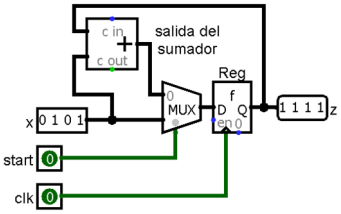
Pregunta 1

línea cache	*etiqueta*	*contenido*
c1	7c1	
4a	64a	
13	d13	

I.- La figura muestra un extracto del estado inicial de un *caché* de 4 KB (2^{12} bytes) de 1 grado de asociativad con 256 líneas de 16 bytes. Un programa accede a las siguientes direcciones de memoria (en hexadecimal): 64**a**0, d**13**8, 7**c1**c, 64**a**4, 9**c1**0, 5**13**8, 9**c1**4, 5**13**c y 7**c1**8. Indique qué accesos a la memoria son aciertos en el caché, cuáles son desaciertos y rehaga la figura mostrando el estado final del caché. Observe que el caché no está vacío inicialmente y por lo tanto el acceso 64**a**0 es un acierto.

II.- La figura muestra un circuito con entradas *start*, *clk*, *x* y salida *z*. La entrada *x* es 5 y se mantiene constante. Un ciclo del reloj inicia con el cambio de 1 a 0 de *clk* y termina con el siguiente cambio de 1 a 0 de *clk*. En el ciclo 1 del reloj *start* se pone en 1, en el ciclo 2 *start* se pone en 0 y luego se mantiene constante en 0. Indique los valores de la salida del sumador y la salida *z* en los ciclos 2, 3 y 4.



Pregunta 2

saludos.bin

11	que tal	buenos dias
0 3	*4 10*	*11*
k	*1era parte*	*2da parte*

Un archivo *particionado* es un **archivo binario** que contiene 2 partes. Un ejemplo es el archivo *saludos.bin* que muestra la figura de la derecha en donde la primera parte es *que tal* y la segunda *buenos dias*. Los primeros 4 bytes del archivo contienen un entero *k* en formato binario *little endian*, que corresponde a la posición en donde comienza la segunda parte. A continuación (desde la posición 4) viene el texto de la primera parte que corresponde exactamente a *k-4* bytes y luego viene el texto de la segunda parte hasta el final del archivo. El archivo podría ser muy grande.

Programe la función *int main(int argc, char *argv[])* para el comando *parte2* que recibe como parámetro el nombre del archivo particionado y despliega en su salida estándar el contenido de la segunda parte del archivo. Ejemplo de invocación y resultado:

```
$ ./parte2 saludos.bin
buenos dias
```

Restricciones: No puede leer el archivo completo. Lea solo lo estrictamente necesario y por lo tanto debe usar *fseek*. Para leer Ud. *debe usar* la función *fread*.

Observaciones: Las partes no son strings porque no terminan con un \0. Si bien el archivo es binario, las partes solo contienen texto en ascii. No necesita diagnosticar errores de uso, como por ejemplo que el archivo no existe. Por simplicidad considere que el computador también es *little endian*. Recuerde que *fread* retorna el número de ítemes leídos, 0 si se llegó al final del archivo.

Pregunta 3

La función *pbb* del cuadro de la derecha recibe como parámetros (i) la función *gen*, que genera por ejemplo 5 cartas al azar para jugar al poker, (ii) *n*, (iii) la función *eval*, que entrega verdadero si por ejemplo las 5 cartas corresponden a 2 pares, y (iv) un puntero a una estructura con parámetros adicionales para *gen* y *eval*. La función *pbb* genera *n* juegos de cartas al azar y entrega una estimación de la probabilidad de obtener por ejemplo 2 pares. Reprograme la función *pbb* de manera que se use *fork* para hacer la estimación en una máquina dual core. En el proceso hijo genere *n/2* juegos y en el padre los otros *n-n/2*.

```
typedef struct cards Cards;
typedef void (*GenFun)(
   void *ptr, Cards *pcards);
typedef int (*EvalFun)(
   void *ptr, Cards *pcards);

double pbb(GenFun gen, int n,
   EvalFun eval, void *ptr) {
  long long sum= 0;
  for (int i= 0; i<n; i++) {
    Cards cards;
    (*gen)(ptr, &cards);
    if ((*eval)(ptr, &cards))
      sum++;
  }
  return (double)sum / n;
}
```

Pregunta 4

Instrucción	*a0*	*a1*	*a3*	*a4*	*arreglo d*
	d	5			3 8 1 7 0
slli a4,a1,2				20	
add a1,a0,a4		*d*+20			
addi a1,a1,-4		*d*+16			
lw a4,0(a0)				3	
lw a3,0(a1)			0		
sw a3,0(a0)					**0** 8 1 7 0
sw a4,0(a1)					0 8 1 7 **3**

```
f:
    slli a4,a1,2
    add  a1,a0,a4
    addi a1,a1,-4

L2:
    lw   a4,0(a0)
    lw   a3,0(a1)
    sw   a3,0(a0)
    sw   a4,0(a1)
    addi a0,a0,4
    addi a1,a1,-4
    bltu a0,a1,L2
    ret
```

A) La función *f* del cuadro de la derecha está programada en assembler Risc-V. Considere que se invoca *f* recibiendo en *a1* el entero 5 y en *a0* la dirección *d* de un arreglo de 5 enteros con los valores 3, 8, 1, 7 y 0. La tabla de la izquierda muestra la ejecución de la función hasta el 2do *sw*. **Prosiga llenando la tabla** con la ejecución de *addi a0,a0,4* hasta que se ejecute la instrucción *ret*. No incluya lo que ya muestra este enunciado.

B) La tabla muestra las instrucciones Risc-V ejecutadas por un programa. Haga un diagrama que muestre el ciclo en que se ejecuta cada etapa de las instrucciones, considerando una arquitectura en pipeline con etapas *fetch*, *decode* y *execute*. El salto en D sí se produce. No hay predicción de saltos.

```
A addi t0,a0,1
B add  a0,s2,a1
C andi a0,a0,7
D blt  a0,a1,X
E
...
X or   a0,a0,s3
Y subi a0,a0,4
```