Examen de Arquitectura de Computadoras

19 de febrero de 2015

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

Describa el mecanismo de atención a interrupciones en un computador 8086 que dispone de controlador de interrupciones

Respuesta:

Un microprocesador 8086, al recibir un pedido de atención a la interrupción, realiza los siguientes pasos (en caso que las interrupciones estén habilitadas):

- Termina de ejecutar la instrucción actual (en realidad verifica la existencia del pedido al final de un ciclo de instrucción y antes de comenzar el siguiente).
- Obtiene el identificador de la interrupción suministrado por el controlador de interrupciones en el bus de datos, luego de activa la señal INTA.
- Preserva la dirección de la próxima instrucción a ejecutarse y el estado del procesador, salvando en el stack el valor actual de los registros IP, CS, y FLAGS.
- Enmascara las interrupciones
- Accede al vector de interrupciones utilizando el identificador como indice (realiza accesos a memoria, en las direcciones *id_interrupt* * 4 y *id_interrupt * 4 + 2)* para obtener el desplazamiento y segmento de la dirección de la primera instrucción de la rutina de servicio de la interrupción.
- Realiza un jump far a la dirección obtenida en el vector de interrupciones.

Pregunta 2

Indique dos diferencias entre CPUs construidas por lógica cableada y microprogramada. ¿Cuál implementación se usa más comúnmente en RISC? ¿Y en CISC? Justifique

Respuesta:

Dos diferencias entre CPUs construidas por lógica cableadas y microprogramadas son:

Las construidas con “lógica cableada” son de más rápida ejecución (generalmente se minimiza el número de ciclos de reloj en el ciclo de instrucción). Las construidas por “lógica microprogramada” generalmente exigen de mayor cantidad de pasos (ver MIC-1, donde la decodificación es secuencial) y por tanto exigen más ciclos de reloj para ejecutar una instrucción.

Otra diferencia es que las construidas por lógica microprogamada son mucho más flexibles a la

hora de modificar el set de instrucciones que implementan, ya que para éstas el proceso implica modificar el contenido de la ROM que almacena el microprograma, mientras que para las de lógica cableada implica el re-diseño del circuito prácticamente desde cero.

La primer diferencia se alinea perfectamente con la filosofía RISC (instrucciones simples de rápida ejecución), y por eso se suele implementar los CPUs RISC de esta manera.

En CISC se utlilizan normalmente CPUs microprogramadas, ya que por la complejidad de las instrucciones, es más práctico implementarlas usando un autómata más simple que las implemente mediante un microprograma.

Pregunta 3

¿Qué son los hazards de control? Explique por qué la predicción de saltos ayuda a mitigar los hazards de control.

Respuesta:

Los hazards de control son uno de los tipos de obstáculos o problemas que aparecen al utilizar procesadores con pipelines. Ocurren cuando el hardware encuentra que el proceso de instrucciones que viene realizando no se corresponde con la secuencia lógica de ejecución del programa, en función de la presencia de una instrucción de salto en el mismo. Esto es debido a que en las etapas tempranas del pipeline aún no se conoce si un salto será tomado (y muy tempranamente tampoco se conoce siquiera si la instrucción **es** un salto!). Las instrucciones que están siendo procesadas que no corresponden deben ser descartadas, “vaciando” el pipeline y perjudicando así su rendimiento.

La predicción de saltos consiste en realizar la decisión de qué rama seguir en etapas anteriores del pipeline a la de la ejecución propiamente dicha del salto. De este modo, se reduce el número de instrucciones cargadas que deban ser eliminadas, siendo el peor caso aquel en que la predicción falla y el resultado es igual a que no se tuviera ningún tipo de predicción.

Pregunta 4

¿Con qué tipo de memoria está asociado el concepto de *refresh*? ¿Por qué es necesario?

Respuesta:

El concepto de refresh está asociado a las memorias *DRAM (Dynamic RAM).*

El circuito de “refresco” es necesario pues por la forma en la que se guardan los datos en esta memoria (almacenamiento de carga eléctrica en transistores), los mismos se pierden con el tiempo (debido a la existencia de “corrientes de fuga”). El “refresco” consiste en leer toda la memoria periódicamente, ya que por ser la lectura “destructiva”, la propia circuitería de la memoria prevé re-escribir el valor de un bit leído (en forma transparente para el resto del sistema). Para minimizar la cantidad de lecturas requeridas para el “refresco” la memoria DRAM está organizada en forma matricial y alcanza con activar una columna por vez para que se “refresquen” todos los bits de la misma.

Problema 1

Se desea construir un circuito con 1 bit de entrada y 4 bits de salida, que dada una señal de entrada de 3 bits serial (la señal de 3 bits entra en 3 periodos consecutivos de reloj), en el periodo de reloj correspondiente al último bit de la señal de entrada, la salida sea igual a la señal de entrada serial. La salida antes del último bit de la señal de entrada es indeterminada. El bit más significativo deberá indicar cuando la salida es válida (en el ultimo periodo de la entrada serial) o no (en los dos periodos restantes)

Ejemplo:

tiempo		0	1	2	3	4	5	...
entrada serial		0	1	0	1	1	0	
Salidas	válida	0	0	1	0	0	1	
	O_2	X	X	0	X	X	0	
	O_1	X	X	1	X	X	1	
	O_0	X	X	0	X	X	1	

Se pide:

Diseñar y dibujar el circuito utilizando la metodología del curso (máquina de estados, tabla de estados, codificación, tabla de transiciones/tablas de verdad de flip-flops y salidas, minimización de expresiones, dibujo del circuito). Se dispone de Flip-Flops tipo D y compuertas básicas.

Solución Problema 1

Entradas: X
Salidas: $Y_2Y_1Y_0V$

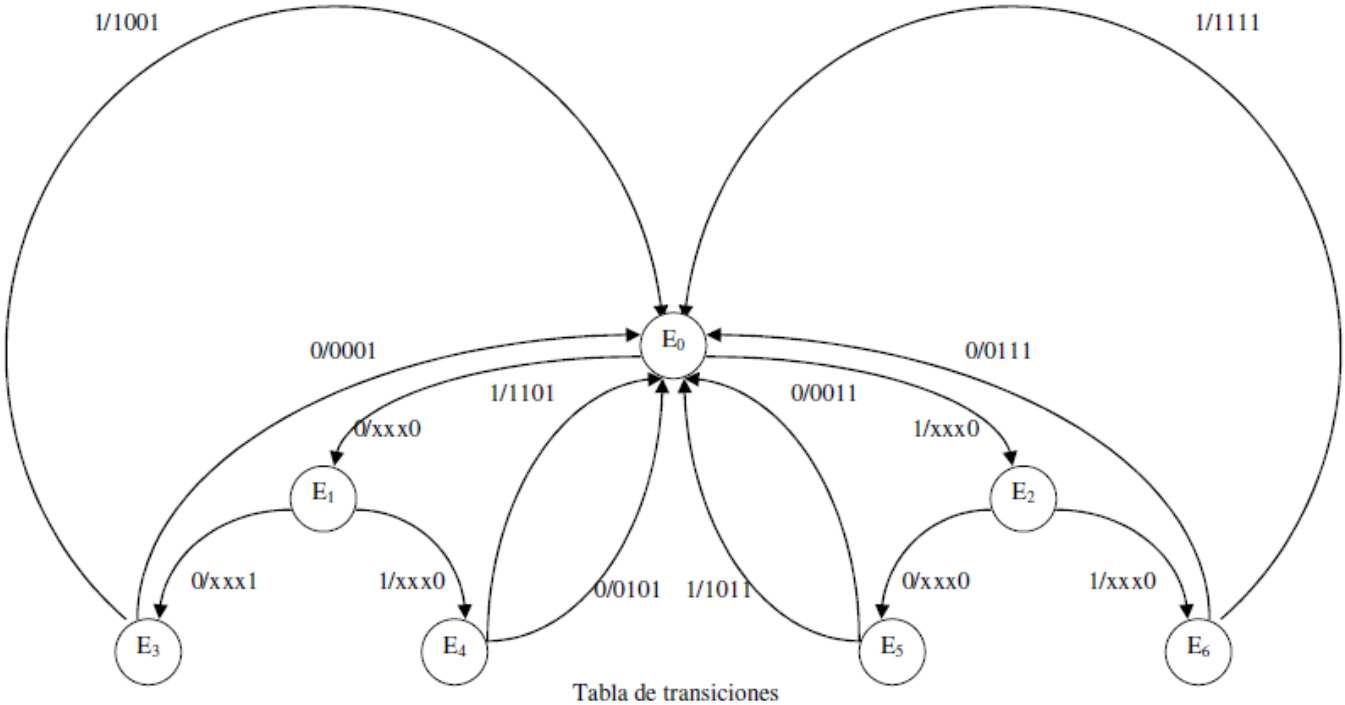


Tabla de transiciones

Estados	Codificacion	X=0	X=1	X=0	X=1
	$Q_2Q_1Q_0$	$D_2D_1D_0$	$D_2D_1D_0$	$Y_2Y_1Y_0V$	$Y_2Y_1Y_0V$
E_0	000	001	010	xxx0	xxx0
E_1	001	011	100	xxx0	xxx0
E_2	010	101	110	xxx0	xxx0
E_3	011	000	000	0001	1001
E_4	100	000	000	0101	1101
E_5	101	000	000	0011	1011
E_6	110	000	000	0111	1111
	111	xxx	xxx	xxxx	xxxx

$Q_2Q_1\backslash Q_0X$	00	01	11	10
00	0	0	1	0
01	1	1	0	0
11	0	0	x	x
10	0	0	0	0

$D_2 = Q_2' Q_1' Q_0 X + Q_2' Q_1 Q_0'$

$Q_2Q_1\backslash Q_0X$	00	01	11	10
00	x	x	x	x
01	x	x	0	0
11	1	1	x	x
10	0	0	1	1

$Y_0 = Q_2 Q_1 + Q_1 Q_0$

$Q_2Q_1\backslash Q_0X$	00	01	11	10
00	0	1	0	1
01	0	1	0	0
11	0	0	x	x
10	0	0	0	0

$D_1 = Q_2' Q_1' Q_0 X' + Q_2' Q_1' X$

$Q_2Q_1\backslash Q_0X$	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	1	1	x	x
10	1	1	1	1

$V = Q_2 + Q_1 Q_0$

$Q_2Q_1\backslash Q_0X$	00	01	11	10
00	1	0	0	1
01	1	0	0	0
11	0	0	x	x
10	0	0	0	0

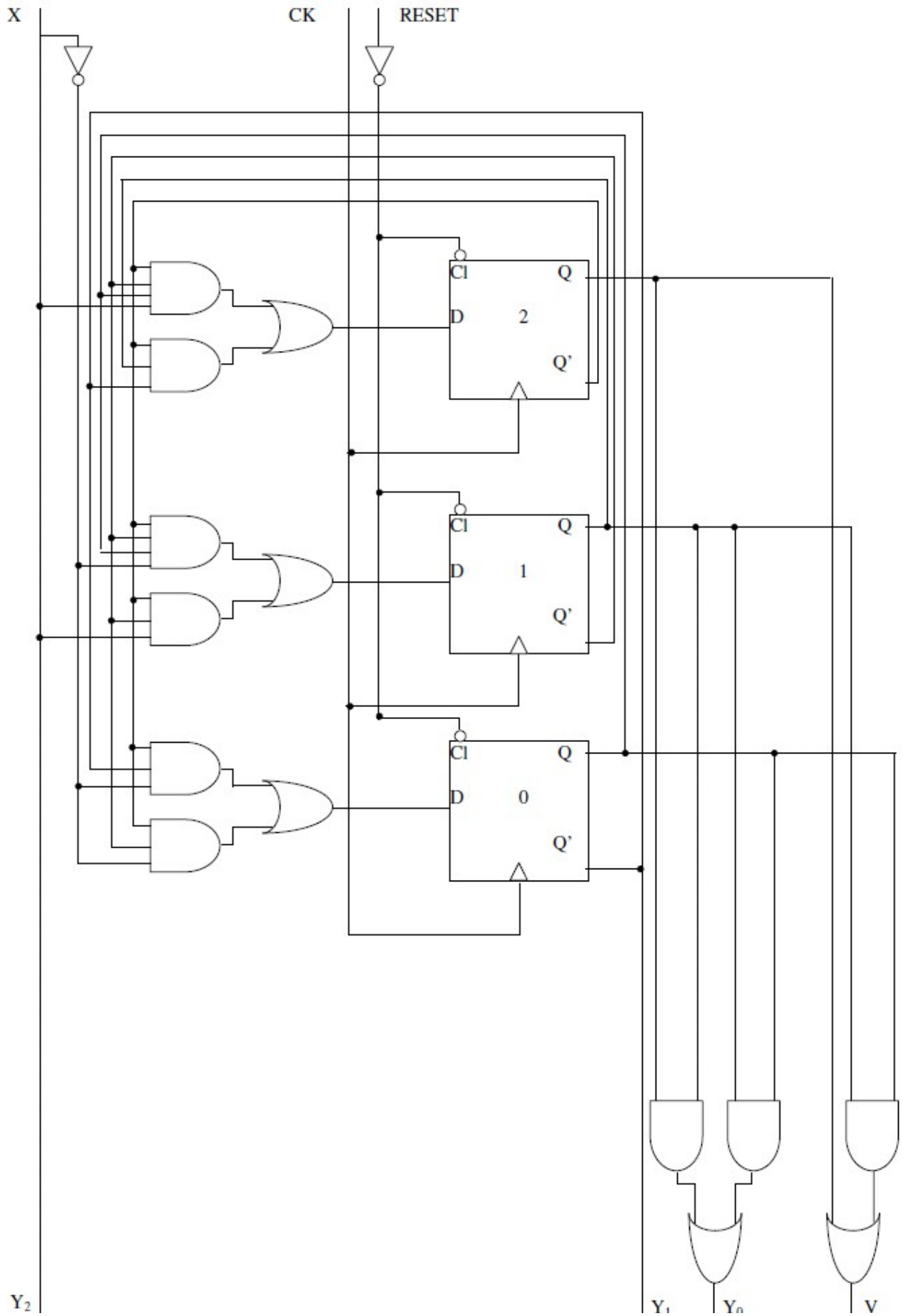
$D_0 = Q_2' Q_0' X' + Q_2' Q_1' X'$

$Q_2Q_1\backslash Q_0X$	00	01	11	10
00	x	x	x	x
01	x	x	1	0
11	0	1	x	x
10	0	1	1	0

$Y_2 = X$

$Q_2Q_1\backslash Q_0X$	00	01	11	10
00	x	x	x	x
01	x	x	0	0
11	1	1	x	x
10	1	1	0	0

$Y_1 = Q_0'$


X
CK
RESET
Cl
Q
D
2
Q'
Cl
Q
D
1
Q'
Cl
Q
D
0
Q'
Y2
Y1
Y0
V

Problema 2

Suponga un procesador 8086 y el siguiente fragmento de código:

```
void funcion(short largo){
	short i, j;
	for (i = 0; i < largo; i++){
		short suma = 0;
		for (j = 0; j < largo; j++){
			if (i != j){
				suma += MATRIZ[i][j];
			}
		}
		MATRIZ[i][i] = suma;
	}
}
```

Donde MATRIZ es una matriz de variables de tipo short, que se ubica en memoria por filas (es decir, primero en memoria el valor MATRIZ[0][0], luego MATRIZ[0][1], así hasta completar la primer fila y luego MATRIZ[1][0], MATRIZ[1][1], etc).

Se pide:

Parte a) .

Compilar el código a assembler 8086. Suponga que la matriz está ubicada en ES:[BX], y que el parámetro 'largo' se recibe en DX

Se debe preservar el valor de todos los demás registros.

Parte b) .

Suponga que entre el CPU y la memoria principal se coloca un cache L1, asociativo por conjuntos de dos vías, con capacidad de 16KB y 32 conjuntos en total.

Sabiendo que el parámetro *largo = 128*, calcule el hit rate en la ejecución del código anterior. Ignore los accesos a memoria producidos por fetch de instrucciones. Suponga que el valor de ES es tal que ES:BX coincide con el inicio de una línea de cache.

Solución

Parte a)

```
funcion PROC

PUSH CX
PUSH AX
PUSH SI
PUSH DI

XOR SI, SI      ; i = 0

     for:
     CMP SI, DX
     JGE fin

     XOR CX, CX   ;_suma = 0

     XOR DI, DI    ; j = 0

          for2:

          CMP DI, DX    ; for (j = 0; j < largo ....
          JGE finfor2

          CMP DI, SI      ; if ( i != j)
          JE post_for2

          ; armo en AX el offset hasta MATRIZ[i][0]
          MOV AX, SI       ;  AX = i
          PUSH DX; preservo DX porque se modifica en la multiplicación de 16 bits
          MUL DX             ;  AX = i * largo
          POP DX
          SHL AX, 1          ;  ahora ES:AX apunta a MATRIZ[i][0]
          ADD AX, DI        ;  ( DI = j )
          ADD AX, DI        ;  ahora ES:AX apunta a MATRIZ[i][j]

          ADD BX, AX

          ADD CX, ES:[BX]

          SUB BX, AX
      post_for2:
          INC DI

          JMP for2

     finfor2:

     ; matriz[i][i] = suma

     SUB AX, DI
     SUB AX, DI    ; ES:AX apunta a MATRIZ[i][0]
```

```
      ADD AX, SI
      ADD AX, SI   ; ES:AX apunta a MATRIZ[i][i]

      ADD BX, AX
      MOV ES:[BX] , CX
      SUB BX, AX

      INC DI

      JMP for

fin:

POP DI
POP SI
POP AX
POP CX

RET

ENDP
```

Parte b)

Como el cache tiene una capacidad de 16KB, y este tiene 32 conjuntos de 2 vías, el cache tiene 64 líneas. Por tanto, cada línea guarda 16KB / 64 = 256 bytes.

Sabiendo que cada línea ocupa 256 bytes, se necesitan 8 bits para indicar el byte dentro de la línea. Como hay 32 conjuntos, se precisan 5 bits para indicar el conjunto. Por tanto, la dirección de memoria se interpreta de la siguiente manera:

Dirección de memoria: tag (7 bits) | set (5 bits) | byte (8 bits)

Según el código, la matriz se recorre secuencialmente, exceptuando la diagonal (primero MATRIZ[0][1], luego MATRIZ[0][2], etc). Al llegar al final de la línea, se escribe la suma en MATRIZ[i][i] (diagonal). El primer acceso a MATRIZ[0][1] es **miss** por no estar la matriz en memoria. Esto hace que se cargue la línea correspondiente con los valores de MATRIZ[0][0], MATRIZ[0][1] hasta MATRIZ[0][127] (porque cada posición ocupa 2 bytes). De este modo, los siguientes 127 accesos son **hit** (las 126 lecturas restantes y la escritura en MATRIZ[0][0]).

Este comportamiento se repite en todas las filas de la matriz. De cada 128 accesos, 127 son hit y 1 es miss.

Resumiendo:

Se recorre toda la matriz (128 * 128) accesos. Si bien en el doble for se omite la diagonal, la misma se asigna al final.

Cantidad de accesos = 128 * 128
Cantidad de hits = 128 * 127

Hit rate = (128 * 127) / (128 * 128) = 127 / 128 = 0.9921875