Examen de Arquitectura de Computadoras

19 de diciembre de 2023

Instrucciones:

- **Indique su nombre, apellido y número de cédula en todas las hojas que entregue.**
- **Las hojas deben estar numeradas y en la primer hoja debe escribirse el total.**
- **Apague su celular. No puede utilizar material ni calculadora.**
- **La duración del examen es de tres horas. En dicho tiempo debe también completar sus datos.**
- **Para aprobar debe contestar correctamente un ejercicio entero y dos preguntas.**
- **DEBEN justificarse todas las respuestas.**

Pregunta 1

Considere las siguientes tiras de bits: 1010 1101 0110 1101 y 0110 1111 0000 1100

a) Calcule la suma de ambos números interpretándolos como entero sin signo de 16 bits. Indique si el resultado es representable.
b) Calcule la suma de ambos números interpretándolos como enteros en complemento a 2 de 16 bits. Indique si el resultado es representable.

Respuesta

a)

```
  1010 1101 0110 1101
+ 0110 1111 0000 1100
-----------------------------
 1 0001 1100 0111 1001
```

El resultado consta de 17 bits (hay carry), por lo cual no es representable.

b)

El resultado se mantiene (complemento a 2 preserva la suma), y es correcto ya que la suma de un numero negativo (1010 1101 0110 1101) con un positivo (0110 1111 0000 1100) en complemento a 2 siempre es representable.

Pregunta 2

Teniendo en cuenta el mecanismo de interrupciones para un procesador 8086 con un controlador de interrupciones, describa la secuencia de eventos que ocurren en el proceso de atención a una solicitud de interrupción generado por un controlador de E/S.

Respuesta

Para atender una solicitud de interrupción, la CPU realiza los siguientes pasos:

- El controlador de E/S que desea ser atendido envía la solicitud al controlador de interrupciones, que recibe el pedido a través de una entrada IRQ de este. El controlador de interrupciones entonces genera un pedido a la CPU a través de la señal INT.
- La CPU consulta si hay un pedido de interrupción al final del ciclo de instrucción, luego de la etapa de

"write" y antes del siguiente "fetch".

- Se salva en el stack la dirección de retorno, es decir el valor actual del puntero de instrucción (IP) y el segmento (CS), así como el registro de flags.
- Se identifica al periférico que realizó el pedido de interrupción. Para esto, la CPU coloca en 1 la salida INTA para avisar al controlador que va a atender a la interrupción, y este coloca en el bus de datos el identificador del periférico que realizó el pedido (ID_INT, 8bits).
- Obtiene la dirección de la rutina de servicio a la interrupción correspondiente a través del vector de interrupciones. La dirección buscada se encuentra en formato segmentado en las direcciones ID_INT * 4 (desplazamiento) e ID_INT * 4 + 2 (segmento).
- Enmascara las interrupciones.
- Se realiza un jump far al inicio de la rutina de atención a la interrupción.

Pregunta 3

a) Describa los principios de localidad espacial y temporal.
b) Considere dos memorias cache con las siguientes especificaciones:

- Capacidad 16 KB, correspondencia totalmente asociativa, 2048 líneas.
- Capacidad 4 KB, correspondencia directa, 32 líneas.

¿Cuál de estas memorias tiene mayor hit rate en el siguiente programa? Considere que las variables *sum* e *i* se guardan en registros.

```
#define TAM_MEMORIA ...
short mem[TAM_MEMORIA];
int sum = 0
for (int i = 0; i < 1024; i++){
     sum += mem[i] * 2;
}
```

Respuesta

a)

El principio de localidad establece que los programas acceden a una porción relativamente reducida del espacio de direcciones en un determinado lapso de tiempo.

- Localidad temporal: si un ítem es referenciado en determinado momento, es común que vuelva a ser referenciado poco tiempo después.
- Localidad espacial: cuando un ítem es referenciado en determinado momento, es común que los ítems con direcciones “cercanas” también sean accedidos poco tiempo después.

b)

Cache 1:
Capacidad 16 KB, correspondencia totalmente asociativa, 2048 líneas.
Es decir 2048 lineas de 8B, donde cada dirección de memoria puede guardarse en cualquier línea.
Cache 2:
Capacidad 4 KB, correspondencia directa, 32 líneas.
Es decir 32 lineas de 128B sonde cada dirección de memoria puede guardarse en una única línea.

El acceder a direcciones consecutivas, la asociatividad no presenta una ventaja ya que en la línea siguiente se guardará el siguiente bloque de memoria (igual que en la correspondencia directa). Además, dentro de cada línea

se accederá a todas las posiciones de memoria.

Es decir, en ambas memorias sucederá un comportamiento análogo, se llenara una línea, se accederá a todas sus posiciones de memoria y luego se repetirá el mismo proceso en las líneas consecutivas hasta obtener las 1024 palabras.

De esta forma el hit-rate está definido por la cantidad de palabras que se pueden almacenar en una línea, sabiendo que habrá un miss solo la primera vez, es decir:

$$hit-rate = 1 - \frac{1}{palabras-por-linea}$$

entonces el mayor hit-rate será de la cache con mayor largo de línea independientemente del largo de la palabra. De esta forma la caché más apropiada es la segunda (128B > 8B).

Pregunta 4

Explique qué tipo de circuito es la Unidad de Control de una CPU y qué acciones realiza. Describa los estilos de diseño de Unidad de Control vistos en el curso.

Respuesta

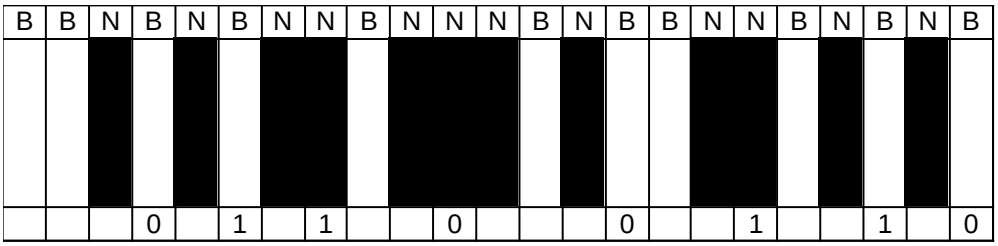
La unidad de control es una máquina secuencial. Su acción es realizar el ciclo de instrucción, un conjunto de acciones ordenado y secuencial para lograr ejecutar cada instrucción, aplicando una operación determinada sobre los operandos correspondientes y almacenando el resultado en un lugar indicado.

Las opciones vistas en el curso para el diseño de la unidad de control son el control cableado o la microprogramación.

En el caso del control cableado, la UC se diseña como un circuito secuencial utilizando, por ejemplo, la metodología de Máquinas de Estado. En el caso de la microprogramación, la UC se construye en base a una máquina secuencial más simple que ejecuta los microprogramas que contienen la secuencia de valores de las señales internas de control de la CPU para lograr la ejecución de las instrucciones.

Problema 1

Se considera un sistema de codificación de bits mediante barras blancas y negras, como las utilizadas en los conocidos códigos de barras. Para simplificar consideraremos que las barras son siempre de un ancho múltiplo de un cierto ancho unitario.
El sistema funciona de la siguiente manera: una secuencia de bits comienza con dos o más barras blancas. Luego una barra negra representa el bit 0, mientras que dos o más barras negras seguidas representan el 1.
El sistema de codificación es "entrelazado" por lo que el siguiente bit se codifica al revés: una barra negra sola representa al 1 y dos o más barras negras seguidas al cero. En general todos los bits impares se codifican como el primero de la secuencia y todos los pares como el segundo.
Los bits de una secuencia están separados por una única barra blanca.
El fin de una secuencia de bits está indicado por dos o más barras blancas, que también se interpretan como el comienzo de una nueva secuencia de bits.



Se desea construir un circuito secuencial que decodifique los bits codificados con este sistema de barras. El circuito recibe como entrada el color de la barra leída (0 = blanca, 1 = negra) y debe generar dos salidas, una que indique si ha decodificado un bit (0 = falso, 1 = verdadero) y otra que sea el valor del bit decodificado.
Nota: el clock del circuito es tal que coincide con la cadencia con la que se leen las barras (en cada flanco del reloj una nueva barra es leída).

Se pide: diseñar y dibujar el circuito secuencial descripto en base a flip flops tipo D y compuertas básicas.
Debe utilizarse la metodología del curso en la resolución de este problema.

Solución

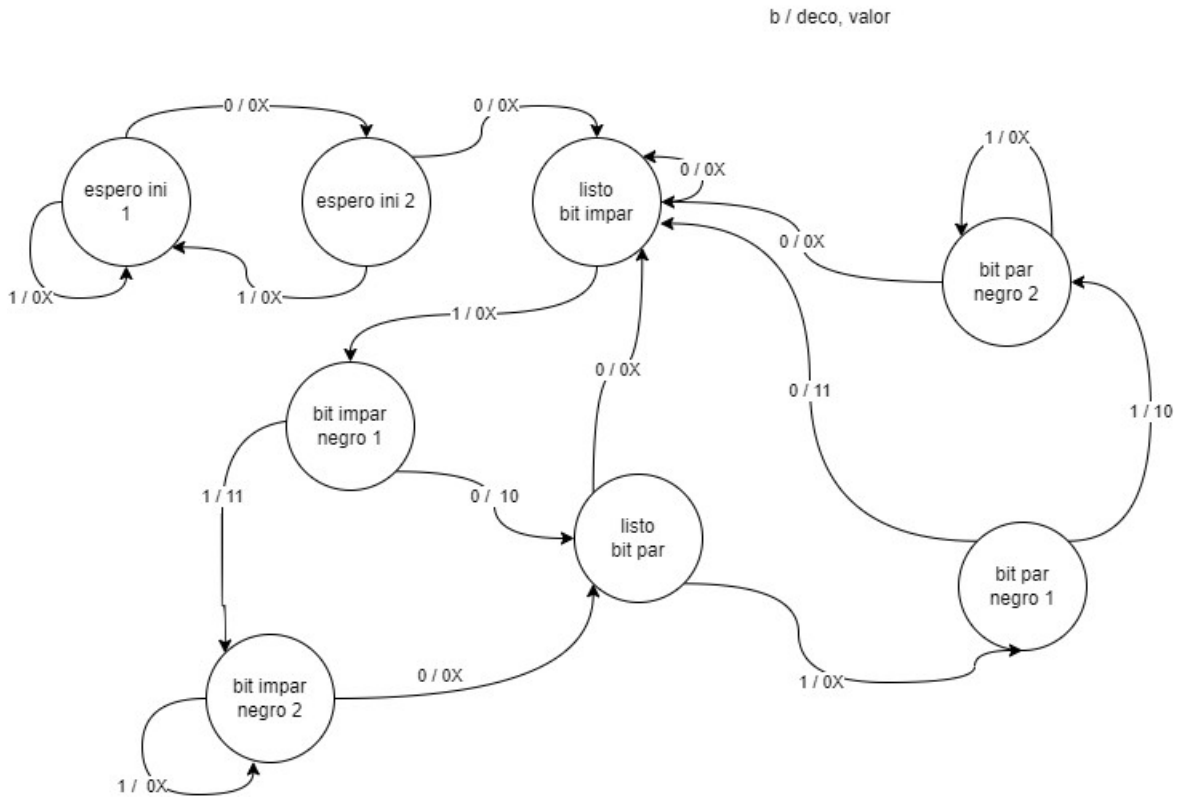


Tabla de estados:

Estado Actual	Entrada	Próximo Estado	Salida	
	b		**deco**	**valor**
Espero ini 1	0	Espero ini 2	0	X
Espero ini 1	1	Espero ini 1	0	X
Espero ini 2	0	Listo bit impar	0	X
Espero ini 2	1	Espero ini 1	0	X
Listo bit impar	0	Listo bit impar	0	X
Listo bit impar	1	Bit impar negro 1	0	X
Bit impar negro 1	0	Listo bit par	1	0
Bit impar negro 1	1	Bit impar negro 2	1	1
Bit impar negro 2	0	Listo bit par	0	X
Bit impar negro 2	1	Bit impar negro 2	0	X
Listo bit par	0	Listo bit impar	0	X
Listo bit par	1	Bit par negro 1	0	X
Bit par negro 1	0	Listo bit impar	1	1
Bit par negro 1	1	Bit par negro 2	1	0
Bit par negro 2	0	Listo bit impar	0	X
Bit par negro 2	1	Bit par negro 2	0	X

Estado inicial: espero ini 1.

Son 8 estados, por lo tanto se precisan 3 FF para codificarlos. Se dispone de flip flops tipo D, por lo tanto se utilizarán 3 FF tipo D.

Codificación de estados:

espero ini 1: 000
espero ini 2: 001
listo bit impar: 010
bit impar negro 1: 011
bit impar negro 2: 100
listo bit par: 101
bit par negro 1: 110
bit par negro 2: 111

Considerando que la ecuación del FF tipo D es Q(n+1) = D(n), se escribe la siguiente tabla de verdad

Tabla de Verdad:

Estado Actual - Q(n)	Entrada	Próximo Estado – Q(n+1) = D(n)	Salida	
q2q1q0	**b**	**d2d1d0**	**deco**	**valor**
000	0	001	0	X
000	1	000	0	X
001	0	010	0	X
001	1	000	0	X
010	0	010	0	X
010	1	011	0	X
011	0	101	1	0
011	1	100	1	1
100	0	101	0	X
100	1	100	0	X
101	0	010	0	X
101	1	110	0	X
110	0	010	1	1
110	1	111	1	0
111	0	010	0	X
111	1	111	0	X

Mapas K

q2q1\q0b	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	0	1	1	0
10	1	1	1	0

d2 = q2!q1!q0 + q2.b + !q2.q1.q0

q2q1\q0b	00	01	11	10
00	0	0	0	1
01	1	1	0	0
11	1	1	1	1
10	0	0	1	1

d1 = q2.q0 + q1!q0 + q0!b!q1

q2q1\q0b	00	01	11	10
00	1	0	0	0
01	0	1	0	1
11	0	1	1	0
10	1	0	0	0

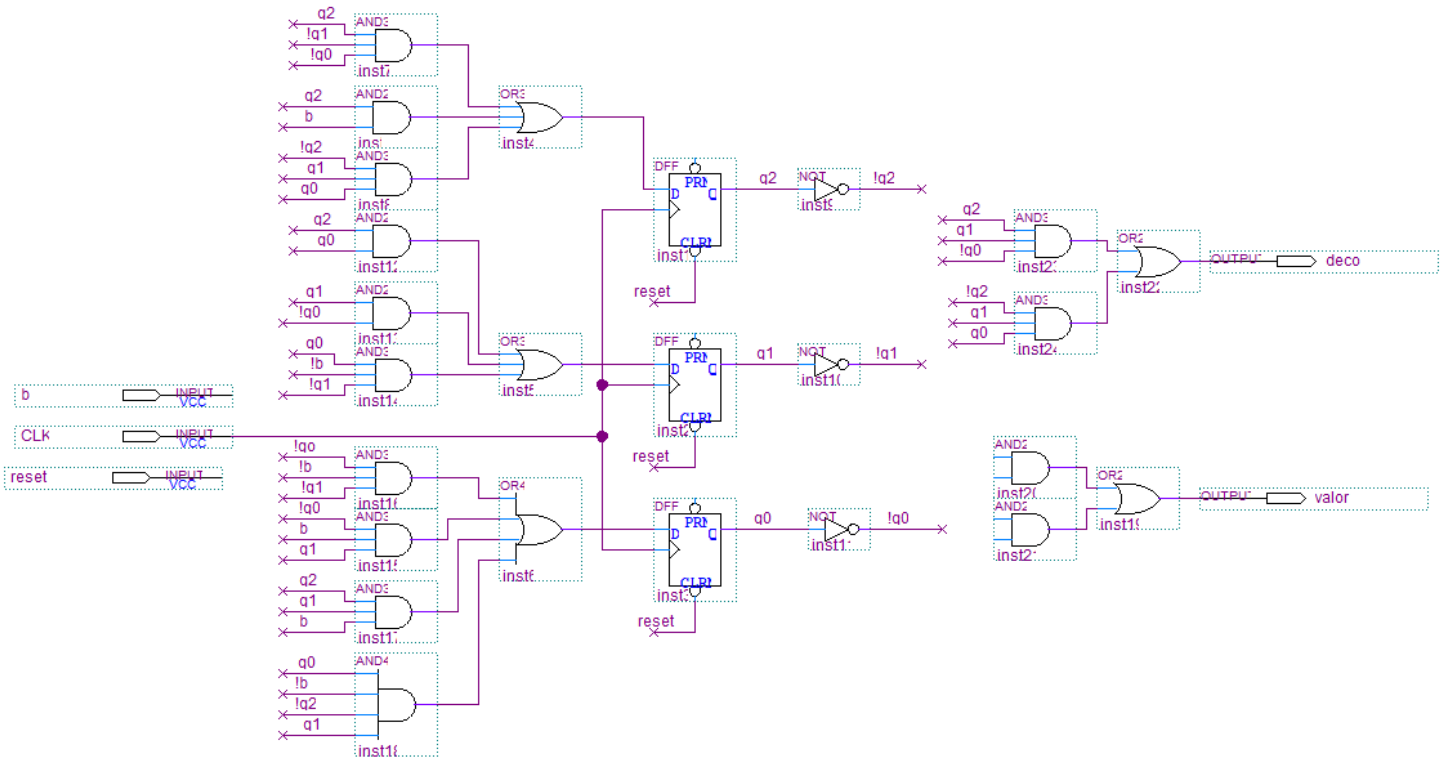
d0 = !q0!b!q1 + !q0.b.q1 + q2.q1.b + q0!b!q2.q1

q2q1\q0b	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	1	1	0	0
10	0	0	0	0

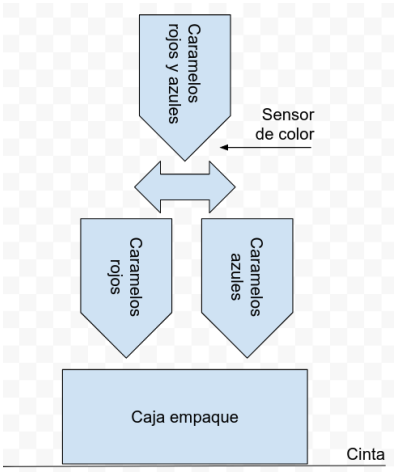
deco = q2.q1!q0 + !q2.q1.q0

q2q1\q0b	00	01	11	10
00	X	X	X	X
01	X	X	1	0
11	1	0	X	X
10	X	X	X	X

valor = !q0!b+ q0.b


b
CLK
reset
INPUT
VCC
q2
!q1
!q0
q2
b
!q2
q1
q0
q2
q0
q1
!q0
q0
!b
!q1
!q0
b
q1
q2
q1
b
q0
!b
!q2
q1
reset
reset
reset
q2
!q2
q1
q0
q1
!q1
q0
!q0
q2
q1
!q0
q2
q1
q0
OUTPUT
deco
OUTPUT
valor

Problema 2



Se desea construir un sistema que empaque en cajas caramelos de frutilla (rojos) y arándanos (azules) de forma balanceada. En cada caja se deben colocar **CANT_CARAMELOS** caramelos de cada color.

Se dispone de un sensor de color que devuelve el color del objeto en su zona de lectura (0 si es azul y 1 si es rojo). La lectura del sensor es ruidosa por lo que se deben tomar muestras y se deben obtener **MIN_*MUESTRAS*** *seguidas del mismo color para dar por v*álido el color de un caramelo. Cada vez que el sensor realiza una lectura nueva genera una interrupción que invoca a la rutina ***hayMuestra()***.

Desde la zona de lectura del sensor de color se debe guiar el caramelo al contenedor rojo o al contenedor azul según el color del mismo, accionando válvulas asociadas a la entrada de cada contenedor. Estos contenedores tienen una capacidad máxima de **MAX_CARAMELOS** caramelos. Si se llena alguno de los contenedores se debe detener el empaque y se debe prender una alarma a la espera de la intervención humana, que luego de corregir la situación reiniciará la máquina.

La caja se llena controlando las válvulas existentes entre los contenedores de caramelos rojos y azules, y la caja.
Una vez llena la caja se debe mover la cinta que la transporta hasta colocar una nueva caja para llenar.

Se interactúa con el hardware a través del puerto de E/S, de 8 bits de lectura y escritura en la dirección **ASTRA**:

- bit 0: indica la lectura del sensor de color (0 = rojo, 1 = azul).
- bit 1: controla apertura de la válvula desde el sensor de color al contenedor rojo.
- bit 2: controla apertura de la válvula desde el sensor de color al contenedor azul.
- bit 3: controla apertura de la válvula desde el contenedor rojo a la caja a empacar (0 cierra, 1 abre).
- bit 4: controla apertura de la válvula desde el contenedor azul a la caja a empacar (0 cierra, 1 abre).
- bit 5: controla la cinta transportadora de cajas (0 está quieta, 1 se mueve).
- bit 6: controla la alarma (1 prende la alarma)

Las válvulas que controlan el pasaje de un caramelo hacia los contenedores de color garantizan el pasaje de un único caramelo al recibir un flanco ascendente (transición de 0 a 1) en su señal de control.
Para dejar pasar un caramelo desde los contenedores de colores a la caja a empacar debe mantenerse abierta la válvula de salida del contenedor (camino del contenedor a la caja) por 100 ms.
Para cambiar la caja por una nueva se debe mantener encendida la cinta por 5 s.
Se dispone de un reloj externo que interrumpe al procesador con una frecuencia de 1 KHz que invoca a la rutina de interrupción ***timer()***.

Se pide: Implementar en un lenguaje de alto nivel el sistema propuesto sabiendo que el procesador está dedicado al control de la empaquetadora.

Solución

```
#define CANT_CARAMELOS ...
#define MAX_CARAMELOS ...
#define MIN_MUESTRAS ...
#define COLOR_AZUL 1
#define COLOR_ROJO 0

#define 100MS 100
#define 5SEG 5000

char colorMuestra = -1
int cantMuestrasIguales = 0;

int cantAzulesEnContenedor = 0;
int cantRojosEnContenedor = 0;
int cantAzulesEnCaja = 0;
int cantRojosEnCaja = 0;

char cargandoCarameloAzul = false;
char cargandoCarameloRojo = false;
char cambiandoCaja = false;

int timerCarameloAzul = 0;
int timerCarameloRojo = 0;
int timerCajaEmpaque = 0;

void main(){
      // instalar interrupciones
      out (ASTRA, 0);
      enable();

      while (true){
            if (cantAzulesEnContenedor > 0 && !cargandoCarameloAzul
                  && cantAzulesEnCaja < CANT_CARAMELOS){
                  timerCarameloAzul = 0;
                  cargandoCarameloAzul = 1;
                  OUT(ASTRA, IN(ASTRA) | 0x10);
            }
            if (cantRojosEncontenedor > 0 && !cargandoCarameloRojo
                  && cantRojosEnCaja < CANT_CARAMELOS){
                  timerCarameloRojo = 0;
                  cargandoCarameloRojo = 1;
                  OUT(ASTRA, IN(ASTRA) | 0x8);
            }

            if (!cambiandoCaja && cantAzulesEnCaja == CANT_CARAMELOS
                && cantRojosEnCaja == CANT_CARAMELOS){
                  cambiandoCaja = 1;
                  timerCajaEmpaque = 0;
                  OUT(ASTRA, IN(ASTRA) | 0x20);
            }
      }

}
```

```
interrupt void hayMuestra(){
      char datos = IN(ASTRA);
      char color = datos & 1;
      if (color != colorMuestra){
            cantMuestrasIguales = 0;
      }
      colorMuestra = color;
      cantMuestrasIguales++;

      if (cantMuestrasIguales == MIN_MUESTRAS){
            if (colorMuestra == COLOR_AZUL){
                  OUT(ASTRA, datos | 4);
                  cantAzulesEnContenedor++;
                  cantMuestrasIguales = 0;
            }else if (colorMuestra = COLOR_ROJO){
                  OUT(ASTRA, datos | 2);
                  cantRojosEnContenedor++;
                  cantMuestrasIguales = 0;
            }
            OUT(datos & 0xF9); // bajo a 0 ambas señales

            if (cantRojosEnContenedor >= MAX_CARAMELOS ||
               cantAzulesEnContenedor >= MAX_CARAMELOS){
                  OUT(ASTRA, 0x40);
                  while(true); // tranco el sistema en esta línea hasta
                               // intervención humana (reset)
            }

      }
}

interrupt void timer(){
      if (cargandoCarameloRojo){
            timerCarameloRojo++;
            if (timerCarameloRojo == 100MS){
                  OUT(ASTRA, IN(ASTRA) & 0xF7);
                  cargandoCarameloRojo = 0;
                  cantRojosEnContenedor--;
                  cantRojosEnCaja++;
            }
      }
      if (cargandoCarameloAzul){
            timerCarameloAzul++;
            if (timerCarameloAzul == 100MS){
                  OUT(ASTRA, IN(ASTRA) & 0xEF);
                  cargandoCarameloAzul = 0;
                  cantAzulesEnContenedor--;
                  cantAzulesEnCaja++;
            }
      }
      if (cambiandoCaja){
            timerCajaEmpaque++;
            if (timerCajaEmpaque == 5S){
                  OUT(ASTRA, IN(ASTRA) & 0xDF);
                  cambiandoCaja = 0;
                  cantAzulesEnCaja = 0;
                  cantRojosEnCaja = 0;
            }
      }
}
```