PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2343 — Arquitectura de Computadores — 1/2018
Examen

Jueves 28 de Junio de 2018

El examen se compone de 6 preguntas. Debe responder cada una en una hoja separada, aunque puede utilizar más de una hoja para responder a una pregunta. Si no responde a una pregunta, debe entregar una hoja en blanco para ella. Recuerde poner su nombre en todas las hojas de respuesta.

Al momento de entregar, **evite doblar bordes o corchetear las hojas**, si no que sencillamente indique cuántas hojas son para esa pregunta y qué hoja es cada una en caso de utilizar más de una y procure que su hoja esté lo más lisa posible.

1. **Almacenamiento de datos, computador básico**

   1.1) A continuación, se presenta un diagrama de una memoria `RAM` para el computador básico:

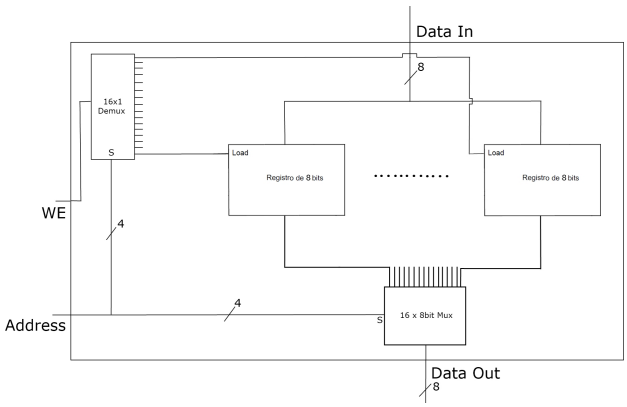

Figura 1: Representación de la memoria `RAM`.

A partir de este diagrama, ¿cómo modificaría esta componente para poder habilitar dos accesos (tanto para lectura como escritura) de forma simultánea? Puede hacer un diagrama o explicar en detalle su implementación. **[1 pt]**

Respuesta

Básicamente, doblando los siguientes buses de datos: *Data In*, `WE`, `Address` y `Data Out`. Esto puede ser haciendo que estos buses sean del doble de tamaño (dividiéndolos dentro del componente del diagrama) o bien que sean dos buses distintos. También se necesitaría una unidad adicional de *Demux* y una de *Mux*. En estos **no pueden** aumentar sus dimensiones para lograr lo pedido, dado que de todas formas solo pueden seleccionar un registro con cada componente. Por último, el bus de salida de cada registro debe tener dos cables, uno a cada uno de los *Mux* de salida.

1.2) Haciendo uso del componente anterior, modifique el computador básico para que pueda implementar las siguientes instrucciones:

- `DREAD (dir1),(dir2)`: Se almacena la palabra ubicada en `dir1` dentro del registro `A` y la palabra ubicada en `dir2` dentro del registro `B`.
- `DWRITE (dir1),(dir2)`: Se almacena en la dirección `dir1` el valor contenido en el registro `A` y en la dirección `dir2` el valor contenido en el registro `B`.
- `RW (dir1),(dir2)`: Se almacena la palabra ubicada en `dir1` dentro del registro `A` y en la dirección `dir2` el valor contenido en el registro `B`.
- `WR (dir1),(dir2)`: Se almacena en la dirección `dir1` el valor contenido en el registro `A` y se almacena la palabra ubicada en `dir2` dentro del registro `B`.

Si su implementación genera algún cambio con respecto al funcionamiento del computador básico, debe indicarlo y explicar qué se debe hacer para que no cambie la ejecución del resto de las instrucciones. Además, para cada instrucción, debe indicar las señales a activar (sean del diagrama original o las añadidas por usted) para su ejecución correcta. Puede realizar un diagrama o explicar en detalle su implementación.
Nota: Puede asumir que no se ejecutarán programas donde `dir1 = dir2`. **[4 pts]**
Respuesta:
Lo más importante para que puedan responder esta pregunta es que den cuenta de la necesidad de modificar la `ROM` para poder obtener más de un literal. Existen muchas formas válidas de hacer esto:

- Pueden ampliar la memoria `ROM` y que cada instrucción tenga 2 literales.
- Pueden segmentar en 2 los bits utilizados originalmente para el literal y separarlos en 2 buses (aunque esto conlleva a un menor poder de representación, algo que deberían comentar).
- Pueden hacer lo mismo del punto 1 pero sin ampliar la `ROM`, indicando que el máximo de instrucciones por programa disminuye.

En cualquiera de estos casos, no obstante, se espera que mencionen el impacto de que ahora en todas las instrucciones habrán dos literales. Basta con que digan que el segundo es igual a cero para el resto de las instrucciones donde no se ocupa. A partir de estos dos literales pueden comenzar a trabajar en el diagrama. Por ejemplo, pueden incluir ahora dos *Mux Address*, uno para cada dirección de ingreso de la `RAM` (lo que podría ampliar, además, las funcionalidades del computador). No obstante, también pueden mantener el mismo *Mux Address* y hacer que el literal `dir2` vaya directamente a la segunda entrada de dirección de la `RAM`. En ambos casos, deben mencionar cómo cambian los buses de selección.
En lo que respecta a las señales de control de la `RAM`, va a depender de cómo interpreten la pregunta anterior (la hayan respondido o no). Si usan dos buses, deben indicar el impacto que genera esto para las instrucciones de lectura normales (`MOV (dir),Reg`; `MOV Reg,(dir)`), donde ahora se debe deshabilitar la segunda señal `WE` añadida. Si usan un único bus del doble de tamaño, deben indicar un estándar de forma que se siga modificando un solo registro o una sola palabra en el ejemplo antes mencionado.
A partir de las modificaciones realizadas, se espera que comenten además que el bus de señales proveniente de la unidad de control aumenta (salvo que su implementación funcione y no incluya la adición de nuevas señales de selección o habilitación). Finalmente, se presenta la tabla a partir de la cual se evidencia el funcionamiento correcto de las instrucciones (se espera la particularidad de que en `RW` y `WR` no estén habilitadas las dos señales `WE` implementadas). No es necesario indicar un *opcode*.
Importante: El análisis anterior supone la ejecución de las instrucciones en un solo ciclo. Esto **no está explicitado** en el enunciado, por lo que es válido implementarlas con dos ciclos (lo que implicaría no modificar la `ROM`) tomando en cuenta lo siguiente:

- Se debe ocupar la nueva memoria `RAM`, ya que está explicitado en el enunciado.
- Si implementan las instrucciones en dos ciclos, se espera que añadan un registro adicional que mantenga el primer literal de la operación intacto para la siguiente ejecución.
- A raíz de lo anterior, deben indicar la combinación de señalas activadas para cada uno de los dos ciclos de las instrucciones.

1.3) Asumiendo que ya posee el computador básico de la implementación anterior, ¿qué otra instrucción podría incluir? No es necesario que mencione las señales involucradas para implementarla o el *opcode* asociado, pero sí que justifique por qué se podría implementar. **[1 pt]**

Respuesta:

Aquí existen varias respuestas posibles, pero se espera que mencionen solo una que tenga sentido en el contexto del problema. Si bien no son las únicas, a continuación se presentan algunas opciones válidas:

- `ADD B,Lit` (Conectando el segundo literal al *Mux A*, si implementan todo con un ciclo).
- `Sop B,(A) | Sop A,(A) | Sop (A),A` (Si se considera la implementación de un segundo *Mux Address*, donde en vez de recibir el bus del registro `B` recibe el del registro `A`).
- `Sop (Dir1),(Dir2)` (Si se conecta la salida de la segunda dirección de la `RAM` al *Mux A* de forma que se pueda realizar esta operación).

Donde `Sop` $\in$ {`ADD, SUB, AND, OR, XOR`}.

2. **Input/Output**

2.1) Sea un dispositivo I/O cuya controladora tiene el siguiente diagrama:

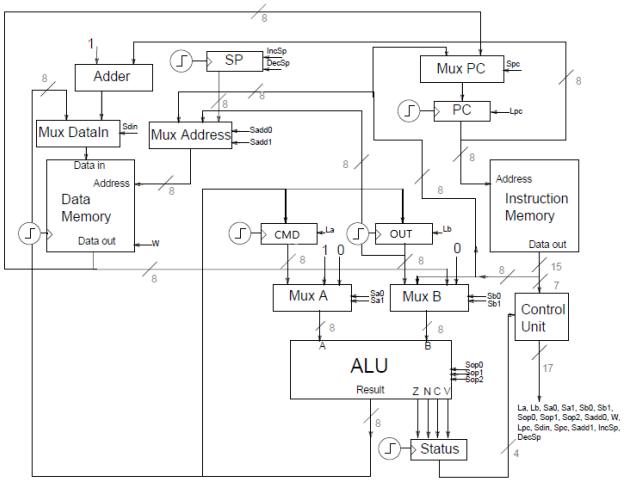

Figura 2: Diagrama de la controladora del dispositivo I/O.

Donde el registro `CMD` puede ser leído y escrito por el computador para entregar un comando al I/O y, además, el registro `OUT` es leído por el computador. En la dirección de memoria 0 siempre se encuentra el estado actual del botón que tiene este I/O. Este dispositivo se conecta vía *ports* a un computador compatible con x86-286. Además, se tienen las siguientes tablas de *ports*, comandos y estado del dispositivo:

Port	**Dispositivo**
0	`DMA` *ports* a memoria
1	Comandos `HDD`
2	Estado `HDD`
3-259	*Buffer* `HDD`
260	`CMD` botón
261	*Buffer* botón

Cuadro 1: *Mapeo* de puertos.

Comando	**Acción**
0	No hacer nada
1	Copiar estado del botón en `OUT`
3	Reiniciar controlador

Cuadro 2: Tabla de estados del I/O.

Estado	**Explicación**
0	Botón no presionado
1	Botón presionado
2	Copiando información

Cuadro 3: Tabla de estados del I/O.

Suponiendo que la controladora de este dispositivo es compatible con la `ISA` del computador básico, escriba el programa que controla este dispositivo. **[3 pts]**

Respuesta

```
DATA:
CODE:
        JMP reset  ; Al prender el IO, se resetea
    entry:
        CMP CMD,1
        JEQ copy
        CMP CMD,2
        JEQ reset
        JMP entry
    copy:
        MOV OUT,2
        MOV OUT,(0)
        MOV CMD,0 ; Esto es opcional
        JMP entry
    reset:
        MOV CMD,0
        MOV OUT,0
        JMP entry
```

Lo que se preguntó es el programa que ejecuta la controladora del dispositivo IO, que es básicamente el computador básico. No se considera puntaje por hacer una `ISR`. Descuento de 1 punto por usar registros `A` y `B` en lugar de `CMD` y `OUT`

2.2) Qué diferencia el esquema de *ports* del esquema de *memory-mapping*? **[1 pt]**
Respuesta
Básicamente *ports* crea un espacio de direccionamiento separado y exclusivo para IO, mientras que en *memory-mapping* potencialmente se quita espacio direccionable de la `RAM`.

2.3) ¿Por qué se incluye una unidad `DMA` en los computadores? **[1 pt]**
Respuesta
De modo tal que la `CPU` pueda realizar otras tareas importantes mientras se completa una copia de memoria.

2.4) Qué diferencia los esquemas de *polling* e interrupciones? **[1 pt]**
Respuesta
En *polling* se pregunta continuamente por el estado del dispositivo, usando ciclos de `CPU` para ello. En interrupciones, el dispositivo IO avisa a la `CPU` cuando tiene algo que necesita atención y en el intertanto la `CPU` puede ejecutar programas sin preocupaciones.

3. ***Caché*, memoria virtual**

3.1) Dibuje un diagrama del proceso de acceso a una dirección de memoria virtual en un computador que además tiene *caché*. Explique qué piezas vistas en el curso intervienen en cada paso e identifique el mejor y peor caso. **[3 pts]**
Respuesta

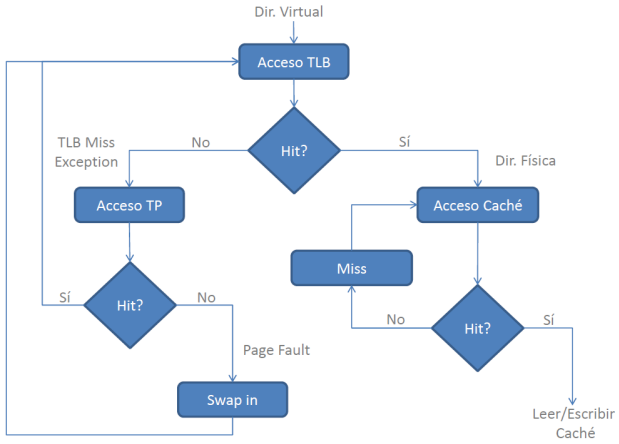

Figura 3: Diagrama de acceso completo, obtenido de las clases.

En el acceso a la `TLB` intervienen la `MMU` y la `TLB`.
En el acceso a la Tabla de Páginas intervienen la `MMU` y la `RAM`, en ambas ramas de decisión también la `TLB` ya que debe ser actualizada. En caso de *Page Fault* participa además el disco duro (o almacenamiento secundario en términos generales).
En el acceso a *caché* interviene la `MMU` y la *caché*, en caso de *miss* interviene además la `RAM`.
El mejor caso es cuando la página se encuentra en la `TLB` y en *caché*, mientras que el peor caso es cuando la página se encuentra en disco y su entrada no está en la `TLB`.

3.2) ¿Por qué se utiliza la memoria virtual? **[1 pt]**
Respuesta
Para aislar los datos de cada programa e impedir que interfieran unos con otros, además, le permite a cada programa utilizar todo el espacio de memoria direccionable si lo requiere.

3.3) En un computador *Von Neumann* donde las instrucciones y los datos de un programa están en el mismo espacio de memoria, suponiendo que se tiene memoria virtual, ¿qué se podría hacer para que las páginas de datos nunca sean interpretadas como instrucciones por el computador? **[2 pts]**
Respuesta
Se puede agregar un bit `NX` a cada página, por ejemplo en la tabla de páginas, que indique si las direcciones contenidas en ella se pueden ejecutar o no. Si el bit `NX` está en 1, cualquier intento de ejecutar su código debiera ignorarse y/o activar un *trap*.

4. **Paralelismo a nivel de instrucción**

4.1) Considere la arquitectura del computador básico con *pipeline*. Una de las grandes desventajas que posee es el hecho de que si la unidad predictora de saltos se equivoca, se pierden tres ciclos, lo que impacta considerablemente el tiempo de ejecución de los programas. ¿Cómo modificaría esta arquitectura para poder perder un ciclo menos por predicción de salto? Puede hacer un diagrama o detallar su implementación. **[3 pts]**

Respuesta

Bastaría con la adición de un sustractor que tenga de entradas los valores de los registros `A` y `B` dentro de la etapa `ID`, de forma que en la etapa `EX` la unidad predictora de saltos pueda determinar si es necesario hacer un *flush* o no, desechando dos ciclos y no tres. Notar que para ello tanto la unidad de salto como la unidad de *hazard* de control se deben ubicar ahora en la etapa `EX`, lo que hace innecesario el uso de la *flag* `Z` de la `ALU`.

4.2) Suponga que tiene una empresa que se encuentra desarrollando un dispositivo relativamente simple, que no ejecuta programas de más de 10 instrucciones. Uno de los desarrolladores sugiere que en pos de la eficiencia lo diseñen con un *pipeline* de 10 etapas para poder paralelizar las ejecuciones. ¿Es esta una decisión conveniente tomando en cuenta el tiempo de ejecución de los programas? Justifique su respuesta. **[2 pts]**

Respuesta

No. Al existir el proceso de *filling* (llenado de las etapas del *pipeline*), recién en el décimo ciclo todas las instrucciones estarían ejecutándose en una etapa del *pipeline*, tomando un total de 19 ciclos de ejecución. Si se tuviera una arquitectura como la del computador simple, el programa tendría una ejecución de 10 ciclos, además de tener una menor limitante en lo que respecta a las instrucciones. En resumen, la decisión perjudicaría el performance del dispositivo. **Importante:** Se asume que la diferencia del tiempo de ejecución de un ciclo en el dispositivo con y sin *pipeline* no es significativa, validando el análisis anterior.

4.3) Considere un computador `RISC`-*Harvard* con un *pipeline* de 20 etapas, donde la unidad de salto se activa en la etapa 10 (20 % de las veces) o en la etapa 19 (80 % de las veces), dependiendo del origen de los parámetros necesarios para resolver el salto. Dado que la unidad de predicción de saltos de este computador acierta en el 50 % de las oportunidades, en promedio, ¿cuántos ciclos por salto pierde este computador? Aproxime al entero mayor. **[1 pt]**

Respuesta

Primero, vemos que en promedio por salto fallido se pierde esta cantidad de ciclos:

$$9 * 0{,}2 + 18 * 0{,}8 = 16{,}2$$

Ahora, la unidad predictora de saltos se equivoca en la mitad de las predicciones. Por lo tanto, en promedio se pierden:

$$0{,}5 * (9 * 0{,}2 + 18 * 0{,}8) = 8{,}1$$

La respuesta, entonces, es 9.

5. **Paralelismo Avanzado**

5.1) ¿Qué diferencia los esquemas UMA y NUMA? **[1 pt]**
Respuesta
En UMA, todos los procesadores comparten el mismo bus de memoria y por ende todos compiten por acceso a la RAM. En NUMA, grupos de procesadores tienen memoria local y existe una conexión para acceder a memoria de otros grupos, con una latencia mayor. Básicamente es UMA particionado.

5.2) ¿Qué desafío plantean los programas paralelos a los computadores multi-núcleo con *caché*? **[1 pt]**
Respuesta
En que se debe mantener la consistencia en caso de que más de un núcleo acceda a las mismas direcciones de memoria e intente hacer modificaciones.

5.3) El protocolo MOESI corresponde a una variante del protocolo MESI estudiado en clases. Este posee el mismo comportamiento, salvo por la existencia de un nuevo estado O ("*Owned*"). Este es utilizado para indicar que uno y solo uno de los controladores de *caché* tiene el permiso para modificar el valor de una línea de la *caché* asociada a un bloque de la memoria y compartirlo con el resto de las cachés que la posean. Esto permite que, en caso de que se hagan lecturas de un recurso compartido, la transferencia de datos se haga entre *cachés* y, además, no es necesario que se escriban los cambios directamente en la memoria principal, solo cuando es estrictamente necesario.

a) ¿Cómo se podría asegurar que sea una sola *caché* la que posea el estado O para un recurso compartido? Comente considerando el procedimiento del protocolo MESI para recursos compartidos (en particular, para los estados M, S e I). **[2 pts]**
Respuesta
Cuando un recurso se comparte por primera vez se asigna el estado E para indicar que es exclusivo. Luego, cuando otro procesador solicita su lectura, tanto para dicho procesador como para el primero el estado se convierte en S, pues está compartido. Posteriormente, si una de las dos *caché* realiza una modificación, la *caché* que realiza la modificación cambia la línea a estado M y la otra invalida su copia. En el protocolo MOESI se puede establecer, en cambio, que la *caché* que escribe cambie a estado O en vez de M para indicar que será el encargado predeterminado para compartir el recurso entre las *cachés*, siempre que no existan solicitudes de escritura que hagan obsoleta su copia, mientras que la segunda mantiene su estado S, con la salvedad de que se le transfiere el nuevo valor de la línea. Naturalmente, el resto de los procesadores que soliciten la lectura del recurso para almacenarlo en sus *cachés* mantendrán un estado S, efectuando la transferencia desde la *caché* con la línea en estado O.

b) Suponga que un par de *cachés* comparten una línea con estados O y S, respectivamente. Si la segunda *caché*, en estado S, solicita realizar cambios en la línea compartida, ¿qué protocolo se puede seguir para mantener la consistencia? **[2 pts]**
Respuesta
Se puede manejar de dos formas, pero en ambas la *caché* que solicita la escritura pasa a estado O. En la primera forma, la *caché* que estaba encargada de la línea compartida puede simplemente invalidar su copia para no causar problemas de inconsistencia. En la segunda, en cambio, simplemente puede cambiar su estado a S y recibir a través de una transferencia el nuevo valor.

6. Considera la siguiente situación hipotética. Tenemos un puente para cruce de autos, pero de una sola pista. Por lo tanto, los autos que vienen del norte y los que vienen del sur tienen que coordinarse para poder cruzarlo. Por supuesto, múltiples autos que vienen del norte pueden cruzar el puente al mismo tiempo (uno de trás de otro), y lo mismo para múltiples autos que vienen del sur. Supongamos que representamos los autos por procesos; entonces tenemos dos tipos de procesos: los procesos `N[1 ... n]` y los procesos `S[1 ... m]`. Para facilitar la coordinación, definimos un (único) proceso coordinador `C`, que puede comunicarse con todos los procesos `N` y todos los procesos `S`, de la siguiente manera: cuando un proceso `N[j]` o `S[k]` se aproxima al puente, envía una solicitud para cruzar el puente al proceso `C`; cuando el proceso `C` recibe una solicitud, toma la decisión de si aceptarla o no según una cierta regla; si la acepta, entonces contesta la solicitud; de lo contrario, deja pendiente la solicitud. La regla dice que si no hay autos cruzando el puente, entonces la solcitud es aceptada; también dice que si hay autos del mismo tipo (que el solicitante) cruzando el puente, entonces la solicitud es aceptada si el número de autos de ese tipo que han cruzado el puente consecutivamente es menor que `q`; finalmente, si `q` autos de un mismo tipo han cruzado el puente consecutivamente, entonces se cambia la dirección del puente.

   6.1) Escribe el código correspondiente a un proceso auto. **[1.5 pts]**

   Respuesta

   El proceso `C` tiene que saber cuál auto está haciendo la solicitud (para poder responder) y si viene del norte o del sur. Una posibilidad es que el canal de solicitud incluya ambos datos:

```
type dir = {N, S}
channel req(dir, id)
```

   Otra es que haya canales separados para los autos del norte y los del sur:

```
channel reqN(id), reqS(id)
```

   Supongamos esta última. Suponemos, además, que cada auto tiene un identificador distinto y, por lo tanto, un canal distinto, `go[ ]`, para recibir las respuestas. Entonces un auto del norte (para uno del sur es análogo) ejecuta simplemente:

```
proc N[i]:
    send reqN(i)
    receive go[i]
```

6.2) Escribe el código correspondiente al proceso `C` **[4.5 pts]**; usa paso de mensajes asincrónico (operaciones *send*, *receive* y *empty*) para la comunicación entre los procesos.

Respuesta

El proceso `C` ejecuta un *loop*. Primero atiende a los autos que vienen del norte, y luego a los del sur (arbitrariamente; puede ser al revés). Se atiende hasta un máximo de q autos consecutivos, o bien hasta que no haya más autos solicitando pasar el puente. Como los canales son en la práctica colas de mensajes, no es necesario que `C` maneje alguna estructura de datos adicional para registrar los autos que vienen del sur que están solicitando pasar el puente:

```
proc C:
        while true:
                k = 0
                while k < q and !empty(reqN):
                        receive reqN(autoId)
                        send go[autoId]()
                        k = k+1
                k = 0
                while k < q and !empty(reqS):
                        receive reqS(autoId)
                        send go[autoId]()
                        k = k+1
```

Nota de corrección: Se permite el uso de pseudocódigo.