Examen de Arquitectura de Computadoras

22 de diciembre de 2016

Instrucciones:

- Indique su nombre, apellido y número de cédula en todas las hojas que entregue.
- Escriba las hojas de un solo lado.
- Las hojas deben estar numeradas y en la primer hoja debe escribirse el total.
- Empiece cada ejercicio en una hoja nueva.
- No puede utilizar material ni calculadora.
- **Apague su celular.**
- La duración del examen es de tres horas. En dicho tiempo debe también completar sus datos.
  Solo se contestarán dudas de letra. No se aceptan preguntas en los últimos 30 minutos del examen.
- Para aprobar debe contestar correctamente un ejercicio entero y dos preguntas.

Pregunta 1

Implemente la función paridad par de 3 bits usando solamente un multiplexor.

Pregunta 2

¿Qué es el circuito de refresco y por qué se debe utilizar en memorias DRAM (RAM dinámicas)?

Pregunta 3

Considere una CPU de 32 bits con memoria totalmente direccionable de a byte y una cache de 16KB con lineas de 32 bytes de largo con una función de correspondencia directa. Considere que la cache tiene almacenada la dirección 0x4C3D2E1F (y su bloque asociado).

Indique 2 nuevas direcciones de forma tal que la primera resulte en un cache hit y la segunda resulte en un remplazo del bloque que contiene a la dirección 0x4C3D2E1F en la cache.

Pregunta 4

Utilizando el algoritmo de suma de punto flotante, indique la representación en punto flotante Media Precisión del resultado de sumar 10,25 y el siguiente número cuyo código binario en la misma representación es 1101000011010000.

Problema 1

Dada la siguiente función recursiva para la búsqueda dentro de strings:

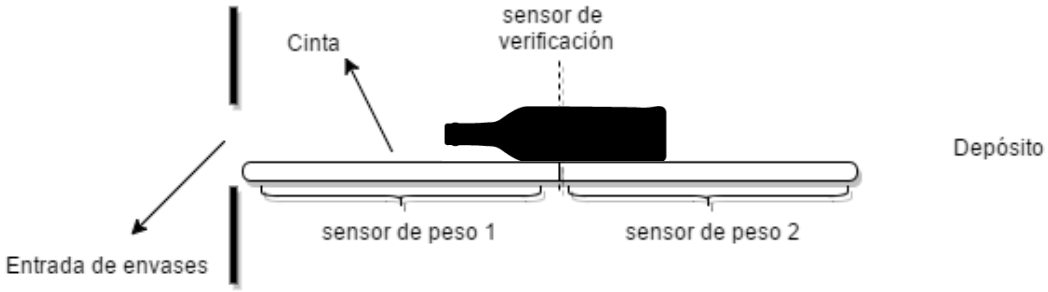
```
int existeSubstring(char* original, char* buscar){
  if (*original == \0)
    return 0;
  else{
     char *iterOriginal = original; char *iterBuscar = buscar;
     while (true){
          if (*iterBuscar == \0)
               return 1;
          else if (*iterBuscar == *iterOriginal){
               iterBuscar++;
               iterOriginal++;
          }else
               return existeSubstring(original+1, buscar);
     }
  }
}
```

Se pide

a. Compilar la función en ensamblador 8086. Asuma que ambos string se encuentran en ES. Los parámetros se pasan por stack y el resultado se devuelve en el stack. Los parámetros se deben retirar del stack.
b. Indique en función del largo del string original el consumo máximo de stack de su implementación.

Problema 1

La empresa ***ENVARQUI*** desea implementar su propio sistema de recepción de envases para supermercados.



La máquina debe encargarse de recepcionar envases (botellas). Para cada envase debe reconocer si el mismo es del tipo retornable y en caso que no lo sea devolverlo al cliente. Cuando un cliente desea utilizar el sistema debe colocar el envase sobre la cinta transportadora, la cual mueve el envase por la máquina. Allí, un sensor detecta si se trata de un envase retornable o no. En caso afirmativo, el envase se debe transportar mediante la cinta hasta la zona de depósito, y si no, se debe devolver a la zona de entrada para que el cliente lo retire.

El movimiento de la cinta transportadora se controla con las señales (de salida) **cinta** y **sentido**. Si **cinta** vale **1**, la misma se moverá en la dirección que indique **sentido**, con la convención que si **sentido** vale **1** se mueve a la derecha (llevando el envase hacia el depósito) y si vale **0** es hacia la izquierda (hacia la boca de entrada).

La cinta contiene dos sensores de peso que permiten conocer las posición del envase, accesibles en las señales (de entrada) **sensor1** y **sensor2**, las cuales indican con un **1** que hay un envase sobre dicha parte de la cinta y **0** en caso contrario. Estos sensores se encuentran dispuestos como muestra la figura, si se activa la cinta hacia la derecha, la fabricación asegura que primero se activará el primer sensor, luego ambos sensores a la vez (mientras el envase pasa por encima de ambos), y luego el segundo sensor únicamente.

La señal (de entrada) **retornable** indica con un **1** que el envase es retornable (y por tanto debe ser aceptado) y con **0** que el envase no lo es (y por tanto debe ser rechazado y devuelto al cliente). Esta entrada es válida mientras el envase se encuentra únicamente sobre el segundo sensor de peso.

En caso de que el envase sea retornable, la cinta debe continuar moviéndose hacia la derecha hasta que el envase caiga en la zona de depósito. En caso de rechazo, la cinta debe moverse en dirección opuesta hasta que salga el envase.

Se pide:
Implementar, utilizando la metodología del curso, el circuito secuencial que controla el sistema de recepción de envases, con las entradas y salidas descriptas. Se dispone de flip flops tipo D y compuertas básicas.
Asuma que el sistema procesa un único envase a la vez.

Solución

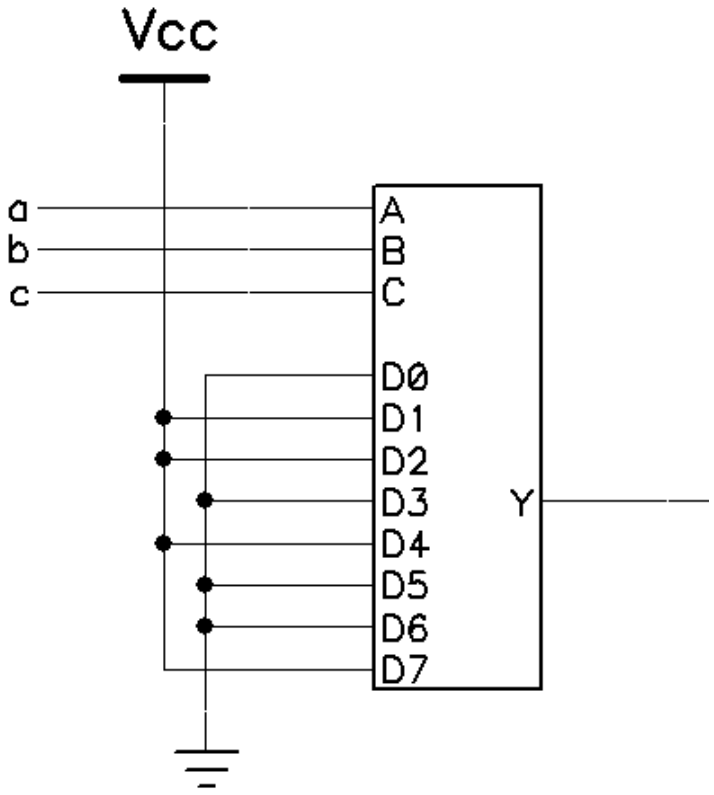
Pregunta 1

La función paridad par es aquella que implementa el exor entre los bits de entrada (de modo de generar un nuevo bit para el sistema de detección de error por paridad que, en el caso de par, hace que el número de unos en el nuevo código es par).

La tabla de verdad es:

a	b	c	pp
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

El circuito con el multiplexor queda:



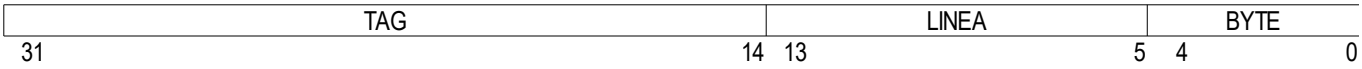
Pregunta 2

Las DRAM (RAM dinámicas) guardan cada bit de información como carga eléctrica almacenada en la capacitancia parásita de la juntura de sus transistores. Por esta razón, por un lado la lectura es destructiva y por tanto el circuito de lectura vuelve a escribir el dato que lee, y por otro la carga eléctrica se pierde con el tiempo por las corrientes de fuga.
Esta último hace necesario que las memorias de este tipo "recarguen" los bits guardados, lo que se logra leyendo periódicamente toda la memoria (por la propiedad del circuito de lectura ya mencionada), tarea que es realizada por el circuito de refresco.

Pregunta 3

Dado que la línea tiene 32 bytes (2^5), se necesitan 5 bits para identificar el byte dentro de la línea. Dado que la cache es de correspondencia directa y tiene 16 Kbytes (2^{14}), tiene 512 líneas (2^{14} / 2^5 = 2^9).

Por tanto la dirección se interpreta así:



Esto quiere decir que la línea que contiene a 0x4C3D2E1F también almacena las direcciones entre 0x4C3D2E00 y 0x4C3D2E1F. Cualquier dirección en ese rango dará un hit, por ejemplo: 0x4C3D2E1A.

Para que haya un reemplazo debemos tener una dirección que contenga la misma identificación de línea, pero distinto TAG, por ejemplo la 0x5C3D2E1F (ésta es evidente porque modificamos los 4 bits más significativos de la dirección que, sin duda, corresponden al TAG).

Pregunta 4

El número 10,25 en base 2 se expresa como 1010,01 = 1,01001 x 2^3.
La representación de Media Precisión tiene 1 bit de signo, 5 bits de exponente y 10 de mantisa.
El desplazamiento del exponente es d = 2^{5-1} – 1 = 15.
Por tanto el código binario del 10,25 en esta representación es:

```
0100100100100000
```

Por su parte el otro código es:

```
1101000011010000
```

Este número es negativo y tiene mayor exponente (5, en lugar de 3), por tanto el resultado va a ser negativo. Para encontrarlo hay que llevar el exponente del 10,25 a 5 (moviendo dos lugares a la derecha la coma binaria) y restar el valor que quede del valor absoluto del número negativo, sin olvidar los 1 implícitos de la representación:

```
  1,0011010000
-
  0,0101001000
 -------------
  0.1110001000
```

Para normalizar hay que correr la coma un lugar a la izquierda (restar 1 al exponente), por lo que el código queda:

```
1100111100010000
```

Problema 1

Parte A

No se pedía preservar los registros, por lo que haremos una solución que no lo hace.

```
; Original en [bp+4], Buscar en [bp+2]
proc existeSubstring
    mov bp, sp
    mov bx, [bp+4]
    cmp byte ptr es:[bx], 0
    jne recorrer
    mov ax, 0
    jmp fin
  recorrer:
    mov si, bx
    mov di, [bp+2]
  while:
    cmp byte ptr es:[di], 0
    jne else_if
    mov ax, 1
    jmp fin
  else_if:
    mov dl, es:[si]
    cmp es:[di], dl
    jne else
    inc si
    inc di
    jmp while
  else:
    inc bx
    push bx
    push [bp+2]
    call existeSubstring
    pop ax
    mov bp, sp
    jmp fin
  fin:
    mov [bp+4], ax
    mov ax, [bp]
    mov [bp+2], ax
    add sp, 2
    ret
endproc
```

Parte B

El consumo de stack para un string de largo N no depende de la búsqueda iterativa, sino de la búsqueda recursiva. El peor caso se da entonces cuando no existe el string buscado dentro del original.
Por cada letra del string original tendremos una llamada recursiva a la función, con el paso base ejecutándose para el fin de string (\0)
Tanto el paso base como el paso recursivo ocupan 6 bytes en el stack (2 parámetros, IP).

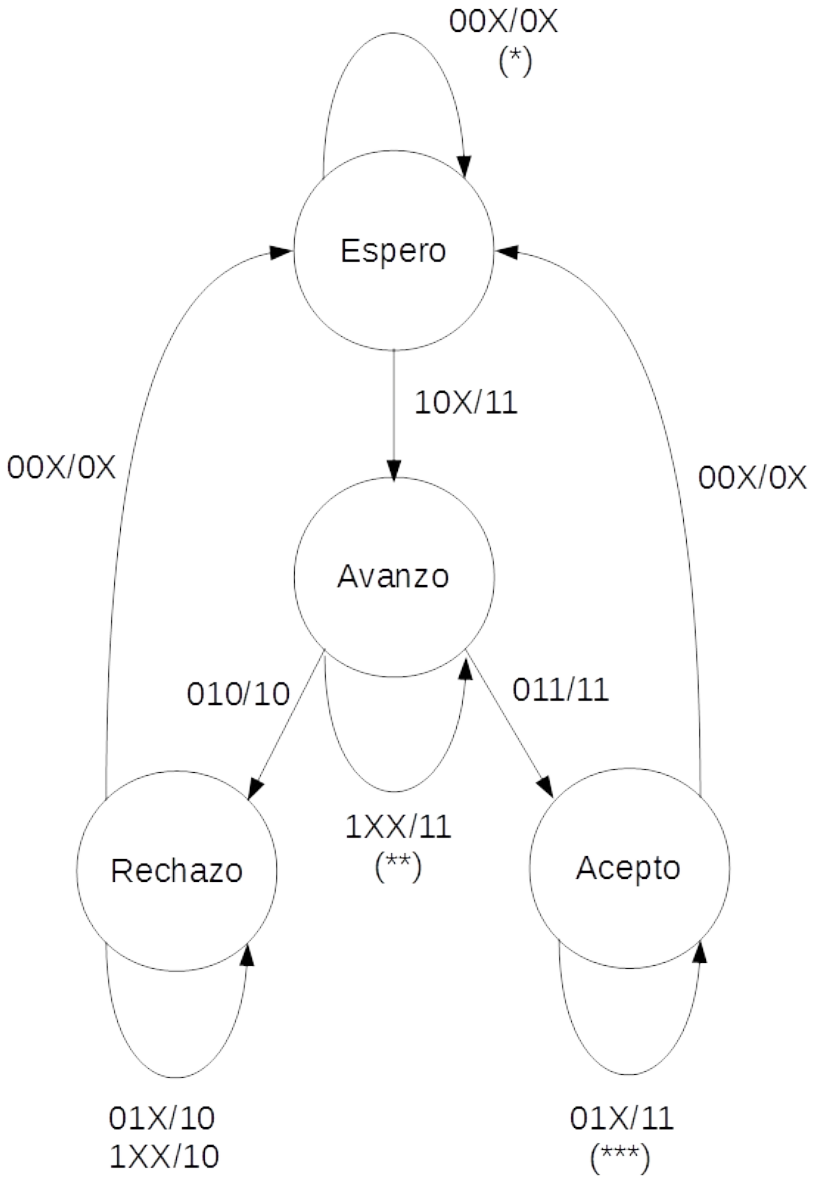
Por lo tanto, en un string de largo N, voy a tener N+1 llamadas con un consumo de stack total de 6 * (N+1) bytes.

Problema 2

Lectura: sensor1 (s1) sensor2 (s2) retornable (r)
Escritura: cinta (c) sentido (d)

Notación: s1 s2 r / c d

La máquina de estados queda:



(*) = por construcción mecánica las combinaciones de entradas X1X son imposible en el estado “Espero” (la botella no puede ser colocada directamente en la parte 2 de la cinta).
(**) = por construcción mecánica las combinaciones de entradas 00X son imposible en el estado “Avanzo” (como llegué a “Avanzo” moviendo a la derecha con el sensor1 en 1, y no muevo a la izquierda en este estado es imposible que los dos sensores estén en 0).
(***) = por construcción mecánica las combinaciones de entradas 1XX son imposible en el estado “Acepto” (ya que llegué a él avanzando a la derecha y cuando el sensor1 pasó a 0, como sigo a la derecha no puede volver a 1).

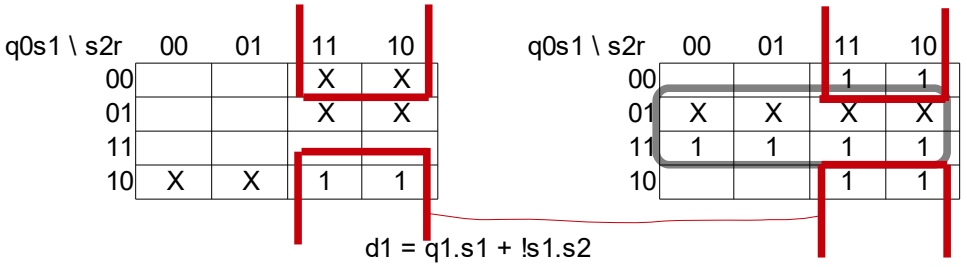
La tabla de estados por lo tanto queda:

Estado Actual	s1	s2	r	Próximo Estado	c	d
Espero	0	0	X	Espero	0	X
Espero	1	0	X	Avanzo	1	1
Espero	X	1	X	X	X	X
Avanzo	0	0	X	X	X	X
Avanzo	0	1	0	Rechazo	1	0
Avanzo	0	1	1	Acepto	1	1
Avanzo	1	X	X	Avanzo	1	1
Acepto	0	0	X	Espero	0	X
Acepto	0	1	X	Acepto	1	1
Acepto	1	X	X	X	X	X
Rechazo	0	0	X	Espero	0	X
Rechazo	0	1	X	Rechazo	1	0
Rechazo	1	X	X	Rechazo	1	0

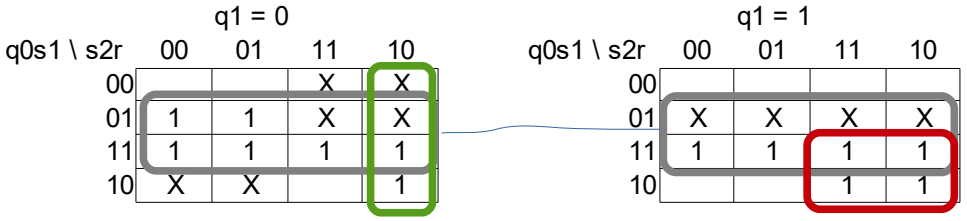
Codificamos los estados como: Espero = 00, Avanzo = 01, Acepto = 10 y Rechazo = 11. Considerando flip-flops tipo D podemos pasar a las tablas de verdad (vamos a omitir la tabla de transiciones y salidas, ya que es bastante directo el pasaje a las tablas de verdad).

q1	q0	s1	s2	r	d1	d0	c	d
0	0	0	0	0	0	0	0	X
0	0	0	0	1	0	0	0	X
0	0	0	1	0	X	X	X	X
0	0	0	1	1	X	X	X	X
0	0	1	0	0	0	1	1	1
0	0	1	0	1	0	1	1	1
0	0	1	1	0	X	X	X	X
0	0	1	1	1	X	X	X	X
0	1	0	0	0	X	X	X	X
0	1	0	0	1	X	X	X	X
0	1	0	1	0	1	1	1	0
0	1	0	1	1	1	0	1	1
0	1	1	0	0	0	1	1	1
0	1	1	0	1	0	1	1	1
0	1	1	1	0	0	1	1	1
0	1	1	1	1	0	1	1	1
1	0	0	0	0	0	0	0	X
1	0	0	0	1	0	0	0	X
1	0	0	1	0	1	0	1	1
1	0	0	1	1	1	0	1	1
1	0	1	0	0	X	X	X	X
1	0	1	0	1	X	X	X	X
1	0	1	1	0	X	X	X	X
1	0	1	1	1	X	X	X	X
1	1	0	0	0	0	0	0	X
1	1	0	0	1	0	0	0	X
1	1	0	1	0	1	1	1	0
1	1	0	1	1	1	1	1	0
1	1	1	0	0	1	1	1	0
1	1	1	0	1	1	1	1	0
1	1	1	1	0	1	1	1	0
1	1	1	1	1	1	1	1	0

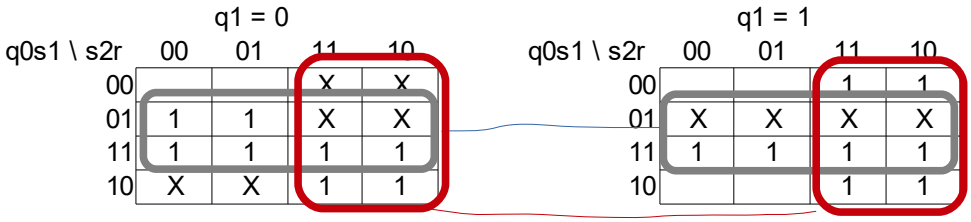
Minimizamos utilizando diagramas de Karnaugh:



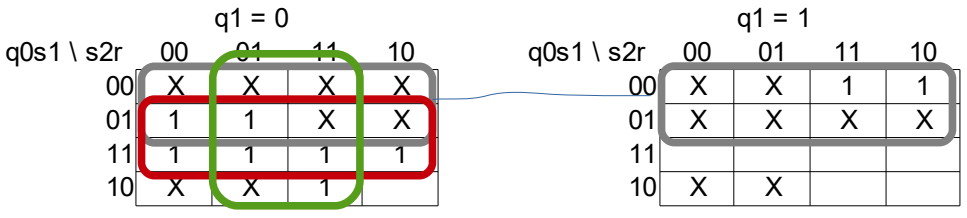
d1 = q1.s1 + !s1.s2



d0 = s1 + q1.q0.s2 + !q1.s2.!r

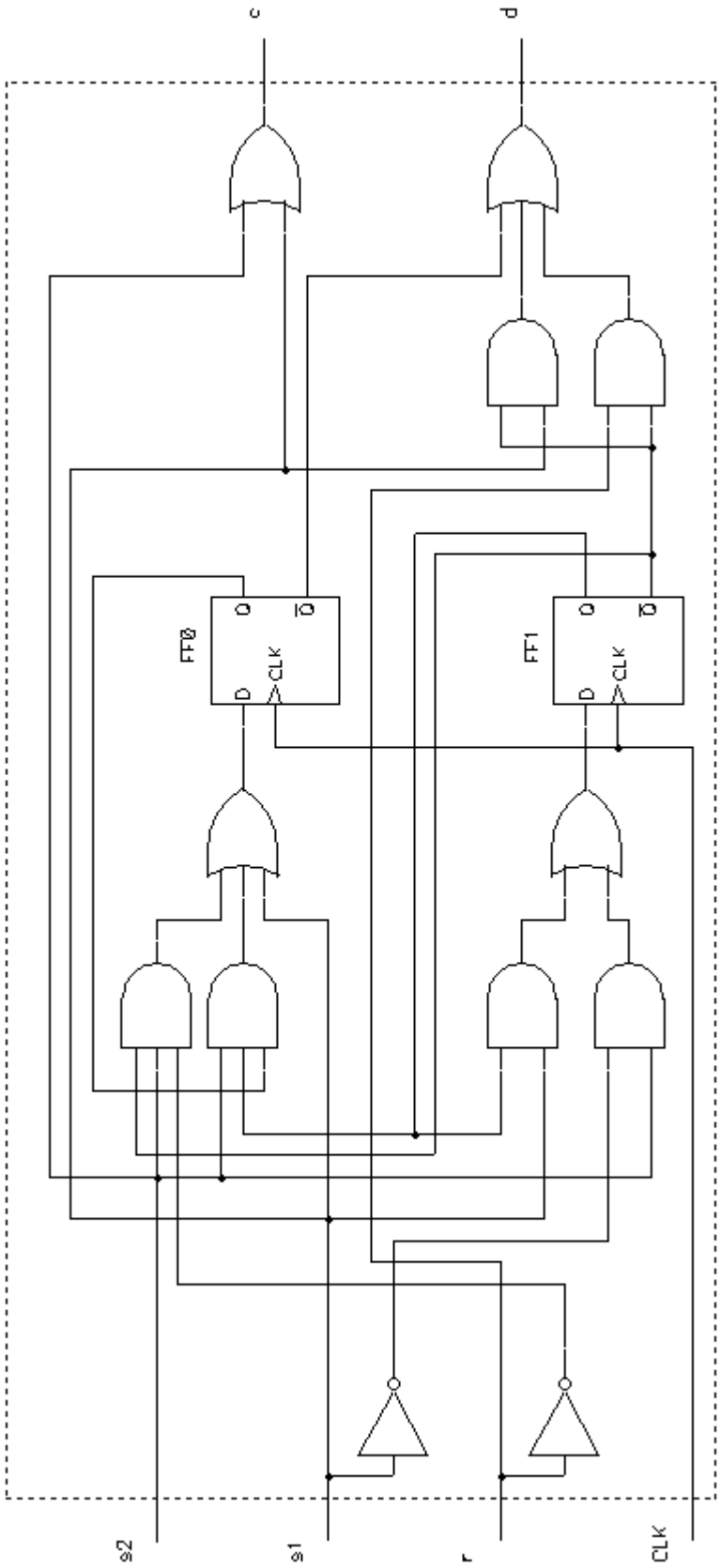


c = s1 + s2



d = !q0 + !q1s1 + !q1r

Por lo tanto el circuito es el mostrado en la siguiente página:


c
d
FF0
FF1
D
CLK
Q
Q
s2
s1
r
CLK