerino due processi P_1 e P_2 in esecuzione su un sistema che implementa la memoria virtuale. Sapendo che alla locazione di memoria `0x00FF4A6C` il processo P_1 ha il valore 51, cosa possiamo dire del valore contenuto in P_2 nella locazione di memoria `0x00FF4A6C`?

☐ Un valore diverso da 51

☐ 51

☐ Non abbiamo abbastanza informazioni per dirlo

☐ P_2 non può accedere all'indirizzo `0x00FF4A6C`

Domanda 9

Si consideri il seguente comando per la shell Unix:

```
cat foo.txt | wc -l > bar.txt
```

Supponendo che il file `foo.txt` esista e non vi siano problemi di permessi, il file `bar.txt` al termine dell'esecuzione del comando conterrà:

☐ Lo stesso contenuto di `foo.txt`

☐ Nulla, il file sarà vuoto

☐ `cat foo.txt | wc -l`

☐ Il numero di righe del file `foo.txt`

Domanda 10

Si consideri il seguente codice:

```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char * argv[])
{
  fork();
  fork();
  printf("a");
  return 0;
}
```

Quale delle seguenti affermazioni è vera?

☐ Stampa `aa`

☐ Stampa `aaa`

☐ Stampa `aaaa`

☐ `printf` non può essere usata in un processo che esegue `fork`

Domanda 11

Si supponga di voler far comunicare due processi P_1 e P_2 tramite una pipe. Se il processo P_1 effettua una istruzione di `read` sulla pipe ma P_2 non scrive dall'altra estremità allora:

☐ P_1 rimarrà bloccato in attesa di avere qualcosa da leggere dalla pipe

☐ P_1 leggerà una sequenza di zeri

☐ P_1 non può effettuare una operazione di `read` se P_2 non effettua una `write`, quindi il programma non compila

☐ L'operazione di `read` in P_1 non verrà mai effettuata perché P_2 non ha usato `kill` per inviare un segnale a P_1

Domanda 12

Si consideri la seguente sequenza di operazioni di un thread t_1 per scrivere nella variabile condivisa `x`:

- Acquisisce il lock L_1

- Acquisisce il lock L_2
- Scrive `x`
- Rilascia il lock L_2
- Rilascia il lock L_1

Il thread t_2 effettua invece le seguenti operazioni:

- Acquisisce il lock L_2
- Acquisisce il lock L_1
- Scrive `x`
- Rilascia il lock L_1
- Rilascia il lock L_2

Quale delle seguenti affermazioni è vera?

☐ Non si può verificare un deadlock perché la variabile condivisa è protetta da almeno un lock

☐ Si può verificare un deadlock dovuto all'ordine di acquisizione dei lock

☐ La variabile potrebbe essere scritta dai due thread contemporaneamente

☐ I lock non sono necessari perché stiamo usando thread e non processi

---

Domanda 13

Si consideri il seguente frammento di codice:

```
void * f(void * arg)
{
  for (int i = 0; i < 3; i++) {
    printf("%d ");
  }
  return NULL;
}

int main(int argc, char * argv[])
{
  pthread_t p1;
  pthread_t p2;
  pthread_create(&p1, NULL, f, NULL);
  pthread_create(&p2, NULL, f, NULL);
  return 0;
}
```

Quale dei seguenti **non** è un possibile output?

☐ `1 2 3 1 2 3`

☐ `1 1 2 2 3 3`

☐ `1 2 1 2 3 3`

☐ `1 3 1 2 2 3`

Domanda 14

Nel seguente frammento di codice non usiamo un lock per accedere alla variabile `x`:

```
void * g(void * arg)
{
  int x = 4;
  for (int i = 0; i < 10; i++) {
    x = x + 1;
  }
  return NULL;
}
```

Quale delle seguenti affermazioni è vera?

☐ Sarebbe stato necessario proteggere `x` con un lock

☐ Non serve proteggere `x` con un lock perché è una variabile locale allocata sullo stack

☐ Non serve proteggere `x` con un lock perché tutte le variabili locali sono condivise tra i thread

☐ `x` è protetta da un lock acquisito implicitamente quando si chiama `pthread_create`

Domanda 15

Quale dei seguenti indirizzi IP si trova nella stessa sottorete di 10.0.2.4/16?

☐ 10.0.5.6

☐ 10.13.2.7

☐ 13.8.2.4

☐ 1.0.0.9