

Examen de Arquitectura de Computadoras

22 de febrero de 2013

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

Indique un procedimiento para determinar si un número Punto Flotante de precisión simple pertenece a las categorías : Normalizado, Desnormalizado o Cero. Además, indique cual es la diferencia entre un número normalizado y uno desnormalizado. Para este último indique cual es su propósito.

Pregunta 2

Explique qué es una memoria caché, el principio de localidad espacial y el principio de localidad temporal. Explique cómo hacen uso las memorias caché del principio de localidad espacial para mejorar el desempeño de la computadora.

Pregunta 3

Considere el pipeline del procesador MIPS visto en clase (etapas IF, ID, EX, MEM, WB):

ADD	\$1, \$3, \$8	; Suma registros \$3 y \$8 y guarda el resultado en \$1
LB	\$2, 0(\$1)	; Carga el byte contenido en la dirección \$1 y lo guarda en \$2
OR	\$1, \$3, \$8	; Realiza el OR binario entre \$3 y \$8 y lo guarda en \$1
ADDU	\$2, \$1, \$6	; Suma registros \$1 y \$6 y guarda el resultado en \$2
AND	\$5, \$6, \$7	; Realiza el AND binario entre \$6 y \$7 y lo guarda en \$5

Realice un diagrama de tiempos mostrando para cada período de reloj, en qué etapa del pipeline se encuentra cada instrucción, mostrando claramente los hazards que se produzcan e indicando su tipo.

Pregunta 4

Dado el siguiente código en assembler 8086:

```
MOV BX, 6
CALL suma_gauss

suma_gauss PROC
    CMP BX, 0
    JZ fin
    XOR AX, AX
    PUSH BX
    DEC BX
    CALL suma_gauss
    POP BX
    ADD AX, BX
    ;;;PRINT ESTADO
fin: RET
ENDPROC
```

Mostrar el valor de los registros SP, AX, BX cuando la ejecución está sobre la marca ;;;PRINT ESTADO, sabiendo que el valor de SP al inicio de la ejecución es 0x1000 y que los llamados son NEAR.

Problema 1

Los módulos de ultrasonido (MUS) utilizan un principio similar a los radares o sonares, los cuales evalúan los atributos de un objetivo mediante la interpretación del eco recibido al enviar una onda de radio o sonido respectivamente.

Los MUS constan de dos elementos bien diferenciados, un emisor de ondas de sonido de alta frecuencia y un sensor que detecta el eco producido. Considerando el tiempo transcurrido desde el envío de la onda y la recepción del eco, es posible calcular la distancia al objeto con el que colisionó la onda. Muchos robots móviles utilizan este tipo de módulos para implementar algoritmos de evasión de obstáculos.

La distancia a los objetos puede calcularse con la ecuación $d = C \times t/2$, donde C es la velocidad del sonido en el aire (344m/s a 20°C) y t es el tiempo transcurrido entre el envío de la onda y su recepción.

La interfaz entre el MUS y el procesador es la siguiente:

- Para generar una onda se debe colocar un 1 en el bit más significativo del byte de E/S solo escritura PING.
- El sensor coloca un 1 en el bit más significativo byte de E/S solo lectura ECHO cuando recibe un eco en el sensor.

Se dispone de reloj que genera una interrupción al procesador atendida por la rutina timer(), la frecuencia del reloj es TIMER_FREQ Hz.

Se pide:

- a) Implementar en un lenguaje de alto nivel la rutina de atención a la interrupción `timer()` y todas las rutinas necesarias para obtener la distancia en **mm** a objetos detectados por un MUS.
- b) Teniendo en cuenta que la instrucción de división del procesador implementa la división entera y no se desea perder precisión en la lectura de distancia al realizar operaciones aritméticas, defina un valor adecuado para TIMER_FREQ. Compilar a assembler 8086 la solución propuesta en la parte anterior y la frecuencia definida. Para el pasaje de parámetros se utiliza el stack.

Observaciones:

- El MUS no es capaz de detectar objetos a más de 8m.
- Cuando no detecta objetos se debe retornar distancia -1.

Problema 2

La empresa de transporte **“Nunca Tardamos Mucho” (NTM)** se encuentra desarrollando su nuevo sistema para cobrar boletos utilizando tarjetas basadas en radiofrecuencia.

El sistema esta compuesto por los siguientes elementos:

- Un módulo que detecta cuando una tarjeta se encuentra a la distancia de funcionamiento con el bit de entrada `tarjeta_presente` (1 = tarjeta presente, 0 = tarjeta no presente)
- Un módulo de radio para decrementar un boleto de la tarjeta. Para decrementar un boleto se pone el bit de salida `decrementar_boleto` en 1. El módulo notifica mediante el bit de entrada `fin_decrementar_boleto` en 1 cuando se decrementó el boleto. Cambia a 0 una vez leído.
- Un módulo de visualización que consta de dos luces, una roja y otra verde, controlado por el bit de salida `luces` (1 : luz verde encendido y roja apagada, 0 : viceversa).
- Un módulo que notifica el ID de la tarjeta que se ha acercado al campo de la antena mediante cuanto bits de entrada `i3`, `i2`, `i1`, `i0` (`i0` es el menos significativo). Cuando se inicia el módulo este devuelve 0 en todos los bits.
- Un módulo de memoria de 4 bits, con señal de habilitación para escritura, controlado por el bit de salida `we_memoria`.

El sistema debe funcionar de la siguiente manera:

Si ningún usuario acerca su tarjeta el sistema indica con luz verde que está disponible. Cuando el usuario coloca la tarjeta a la distancia correcta de funcionamiento se debe comenzar a descontar el boleto indicando al usuario que el proceso esta en curso encendiendo la luz roja. Cuando se finaliza el proceso de débito y si el usuario no retiro en ningún momento la tarjeta se enciende la luz verde y se registra el ID de la tarjeta. Si el usuario retira la tarjeta antes de completar la descarga, la maquina debe

permanecer con la luz roja encendida hasta que una tarjeta sea aproximada a la maquina donde comienza nuevamente el proceso.

Para evitar que el usuario accidentalmente acerque la tarjeta a la distancia correcta de funcionamiento habiendo ya pagado su boleto, el sistema debe controlar antes de debitar que el ID no sea igual al de la tarjeta que se cobró por última vez.

Asuma que la tarjeta siempre tiene saldo para ser descontado y no hay más de una tarjeta al mismo tiempo en el radio de funcionamiento.

Se pide:

Construir el circuito lógico que controla los bits de salida utilizando la metodología vista en el curso. Se dispone de compuertas lógicas y flip-flops tipo D con reset asincrónico.

Observaciones:

Nota: Se sugiere comenzar por construir un comparador de 4 bits

Solución

Pregunta 1

Para saber cual es la categoría de un número de precisión simple (1 bit de signo, 8 bits de exponente y 23 bits de mantisa), podemos seguir el siguiente procedimiento:

1. Si la representación del exponente no es todos los bits en 0 ($e=0\dots0$) o todos los bits en 1 ($e=1\dots1$) sabemos que el número es NORMALIZADO.
2. Si la representación del exponente es todos los bits en 0:
 1. Si la mantisa son todos los bits en 0 ($m=0\dots0$) el número es CERO.
 2. Sino, el número es DESNORMALIZADO.

Un número en punto flotante desnormalizado es aquel en donde para su representación no se utiliza el 1 implícito como en el caso de los números normalizados, sino que se utiliza un 0 implícito.

Su propósito es representar números más pequeños que aquellos representables con el 1 implícito.

Pregunta 2

Una memoria caché es una memoria rápida y más pequeña que la memoria principal que se ubica entre ésta y el procesador de forma de mejorar el tiempo de acceso promedio del procesador a memoria.

El principio de localidad espacial indica que si se accede a un dato en memoria, existe una probabilidad alta de que el siguiente dato accedido tenga una dirección cercana al primero. El principio de localidad temporal indica que si se accede a un dato en memoria, existe una probabilidad alta de que en poco tiempo se vuelva a acceder al mismo dato.

El principio de localidad espacial se aplica en las memoria cache haciendo que cuando deben acceder a memoria principal lo hagan de a bloques, trayendo el dato solicitado por el procesador (en el caso que no esté) así como otros datos cercanos aprovechando que los siguientes accesos sean probablemente a datos próximos al original.

Pregunta 3

	1	2	3	4	5	6	7	8	9	10
ADD	IF	ID	EX	MEM	WB					
LB		-	-	-	IF	ID	EX	MEM	WB	
OR						IF	ID	EX	MEM	WB
ADDU							-	-	-	IF
AND										

	11	12	13	14	15	16	17	18	19	20
ADD										
LB										
OR										
ADDU	ID	EX	MEM	WB						
AND	IF	ID	EX	MEM	WB					

Los hazards presentes son:

Hazard de datos provocado por la dependencia de datos RaW entre ADD y LB en el registro \$1, el cual provoca se deban insertar 3 operaciones NOP luego del ADD.

Hazard de datos provocado por la dependencia de datos RaW entre OR y ADDU en el registro \$1, el cual provoca se deban insertar 3 operaciones NOP luego del OR.

Pregunta 4

llamada 6:

SP = 0xFEAF

AX = 1

BX = 1

llamada 5:

SP = 0xFEE

AX = 3

BX = 2

llamada 4:

SP = 0xFF2

AX = 6

BX = 3

llamada 3:

SP = 0xFF6

AX = 10

BX = 4

llamada 2:

SP = 0xFFA

AX = 15

BX = 5

llamada 1:

SP = 0xFFE

AX = 21

BX = 6

Problema 1

Parte 1

```
#define TIMER_FREQ      ...
#define C               344
#define MAX_DIST        8
#define TICS_MAX_DIST   2 * MAX_DIST * TIMER_FREQ / C

boolean readingDistance = false;
int tics = 0;

int getDistance() {
    int result;

    out(PING,128);           // genero onda en el MUS
    tics = 0;                // inicializo contador de tiempo transcurrido
    readingDistance = true;   // le "aviso" al timer que cuente
    while (readingDistance); // espero a que la onda regrese o timeOut
    if (tics > TICS_MAX_DIST)
        result = -1;         // no recibí eco
    else
        result = 1000 * C * tics / TIMER_FREQ / 2; // calculo la distancia en mm por fórmula

    return result;
} // fin metodo getDistance

void timer() {
    if (readingDistance) {
        tics ++;
        if ((in(ECHO) & 0x80) || (tics > TICS_MAX_DIST))
            readingDistance = false;
    }
} // fin del manejador de interrupciones
```

Parte 2

Uno de los valores posibles para `TIMER_FREQ` es $500 \cdot C$, de esta manera "tics" expresa la distancia en mm, por lo que ahorramos todas las cuentas con la consiguiente ventaja para la precisión del resultado.

```
C            EQU 344
TIMER_FREQ   EQU 500*C
MAX_DIST     EQU 8
TICS_MAX_DIST EQU 2 * MAX_DIST * TIMER_FREQ / C
FALSE        EQU 0
TRUE         EQU 1

readingDistance db FALSE
tics            dw 0

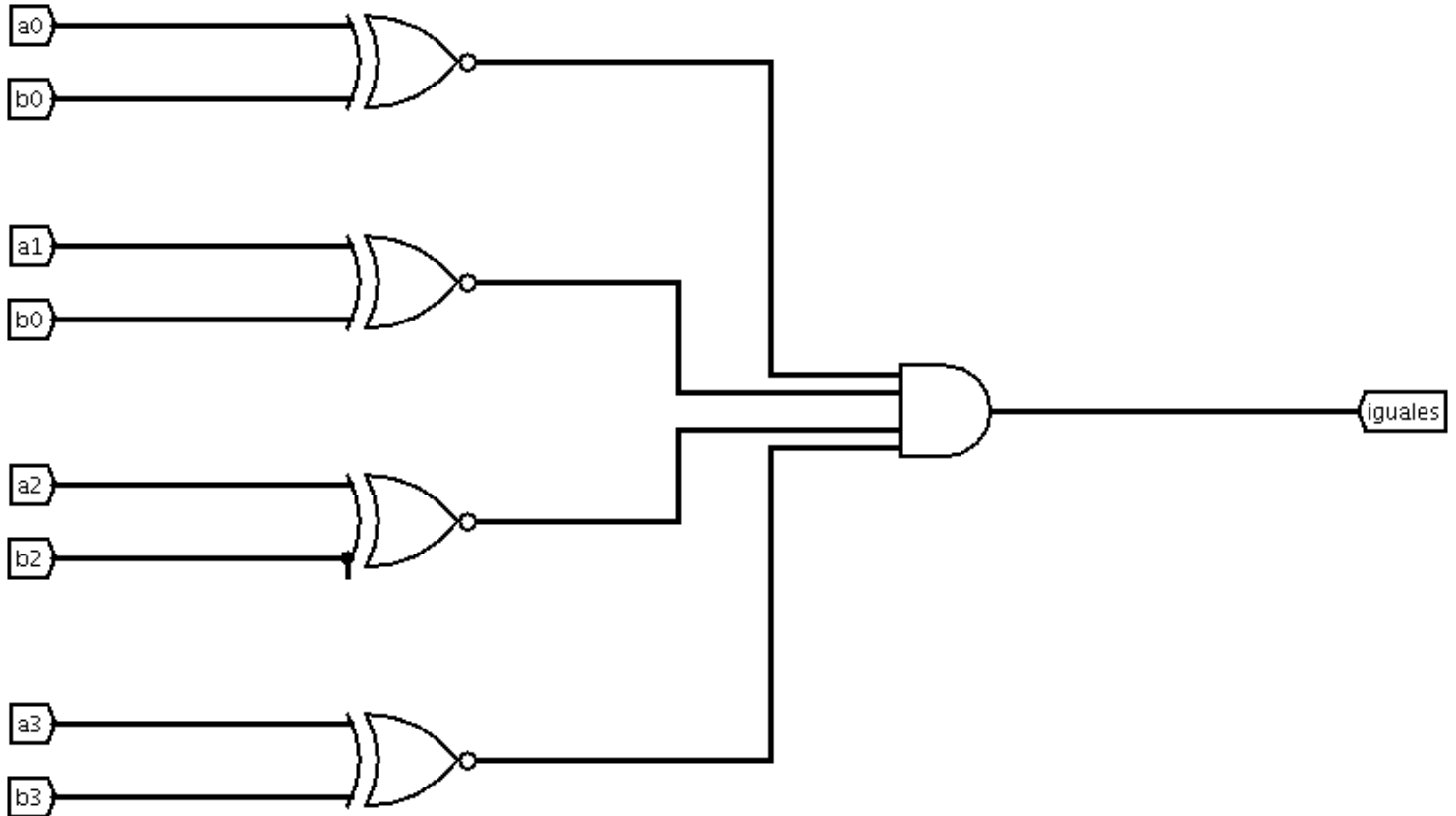
getDistance proc
    mov DX, PING
    mov AL, 80h
    out DX, AL           ; genero onda en el MUS
    mov tics, 0           ; inicializo contador de tiempo transcurrido
    mov readingDistance, TRUE ; le "aviso" al timer que cuente
wait4echo:
    cmp readingDistance, TRUE
    je wait4echo          ; espero a que la onda regrese o timeout
    cmp tics, TICS_MAX_DIST
    jbe else
```

```
        mov AX, -1;          ; no recibí eco
        jmp finIf
else:
    mov AX, tics             ; distancia en mm según fórmula teniendo en cuenta la frecuencia elegida
finIf:
    pop DX
    push AX
    push DX
    ret
getDistance endp

timer proc far
    cmp readingDistance, TRUE
    jne finTimer
    push AX                  ; salvo contexto
    push DX
    mov DX ECHO
    inc tics
    in AL, DX                ; leo de entrada salida el estado del eco
    and AL, 80h              ; aplico máscara más significativo
    jnz setNotReading
    cmp tics, TICS_MAX_DIST
    ja setNotReading
    jmp finIf                ; ninguna de las dos condiciones se cumple
setNotReading:
    mov readingDistance, FALSE
finIf:
    pop DX                  ; recupero contexto
    pop AX
finTimer:
    iret
timer endp
```

Problema 2

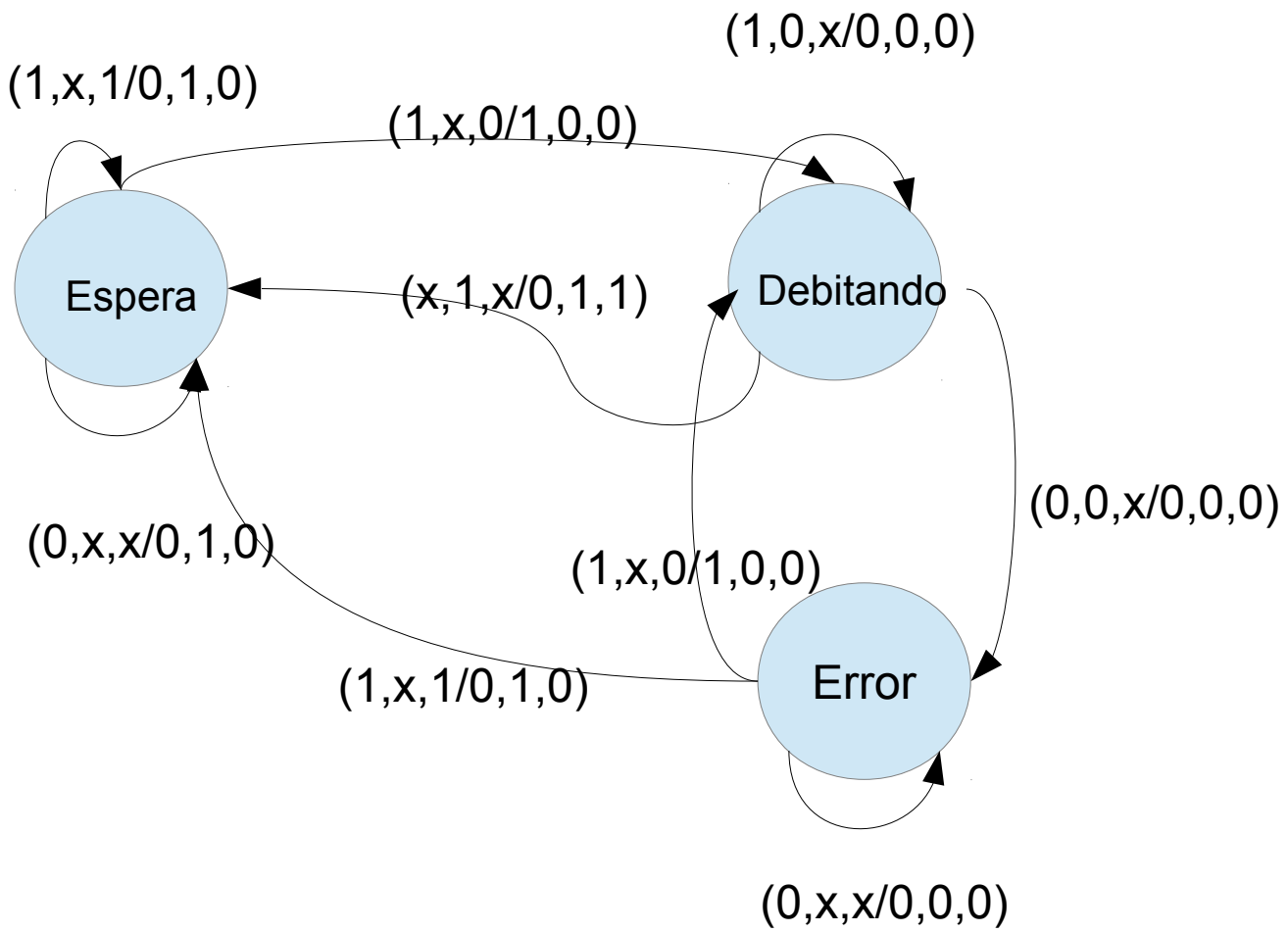
Comenzaremos construyendo un comparador de 4 bits, dado que solo nos interesa la comparación por igual por tanto podemos construirlo fácilmente de la siguiente manera:



La salida “iguales” tomará el valor 1 cuando el número representado por $a_3a_2a_1a_0$ y $b_3b_2b_1b_0$ son iguales.

Con el comparador realizo la comparación del identificador ($i_3 i_2 i_1 i_0$) con el último id almacenado en el módulo de memoria luego de una recarga exitosa. Utilizo la salida del comparador como una entrada para el sistema (señal iguales).

Construyamos una maquina de estados cuyas entradas sean tarjeta_presente, fin_decrementar e iguales y sus salidas decrementar_boleto, luces y we_memoria. Utilizaremos la siguiente codificación: (tarjeta_presente, fin_decrementar, iguales/decrementar_boleto, luces, we_memoria)



En base a este diagrama de estados construimos la tabla de estados:

Estado	Entradas			Prox. Estado	Salidas		
	tarjeta_presente	fin_decrementar	iguales		dec_boleto	luces	wc_mem
espera	0	0	0	espera	0	1	0
	0	0	1	espera	0	1	0
	0	1	0	espera	0	1	0
	0	1	1	espera	0	1	0
	1	0	0	debitando	1	0	0
	1	0	1	espera	0	1	0
	1	1	0	debitando	1	0	0
	1	1	1	espera	0	1	0
	0	0	0	error	0	0	0
debitando	0	0	1	error	0	0	0
	0	1	0	espera	0	1	1
	0	1	1	espera	0	1	1
	1	0	0	debitando	0	0	0
	1	0	1	debitando	0	0	0
	1	1	0	espera	0	1	1
	1	1	1	espera	0	1	1
	0	0	0	error	0	0	0
	0	0	1	error	0	0	0
error	0	1	0	espera	0	1	1
	0	1	1	espera	0	1	1
	1	0	0	debitando	1	0	0
	1	0	1	espera	0	1	0
	1	1	0	debitando	1	0	0
	1	1	1	espera	0	1	0

Codificando los estados como:

espera 00

debitando 01

error 10

Y usando la ecuación de los FF tipo D, pasamos a la tabla de transiciones y salidas:

Estado		Entradas			Prox. Estado		Salidas		
e1	e0	tarjeta_presente	fin_decrementar	iguales	d1	d0	dec_boleto	luces	wc_mem
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0	1	0
0	0	0	1	1	1	0	0	1	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	1	1	0	0	1	0
0	0	1	1	0	0	1	1	0	0
0	0	1	1	1	0	0	1	0	0
0	0	1	1	1	1	0	0	1	0
0	1	0	0	0	0	1	0	0	0
0	1	0	0	0	1	1	0	0	0
0	1	0	1	0	0	0	0	1	1
0	1	0	1	1	1	0	0	1	1
0	1	1	0	0	0	0	1	0	0
0	1	1	0	1	1	0	1	0	0
0	1	1	1	0	0	0	0	1	1
0	1	1	1	1	1	0	0	1	1
1	0	0	0	0	0	1	0	0	0
1	0	0	0	0	1	1	0	0	0
1	0	0	1	0	0	1	0	0	0
1	0	0	1	1	1	1	0	0	0
1	0	1	0	0	0	0	1	1	0
1	0	1	0	1	1	0	0	1	0
1	0	1	1	0	0	0	1	1	0
1	0	1	1	1	1	0	0	1	0
1	1	0	0	0	0	x	x	x	x
1	1	0	0	0	1	x	x	x	x
1	1	0	1	0	0	x	x	x	x
1	1	0	1	1	1	x	x	x	x
1	1	1	0	0	0	x	x	x	x
1	1	1	0	1	1	x	x	x	x
1	1	1	1	0	0	x	x	x	x
1	1	1	1	1	0	x	x	x	x
1	1	1	1	1	1	x	x	x	x

Los diagramas de Karnaugh correspondientes resultan:

e1e0\tp fd	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	x	x	x	x
10	1	1	0	0

I=0

e1e0\tp fd	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	x	x	x	x
10	1	1	0	0

I=1

$$d1 = e0tp'fd' + e1tp$$

d0

e1e0\tp fd	00	01	11	10
00	0	0	1	1
01	0	0	0	1
11	x	x	x	x
10	0	0	1	1

I=0

e1e0\tp fd	00	01	11	10
00	0	0	0	0
01	0	0	0	1
11	x	x	x	x
10	0	0	0	0

I=1

$$d0 = e0'tpi' + e1tpi' + tpfd'i' + e0tpfd'$$

desc_boletos

e1e0\tp fd	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	x	x	x	x
10	0	0	1	1

I=0

e1e0\tp fd	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	x	x	x	x
10	0	0	0	0

I=1

$$desc_boletos = e0'tpi'$$

luces

e1e0\tp fd	00	01	11	10
00	1	1	0	0
01	0	1	1	0
11	x	x	x	x
10	0	0	0	0

I=0

e1e0\tp fd	00	01	11	10
00	1	1	1	1
01	0	1	1	0
11	x	x	x	x
10	0	0	1	1

I=1

$$luces = e1'e0'tp' + e0fd + e1'e0'i + tpfdi + e0'tpi$$

e1e0\tp fd	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	0	0	0
10	0	0	0	0

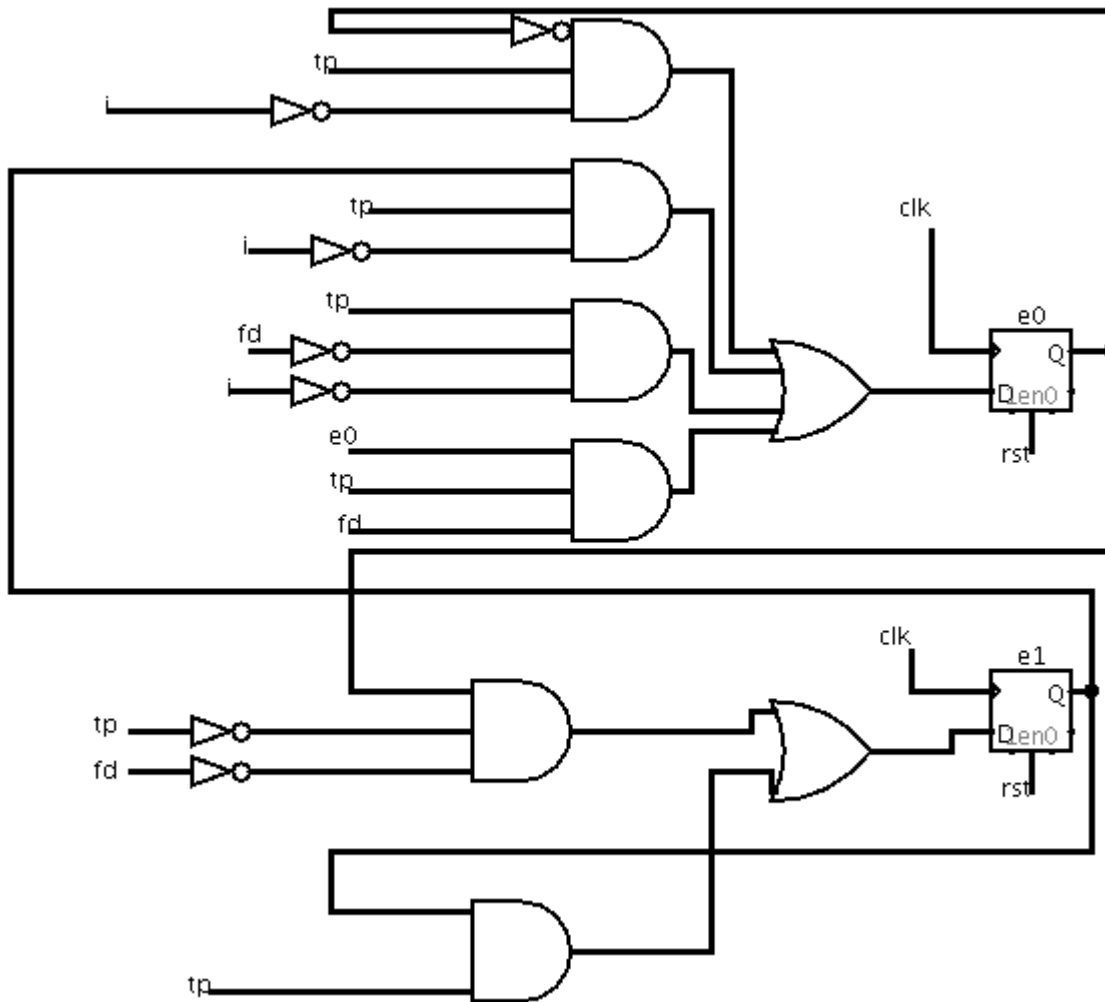
I=0

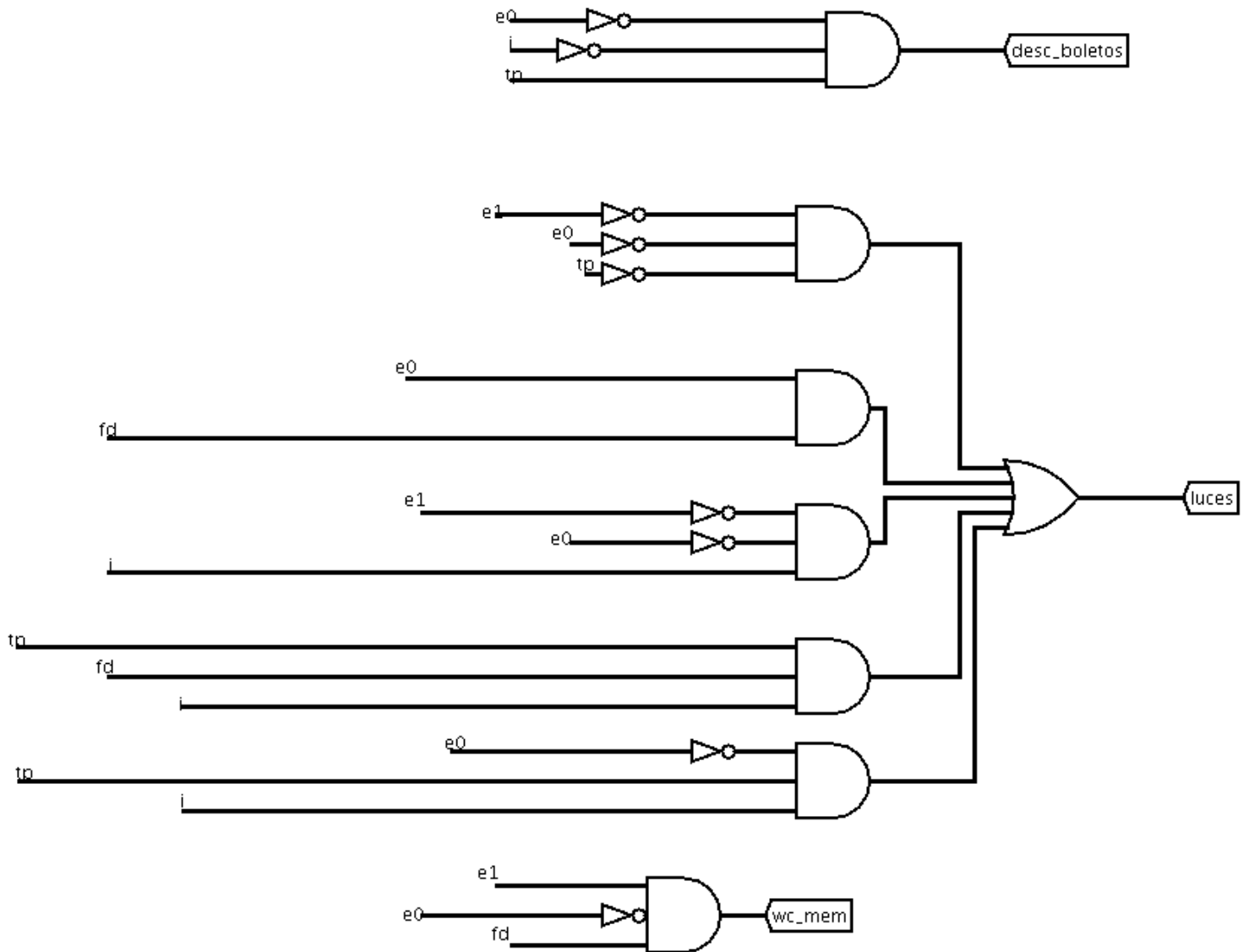
e1e0\tp fd	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	0	0	0
10	0	0	0	0

I=1

$$wc_mem = e1'e0fd$$

El circuito queda entonces:





Una solución alternativa al diseño del problema es la siguiente:

(tarjeta_presente, fin_decrementar, iguales/decrementar_boleto, luces, we_memoria)

(0,x,x/0,0,0)

(0,x,x/0,1,0)

(1,x,0/1,0,0)

