Arquitectura de Computadores – IIC2343

Examen

21 noviembre 2017

1. Un lenguaje de programación de alto nivel para multiprocesadores (varias CPUs que comparten una memoria común a través de un bus) ofrece la siguiente instrucción indivisible **exchange**, definida como el intercambio indivisible de los valores de dos variables (indivisible o ininterrumpible: si un proceso inicia la ejecución de **exchange**, entonces nada lo detiene o interrumpe hasta que haya terminado de ejecutarla, es decir, hasta que haya intercambiado los valores de las variables):

```
exchange(a, b):
   temp = a
   a = b
   b = temp
```

a) Desarrolla una solución al problema de la *sección crítica* para dos procesos usando **exchange**. Recuerda: la sección crítica es una secuencia de una o más instrucciones (no nos importa cuáles) que puede ejecutar un proceso, tal que si el proceso está ejecutando su sección crítica, entonces ningún otro proceso puede estar ejecutando su propia sección crítica al mismo tiempo (para mantener la corrección de los datos compartidos).

Respuesta. La solución clásica; puede haber otras:

```
C = 1; local1 = 0; local2 = 0
```

process P1:	process P2:
`while (true):`	`while (true):`
`  while (local1 == 0):`	`  while (local2 == 0):`
`  exchange(C, local1)`	`  exchange(C, local2)`
sección crítica	*sección crítica*
`exchange(C, local1)`	`exchange(C, local2)`

b) Da argumentos lógicos que aseguren que tu solución cumple las propiedades de i) *exclusión mutua* (a lo más un proceso puede estar ejecutando su sección crítica al mismo tiempo); y ii) *ausencia de deadlocks* (en ningún momento ocurre que un proceso está esperando que el otro haga algo y al mismo tiempo el segundo proceso está esperando que el primero haga algo).

Respuesta.

i) Se satisface exclusión mutua. La única forma en que el proceso `Pi` puede entrar a su sección crítica es que su variable `locali` valga 1; pero la única forma en que esa variable puede llegar a valer 1 es que adquiera ese valor al ejecutarse la instrucción `exchange(C, locali)` dentro del `while (locali == 0)`; y, si esto ocurre, entonces la variable `C` queda con el valor 0. Como la instrucción `exchange` es indivisible, aunque ambos procesos estén ejecutando el `while` "al mismo tiempo", solo uno va a tener éxito en cambiar su `locali` a 1 y dejar `C` en 0, con lo cual el otro proceso no va a poder poner su `locali` en 1. Además, la única forma en que `C` vuelva a valer 1 es que el proceso que sale de su sección crítica ejecute la instrucción `exchange` nuevamente.

ii) Los procesos esperan a que algo pase únicamente cuando ejecutan el `while (locali == 0)` (la sección crítica, por definición, son sentencias que se deben ejecutar bajo exclusión mutua, por lo que la espera ocurre antes, justamente para garantizar la exclusión mutua; y el último `exchange`, al salir de la sección crítica, claramente no depende de ninguna condición para su ejecución). Así, para que ambos procesos estén esperando, necesariamente deben estar ejecutando **`while (locali == 0)`**; pero la única forma en que ambos **`locali`** sean permanentemente 0 es que este sea el valor que tiene la variable **`C`** cada vez que se ejecuta el **`exchange`** (supeditado al **`while`**), y la única forma en que **`C`** valga 0 es que al menos una de las **`locali`** valga 1 (*), contradiciendo la suposición de que ambos **`locali`** son permanentemente 0.

(*) Esto es cierto al comienzo, por la asignación inicial a **`C`**, **`local1`** y **`local2`**; y es cierto cada vez que un proceso termina de ejecutar su sección crítica, instante en el cual su variable **`locali`** es todavía 1, y ejecuta **`exchange`**.

2. En un multiprocesador (varias CPUs que comparten una memoria común a través de un bus), cada CPU puede tener su propia *memoria caché* para así evitar tener que recurrir frecuentemente a la memoria principal a través del bus. Sin embargo, esto normalmente da origen al *problema de coherencia de cachés*. Para solucionar estos problemas, los controladores de cachés son diseñados de manera que "observen" (*snoop*) las solicitudes que pasan por el bus (y que fueron hechas por alguna otra caché) y que hagan algo en ciertos casos. El conjunto de reglas que define qué hacer y cuándo se llama *protocolo de consistencia de cachés*.

a) Considera el protocolo *write through* estudiado en clase, cuya esencia es que todas las operaciones de escritura resultan en que la palabra que está siendo escrita en la caché también es escrita en la memoria para mantener la memoria actualizada todo el tiempo. Si el controlador de caché solo pudiera observar las líneas de dirección del bus, y no las de datos, ¿se vería el protocolo *write through* afectado por esta situación? Explica.

Respuesta.

No. En el protocolo *write through* básico, cuando un controlador de caché observa que una dirección de memoria está siendo escrita por otra CPU, chequea su propio caché para ver si tiene esa misma dirección, y, si la tiene, entonces invalida la línea correspondiente, pero no la actualiza (hasta que su propia CPU lea esa misma dirección más adelante, en cuyo caso el controlador va a buscar la línea a la memoria). Por lo tanto, para que el protocolo funcione correctamente, lo que importa es que el controlador de caché sepa cuál dirección está siendo escrita, pero no es necesario que sepa con qué valor.

Nota. La respuesta podría ser sí, si se estuviera hablando de la versión del protocolo en que el controlador actualiza la línea de su caché en lugar de invalidarla; en este caso, el protocolo se vería afectado, ya que tendría que ser modificado para funcionar como en la versión básica.

b) En los protocolos de tipo *write-back* no todas las escrituras van directamente a la memoria: cuando una línea de la caché es modificada, se pone un bit de la caché en 1 indicando que la línea de la caché está correcta pero la memoria no; finalmente, la línea es escrita en la memoria, pero posiblemente después de sufrir varias escrituras. El protocolo *MESI* estudiado en clase define cuatro estados para cada línea de la caché: *modified*, *exclusive*, *shared* (compartido), *invalid*. Si solo pudiéramos tener tres estados, ¿cuáles estados podrían ser eliminados (solo uno a la vez) y cuáles serían las consecuencias en cada caso?

Respuesta.

Eliminar el estado *invalid* es una mala idea: exigiría mantener todas las líneas de todas las cachés permanentemente actualizadas.

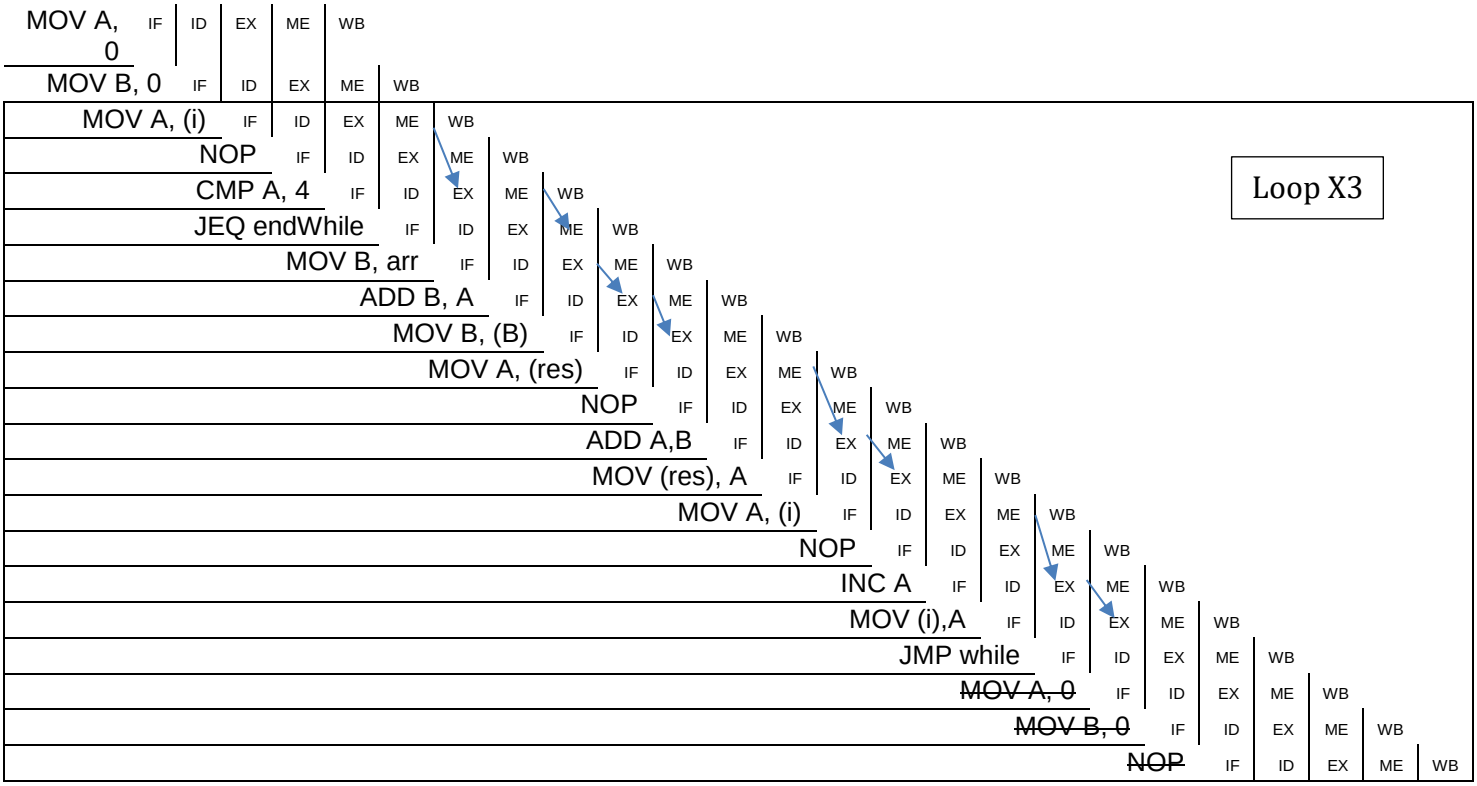
En cambio, los estados *exclusive* y *shared* pueden combinarse en uno, con comportamiento *shared*: la línea con el valor más reciente está en una o más cachés y la memoria está actualizada; si cualquiera de las CPUs involucradas escribe (cambia un valor de) la línea en su caché, entonces le avisa a las otras (aunque no haya "otras") que invaliden sus líneas, y pasa a estado *modified*, es decir, solo mi línea es válida e incluso la memoria es inválida.

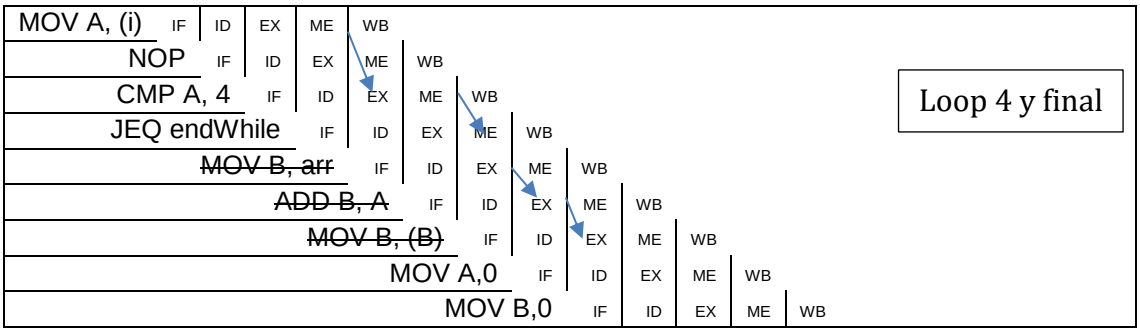
El estado *modified* puede eliminarse —haciendo que la memoria esté siempre actualizada— pero el costo es que el protocolo deja de ser *write-back* y pasa a ser *write through*.

3. Para el siguiente programa en *assembly*, rellena el diagrama del pipeline del computador básico e indica todos los *forwards* necesarios durante la ejecución. Considera que el manejo de *stalling* se realiza mediante NOP en tiempo de compilación, la predicción de saltos es (estáticamente) que no salta, y hay *forwarding* entre todas las etapas. Indica claramente todos los *stalls*, *flushs* y *forwards*.

```
DATA                          CODE
        res   0                       MOV A,0
        arr   1                       MOV B,0
              2                 while:
              3                       MOV A, (i)
              4                       CMP A, 4
        i     0                       JEQ endwhile
                                      MOV B, arr
                                      ADD B, A
                                      MOV B, (B)
                                      MOV A, (res)
                                      ADD A, B
                                      MOV (res), A
                                      MOV A, (i)
                                      INC A
                                      MOV (i), A
                                      JMP while
                                endwhile:
                                      MOV A, 0
                                      MOV B, 0
```

Respuesta.





4. Tú tienes un escáner que puede digitalizar imágenes de 256x256 pixeles en colores. Para ello, el escáner produce para cada pixel tres números entre 0 y 255 (correspondientes a las intensidades de los colores rojo, verde y azul) que coloca en tres *buffers* de datos (uno para cada color). El escáner se conecta mediante *memory-mapped I/O* al computador. El escáner cuenta con un digitalizador que digitaliza una línea de la imagen a la vez; la posición del digitalizador sobre la imagen es controlada por un motor, que recibe como parámetro la línea que debe digitalizar y posiciona al digitalizador donde corresponda; el digitalizador envía de forma contínua lo que ve.

   a) [0.5 pts] Diseña la tabla de comandos y direcciones que utilizaría el escáner.

   Respuesta.

   No hay una tabla dada, se espera una donde asignen direcciones de memoria para el registro de estado del escáner, el registro de comandos y el buffer de datos. Deberían tener cuidado con el vector de interrupciones respecto de las direcciones de memoria asignadas. Para el registro de comandos, deberían indicar solo un comando idle y un comando escanear, al igual que en estado un estado idle y un estado escaneando.

   b) [1 pt] Para el escáner diseñado, escribe una ISR que lo controle. Si haces supuestos, señálalos claramente.

   Respuesta.

   No se ahondará en detalle, se espera que hagan supuestos sobre el cómo pasar parámetros solamente y uso de la DMA para copias de memoria, en alto nivel debería tener un comando de inicialización y otro de escaneo que reciba como parámetro la dirección base donde dejar el contenido del buffer. Es correcto que en este paso cambien de un buffer por color a un buffer unificado con los 3 colores para cada pixel, pero deben detallarlo.

   Se debe vigilar que especifique de dónde adquieren los parámetros y cuales son, descontar si no lo hacen. Deben usar IRET o de lo contrario descontar 0.3 pts

   c) [1.5 pts] Escribe un programa que sea capaz de escribir en disco duro de un computador una imagen desde el escáner. El computador posee una DMA memoria a memoria, con cuatro registros: un registro de comandos, un registro de dirección base, un registro de dirección destino, y un registro de tamaño de copia; la DMA es controlada por la ISR 15h. Por su lado, el HDD tiene un buffer de datos de 2 MB y tres registros: un registro de comandos para saber si debe leer o escribir, un registro para dirección base de HDD, y un registro para tamaño de copia; el HDD es controlado por la ISR16h. En el HDD la imagen debe tener de forma consecutiva los valores de los colores rojo, verde y azul de cada pixel.

   Respuesta.

0.5 pts de descuento si uso no es consistente con lo hecho anteriormente pero se entiende. Se espera que llamen a los traps correspondientes en orden y esperen al DMA para escribir en HDD, luego esperen al HDD. Descontar 0.5 si no reordenan la data.

d) [3 pts] Diseña el microcontrolador del escáner mediante un diagrama y especifica en pseudocódigo qué debe hacer tu controlador. Sé explícito con tus supuestos e incluye los comentarios necesarios para que se entienda cómo funciona tu diseño.

Respuesta.

1.5 pts por diagrama Y/O explicación detallada. 1.5pts por pseudocódigo donde la idea es que usan un contador para ir de 0 a 256 y poner el digitalizador, y con eso listo (no hay descuento por asumirlo instantaneo) leen el digitalizador como si fuera un registro grande o mejor 3 registros (uno por color) y lo ponen en el registro de salida, al terminar marcan el registro de estado y esperan hasta que les llegue instrucción de digitalizar nuevamente. Otras soluciones que hagan sentido son válidas.

Se espera que en diagrama presenten algo parecido al computador básico.

Elige dos preguntas entre las preguntas 5 a 7:

5. Con respecto a la representación de números enteros:

a) Si en vez de bits (dígitos binarios, o en base 2), tuviéramos dígitos en base 4, ¿cuál es el rango de valores que podría ser representado usando *N* de estos dígitos? Considera que queremos representar tanto números positivos como negativos.

Respuesta.

$$[-\frac{4^n}{2}, \frac{4^n}{2} - 1]$$

Puntaje pregunta completa 2 puntos.
Por cada error se baja 0.2. Si tiene más de 4 errores o está todo malo se pone 0 puntos.

b) Si ahora tuviéramos *N* dígitos en base 3, ¿cuál es el rango de valores que podríamos representar? ¿Qué diferencia, además de las obvias, existe con el caso a)?

Respuesta.

$$[-\left\lfloor\frac{3^n}{2}\right\rfloor, \left\lfloor\frac{3^n}{2}\right\rfloor]$$

La principal diferencia es que en base 4, el primer digito me indica el signo ya que si es 0 o 1 entonces es positivo o 0 y si es 2 o 3 entonces es negativo. Mientras que acá debemos fijarnos en los primeros 2 dígitos para poder ver si es positivo o negativo.

1 Punto por el rango (se baja 0.2 por cada error en el rango, si tiene 4 errores o esta malo se pone 0).

1 Punto por la explicación.

c) Para cada una de las representaciones anteriores, a) y b), explica cómo se puede saber si un número es positivo o negativo a partir de la representación utilizada.

Respuesta.

Para saber si un número es negativo o positivo debemos primero fijarnos en la base.

- Si la base es par, si el primer digito es menor a $\left(\frac{b}{2}\right)$ entonces es positivo, en caso contrario negativo.
- Si la base es impar, debemos todos los que sean menores o iguales a $\left\lfloor\frac{b^n}{2}\right\rfloor$ son positivos, negativos en otro caso. Otra manera es sacar el digito central d que es $\left\lfloor\frac{b}{2}\right\rfloor$ y ver si el número es menor o igual al numero $dddd \ldots d$ que es el número conformado por n veces el digito central d. Si es menor o igual entonces es positivo, negativo en otro caso.

1 Punto por la explicación de los pares (o explicar en base 4).

1 Punto por la explicación de los impares (o explicar en base 3). No se baja si no puso el caso de 1 solo digito.

6. Describe, en función de los registros, unidades funcionales, señales de control y conexiones involucradas, la ejecución de la instrucción **`LOAD reg1, offset(reg2)`** (es decir, con direccionamientos por registro e indexado):

a) Describe, en orden, los 5 pasos principales en que se descompone la ejecución de la instrucción.

b) Dibuja el diagrama de los registros, unidades funcionales, señales de control y conexiones involucradas, de manera que quede clara la correspondencia con la descripción dada en a).

c) Explica qué diferencias principales habría si la instrucción fuera **`STORE reg1, offset(reg2)`**.

Respuestas.

a)

1) Se lee una instrucción desde la memoria de instrucciones, y se incrementa el PC (o *instruction pointer*).

2) Se lee el valor del registro 2.

3) La ALU calcula la suma del valor obtenbido del registro 2 más el *offset* (especificado en la propia instrucción).

4) El resultado de esta suma se usa como la dirección en la memoria de datos.

5) El dato así obtenido desde la memoria de datos se escribe en el registro 1.

b)

c) La unidad de control debería indicar que se trata de un *write* en lugar de un *read*; el valor del registro 1 sería el valor que se almacenaría en la memoria de datos; y la operación final de escribir en el registro 1 no ocurriría.

7. Responde:

a) ¿Por qué se prefiere usar complemento de 2 para representar números enteros negativos, en lugar de otras representaciones tales como complemento de 1 o bit de signo?

Respuesta.

Se prefiere el complemento de 2 porque es barato de implementar, además, tiene la propiedad de que se puede operar de forma normal sin consideraciones especiales, como es el caso de bit de signo.

b) ¿Cuál es el rol de la unidad de control del computador básico?

Respuesta.

La unidad de control es la encargada de orquestar el flujo de datos dentro de los componentes que conforman el computador con el fin de ejecutar las instrucciones que se le pasan. Incluye lo que es el decoding de las instrucciones desde opcodes a palabras de control completas. Además, en caso de que se tengan multiples modos de operación en el CPU es el encargado de lidiar con esa complejidad, por ejemplo, modos virtual y real de x86.

c) ¿Qué ventajas y desventajas tiene que la comunicación con los dispositivos de I/O sea *memory-mapped*, en lugar de usar el concepto de *ports* ?

Respuesta.

La ventaja es que no requiere introducir nuevo hardware para manejar las solicitudes de IO además del Address-Decoder y permite que las DMA trabajen sin modificaciones con ellos, la desventaja es que ocupa direcciones de memoria que ya no serán usables para los datos de los programas y eso es poco deseable. 0.5 ventaja/desventaja. Lo importante es que haga sentido.

d) ¿Cuáles son los principios —y en qué consisten en la práctica— en los que se basa el uso de memoria caché?

Respuesta.

Localidad espacial: Un programa tiende a utilizar memoria que está contigua.

Localidad temporal: Un programa que accede a una variable tiene una alta probabilidad de volver a accederla en el future.