Examen de Arquitectura de Computadoras

19 de diciembre de 2019

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.**
- **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.**
- **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

Se considera la siguiente representación (en 8 bits) de enteros sin signo de escala variable:

- Los tres bits más significativos conforman un número E que indica la escala (2^E) que se le aplica a los cinco bits menos significativos.
- Los primeros 2^5 números se codifican con la escala E=0, los siguientes 2^6 números con escala E=1, los siguientes 2^7 números con escala E=2 y así sucesivamente.

La representación propuesta es una función en [0, …, MaxRep] → [0, …, 255]

a) Indique el rango de la representación (MaxRep), y si conserva el orden, la suma y la multiplicación.

b) Decodifique los números 00100001, 01100001

Pregunta 2

Indique cuál es el contenido de cada dirección de memoria del stack, SP y BP luego de ejecutar las siguientes instrucciones en 8086 para los siguiente valores iniciales: AX = 0x0ACA, BX = 0x00BA, CX = 0x0000, DX = 0xFEDE, SP = 0x3F00, BP = SP.

```
1.PUSH DX
2.MOV  CX, -1
3.PUSH CX
4.PUSH BP
5.MOV  BP, SP
6.MOV  [BP + 2], AX
7.MOV  [BP + 4], BX
```

Pregunta 3

Explique cuándo se utiliza un algortimo de sustitución en una memoria caché y para qué casos de correspondencia es necesario implementarlo.

Pregunta 4

a. Construir un flip flop D a partir de un flip flop JK.

b. Construir un flip flop T a partir de un flip flop JK.

Problema 1

Se propone diseñar un controlador de interrupciones enmascarables para una CPU que cuenta con una entrada de solicitud de interrupción (entrada de la CPU *interrupt request*, INTR, que es una salida del controlador) y una salida de aceptación de interrupción (salida de la CPU *interrupt acceptance*, INTA, que es una entrada del controlador). El controlador debe ser capaz de atender interrupciones generadas por tres fuentes externas (F2, F1 y F0), que utilizan sus propias señales de solicitud de interrupción (entradas del controlador IR2, IR1 e IR0). Las prioridades están definidas por prioridad(F0) > prioridad(F1) > prioridad(F2). El controlador considera también una entrada de tres bits *interrupt mask* (IM2, IM1, IM0) para enmascarar las interrupciones, de modo que cuando IMi = 0 la señal de solicitud IRi está enmascarada (implicando que la solicitud no debe ser atendida). Para informar a las fuentes F0, F1 y F2 que la CPU ha aceptado la interrupción se retorna el valor 1 en la posición correspondiente en un registro de aceptación (salidas del controlador IA2, IA1 e IA0). Un diagrama abstracto de entradas y salidas del controlador propuesto se presenta en la Figura 1.

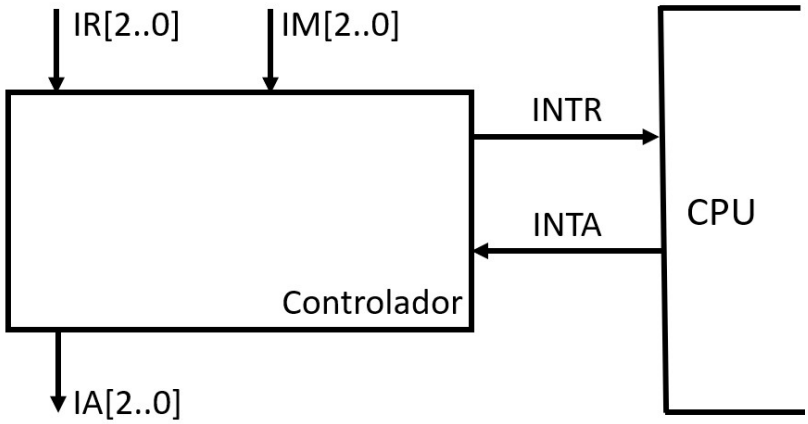


Figura 1. Diagrama abstracto de entradas y salidas del controlador propuesto.

Se solicita:

a) Construir la tabla de verdad reducida de un controlador de interrupciones que cumpla con la especificación propuesta.
b) Diseñar un circuito para implementar el controlador de interrupciones, utilizando compuertas NOT, compuertas AND (de dos o tres entradas) y ***solamente una*** compuerta OR de tres entradas.
c) Diseñar un circuito para un controlador capaz de atender interrupciones desde seis fuentes diferentes F0, F1, F2, F3, F4, F5, con prioridades definidas por prioridad(F0) > prioridad(F1) > prioridad(F2) > prioridad(F3) > prioridad(F4) > prioridad(F5).

Problema 2

Considere las siguientes estructuras de datos para almacenar árboles de enteros sin signo.

```
struct nodo_arbol {
      unsigned short int dato;
      nodo_arbol * hijo_izq;
      nodo_arbol * hijo_der;
} arbol[N];
```

y el siguiente seudo-código de la función que busca el mínimo y el máximo valor de dato en un árbol dado.

```
minimo, maximo busco_minmax(nodo_arbol * rama) {

      unsigned short int dato_min, dato_max, res_min, res_max;

      res_min = rama→dato;
      res_max = rama→dato;
      if (rama→hijo_izq != 0) {
            dato_min, dato_max = busco_minmax(rama→hijo_izq);
            if (dato_min < res_min) res_min = dato_min;
            if (dato_max > res_max) res_max = dato_max;
      }
      if (rama→hijo_der != 0) {
            dato_min, dato_max = busco_minmax(rama→hijo_der);
            if (dato_min < res_min) res_min = dato_min;
            if (dato_max > res_max) res_max = dato_max;
      }
      return(res_min, res_max);
}
```

Se solicita:

a) Escribir la función que busca el mínimo y el máximo valor de dato en un árbol dado en ensamblador 8086. Considere que la estructura del árbol está totalmente contenida en el segmento apuntado por el registro ES. El puntero argumento se pasa por stack, así como se devuelven en el stack el minimo y el máximo. La forma de invocar es del estilo:

```
PUSH “arbol”
CALL busco_minmax
POP “minimo”
POP “maximo”
```

b) Calcular el máximo valor de “N” para que la función pueda ejecutar siempre en una máquina 8086, justificando cual es la restricción que prevalece.

Solución Pregunta 1

a) Vamos a definir una función que mapea todos los números de [0..MaxRep] al rango [0..255]. A cada escala le asignamos un desplazamiento que hace que la función sea sobreyectiva. El mismo depende de la escala teniendo en cuenta que para cada escala hay 32 elementos.

Para E=0 el desplazamiento es 0 y se representaron los primeros 2^5 números (numera hasta 2^5-1).

Para E=1 el desplazamiento es $32*2^0$ y en esa escala el rango abarca $32*2^1$ números.

Para E=2 el desplazamiento es la suma de los anteriores, o sea $32*(2^0+2^1)$ y en esa escala el rango abarca $32*2^2$ números.

Se procede de la misma manera para cada escala con lo cual para E=7 el desplazamiento es 32*(1111111b), o sea 111111100000b

A su vez en esa escala el mayor número es $31*2^7$, o sea 111110000000b. Sumando queda 1111101100000b = 8032.

La representación conserva el orden pues para una misma escala los bits menos significativos indican el valor al cual aplicarle la escala (quedando entonces ordenados) y por construcción cada cada elemento de una escala superior mapea números mayores que cualquier elementos de las escalas inferiores.

No conserva ni la suma ni la multiplicación. Puede verse por ejemplo dado que los bits más significativos indican la escala cualquier suma hace que se multipliquen las escalas, y los bits menos significativos desbordan hacia la escala.

b)

00100001: Tiene escala 1 y valor 1 => n= 32+1*2 = 34

01100001: tiene escala 3 y valor 1 => n= 32+64+128 + 1*8 = 232

Solución Pregunta 2

Stack:

dir: SS:0x3EFA valor: 0x3F00
dir: SS:0x3EFC valor: 0x0ACA
dir: SS:0x3EFE valor: 0x00BA

BP = 0x3EFA
SP = 0x3EFA

Solución Pregunta 3

Los algoritmos de sustitución se utilizan cuando una línea debe ser cargada en caché, pero el conjunto correspondiente tiene todas las líneas ocupadas. En ese caso, se debe seleccionar alguna de

las líneas del conjunto para reemplazar, el algoritmo que indica cómo elegir dicha línea es el *algoritmo de sustitución*.

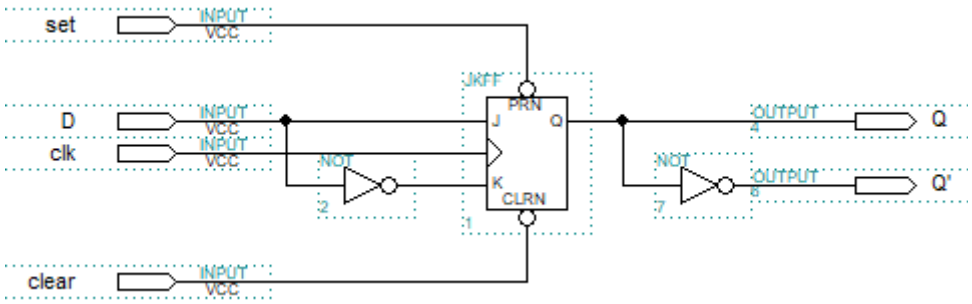
Los algoritmos de sustitución se utilizan para las funciones de correspondencia asociativa por conjuntos de N vías y totalmente asociativa, pues en ambos casos la caché se divide en conjuntos con más de una línea por conjunto (en el caso de la función de correspondencia totalmente asociativa, todas las líneas del caché forman un único conjunto).

Solución Pregunta 4

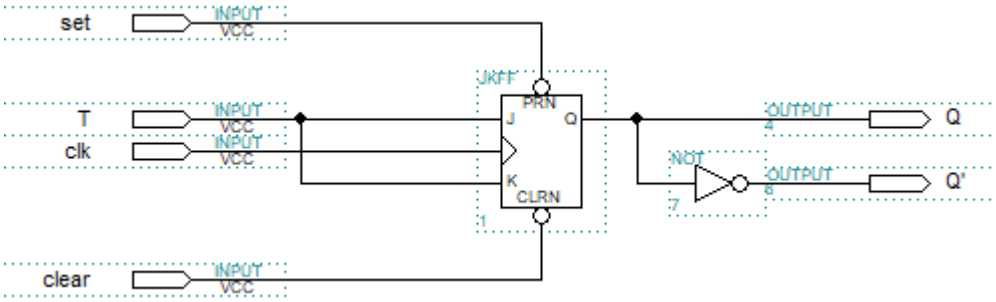
La tabla de la verdad reducida del flip flop JK es la siguiente:

J	K	Q(n+1)
0	0	Q(n)
0	1	0
1	0	1
1	1	Q(n)'

A partir de ella armamos los circuitos pedidos:
a) Colocamos J=D y K=D' y logramos cumplir la ecuación del FF-D: $Q_{n+1} = D_n$



b) Colocamos J=K=T y logramos cumplir la ecuación del FF-T: $Q_{n+1} = T_n'Q_n + T_nQ'_n$



Solución Problema 1

a) La tabla de verdad reducida de un controlador de interrupciones que cumpla con la especificación propuesta se construye de modo simple, analizando las entradas posibles y teniendo en cuenta las prioridades definidas para las solicitudes de interrupción provenientes de las fuentes externas. Las salidas IA_i solamente tomarán valores relevantes cuando la CPU acepte un pedido de interrupción (INTA=1), por lo cual la tabla se reduce significativamente. Solo es necesario considerar el caso con INTA=0 para determinar INTR. Luego, deben respetarse las prioridades y considerar la entrada de enmascaramiento, por lo cual la tabla se construye con un procedimiento sistemático que corresponde a "una interrupción se acepta cuando se realizó el pedido correspondiente, la interrupción no está enmascarada y no se esté solicitando al mismo tiempo una interrupción de mayor prioridad (que no esté enmascarada)". La tabla de verdad reducida del controlador propuesto se presenta en la Tabla 1.

Entradas							Salidas			
INTA	IM0	IM1	IM2	IR0	IR1	IR2	IA0	IA1	IA2	INTR
1	1	X	X	1	X	X	1	0	0	X
1	0	1	X	X	1	X	0	1	0	X
1	X	1	X	0	1	X	0	1	0	X
1	0	0	1	X	X	1	0	0	1	X
1	0	X	1	X	0	1	0	0	1	X
1	X	0	1	0	X	1	0	0	1	X
1	X	X	1	0	0	1	0	0	1	X
1	0	0	0	X	X	X	0	0	0	0
1	0	0	X	X	X	0	0	0	0	0
1	0	X	0	X	0	X	0	0	0	0
1	0	X	X	X	0	0	0	0	0	0
1	X	0	0	0	X	X	0	0	0	0
1	X	0	X	0	X	X	0	0	0	0
1	X	X	0	0	0	X	0	0	0	0
1	X	X	X	0	0	0	0	0	0	0
0	1	X	X	1	X	X	0	0	0	1
0	X	1	X	X	1	X	0	0	0	1
0	X	X	1	X	X	1	0	0	0	1

Tabla 1. Tabla de verdad reducida del controlador propuesto

b) De la tabla de verdad se deducen las expresiones algebraicas de las salidas para construir el circuito:

INTR = (IM0 . IR0 + IM1 . IR1 + IM2 . IR2). Tomamos los valores X de INTR de la tabla como "1" para simplificar (evitamos la dependencia con INTA). Para INTR se utiliza la compuerta OR de tres entradas disponible)

IA0 = INTA . IM0 . IR0

IA1 = INTA ($\overline{IM0}$. IM1 . IR1 + IM1 . $\overline{IR0}$. IR1) = INTA . IM1 . IR1 ($\overline{IM0}$ + $\overline{IR0}$) = (se aplica de Morgan para eliminar la suma, ya que solo se dispone de una compuerta OR de tres entradas) INTA . IM1 . IR1 . ($\overline{IM0 . IR0}$)

IA2 = INTA ($\overline{IM0}$. $\overline{IM1}$. IM2 . IR2 + $\overline{IM0}$. IM2 . $\overline{IR1}$. IR2 + $\overline{IM1}$. IM2 . $\overline{IR0}$. IR2 + IM2 . $\overline{IR0}$. $\overline{IR1}$. IR2) = INTA . IM2 . IR2 ($\overline{IM0}$. $\overline{IM1}$ + $\overline{IM0}$. $\overline{IR1}$ + $\overline{IM1}$. $\overline{IR0}$ + $\overline{IR0}$. $\overline{IR1}$) = INTA . IM2 . IR2 ($\overline{IM0}$ + $\overline{IR0}$) ($\overline{IM1}$ + $\overline{IR1}$) = (de Morgan para eliminar las sumas) INTA . IM2 . IR2 ($\overline{IM0 . IR0}$) ($\overline{IM1 . IR1}$)

El circuito se presenta en la Figura 2.

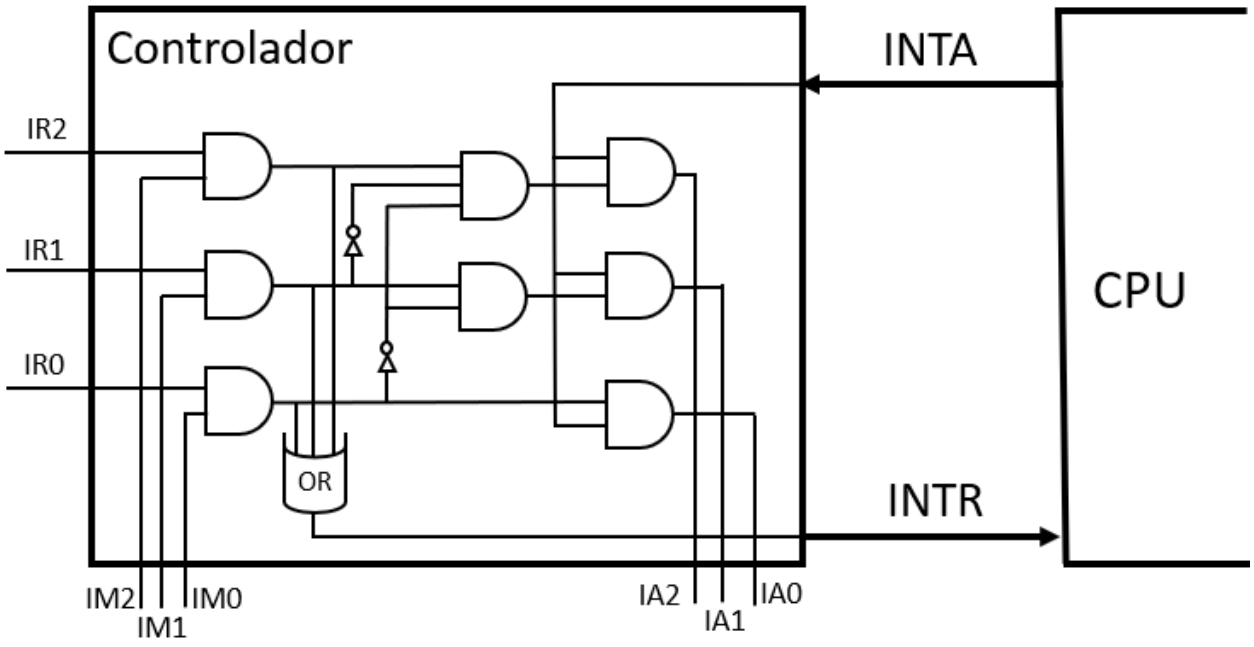


Figura 2. Diseño de un circuito para el controlador propuesto.

c) Para atender interrupciones provenientes desde más de tres fuentes se puede implementar un mecanismo de *daisy chaining* utilizando el controlador propuesto. Simplemente se conectan en serie tres controladores, según el diagrama que se muestra en la Figura 3.

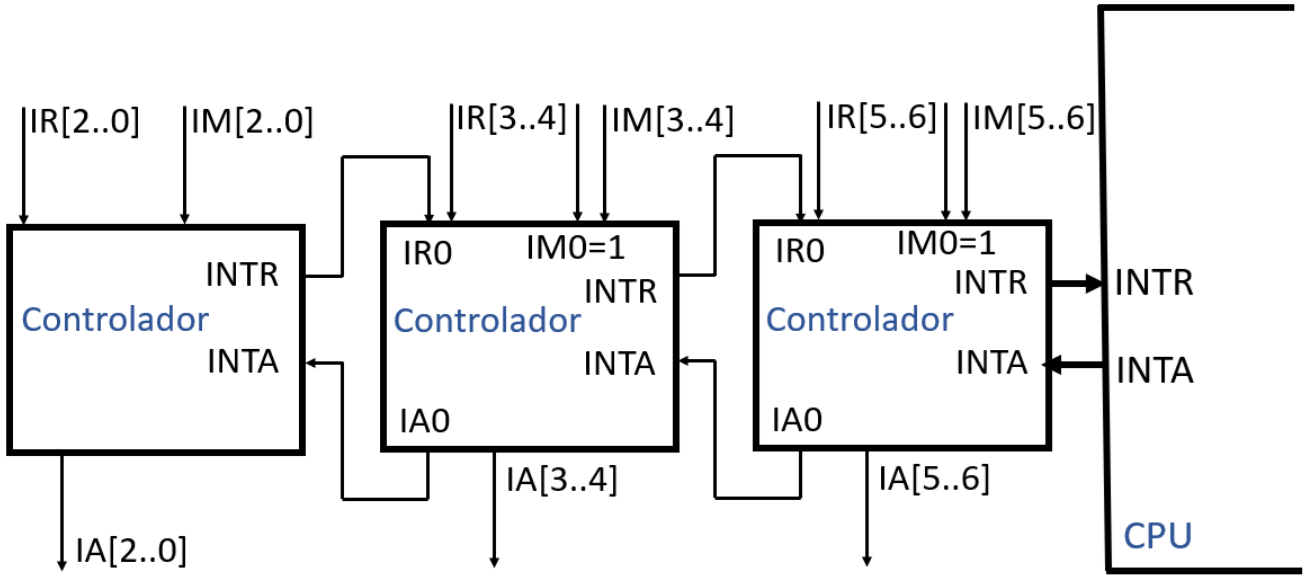


Figura 3. Diseño de un circuito para atender interrupciones desde seis fuentes.

En el caso propuesto se necesitan tres circuitos controladores en cascada, ya que el primero permite atender tres interrupciones, el segundo permite atender dos (ya que su primer entrada está conectada al primer controlador) y el tercero permite atender la sexta. Se podría atender una interrupción más, en caso de ser necesario.

Solución Problema 2

a)

```
busco_minmax    PROC
        PUSH BP         // hago lugar para segundo resultado
        PUSH BP
        MOV BP, SP
        PUSH AX         // minimo
        PUSH BX         // puntero al nodo
        PUSH CX         // maximo
        PUSH DX

        MOV BX, [BP + 6]
        MOV AX, ES:[BX]
        MOV CX, ES:[BX]
        CMP WORD PTR ES:[BX + 2], 0
        JZ pruebo_derecha

        // llamada recursiva izquierda
        PUSH ES:[BX + 2]
        CALL busco_minmax
        POP DX          // minimo
        CMP DX, AX
        JA sigo_1
        MOV AX, DX
sigo_1:
        POP DX          // maximo
        CMP DX, CX
        JB sigo_2
        MOV CX, DX
sigo_2:

pruebo_derecha:
        CMP WORD PTR ES:[BX + 4], 0
        JZ fin

        // llamada recursiva derecha
        PUSH ES:[BX + 4]
        CALL busco_minmax
        POP DX          // minimo
        CMP DX, AX
        JA sigo_3
        MOV AX, DX
sigo_3:
        POP DX          // maximo
        CMP DX, CX
        JB sigo_4
        MOV CX, DX
```

```
sigo_4:
fin:
        MOV DX, [BP + 4]
        MOV [BP + 2], DX    // acomodo dirección de retorno
        MOV [BP +4], AX
        MOV [BP +6], CX
        POP DX
        POP CX
        POP BX
        POP AX
        POP BP
        RET
busco_minmax        ENDPROC
```

b)

El tamaño del arreglo de nodos (N) está limitado por las siguientes restricciones:

- la estructura debe caber en un segmento de 64 Kbytes. Dado que cada nodo tiene 6 bytes (2 de datos, 2 de puntero izquierdo y 2 de puntero derecho) $6 * N \leq 65536$ (*) => $N < 10923$
- el tamaño del stack está limitado también a un segmento, por lo que se debe analizar el consumo del stack en el peor caso.
  En el peor caso el árbol degenera en una lista de N nodos. Para recorrer la lista se necesita una llamada inicial y (N–1) llamadas recursivas. Todas las llamadas consumen 8 palabras de stack, es decir 16 bytes. Por tanto $16 * N < 65356$ => $N < 4096$.

La restricción más significativa, en este caso, es la segunda, por lo que debe ser $N < 4096$.

(*) se podría argumentar que la posición 0 del segmento no puede usarse porque el 0 es usado como indicador de fin para los punteros. En esta oportunidad el código de alto nivel propuesto admite que el árbol comience en la posición 0 (no así ninguna de sus ramas hijas), por lo que habría un caso en que se puede usar esa posición. De todos modos debido a que 65536 no es divisible entre 6, esta consideración no cambia el valor del límite que se haya por el criterio del tamaño de la estructura.