Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación


**IIC2343 – Arquitectura de Computadores (II/2015)**

# Interrogación 2

## Pregunta 1

a) Compare las arquitecturas Harvard y Von Neumann desde el punto de vista del tiempo de ejecución de las instrucciones. Fundamente y explique claramente las diferencias. **(1 pto.)**
**Solución:** Al necesitar cómo mínimo dos accesos a memoria no simultáneos para ejecutar una instrucción arbitraria, la arquitectura Von Neumann es en teoria más lenta que la Harvard, que necesita sólo un ciclo para hacer lo mismo.

b) ¿Cuántas pasadas por el código fuente debe hacer como mínimo un assembler para x86? ¿Cuántas debe hacer el assembler del computador básico? **(1 pto.)**
**Solución:** Al no conocer a priori el lugar donde están las declaraciones de variables, el assembler para x86 necesitará como mínimo 2 pasadas: la primera para almacenar los nombres y ubicación de las variables y la segunda para generar el código. Por otro lado, el assembly del computador básico sólo necesita una, ya que las variables sólo se declaran en el segmento DATA.

c) ¿Cómo favorece a los dispositivos móviles la ley de Moore? **(1 pto.)**
**Solución:** Al mejorar la miniaturización, es posible construir procesadores más potentes, que utilizan menos espacio y consumen menos energía. Esto es exactamente lo que necesita un procesador movil.

d) Describa el procedimiento que realiza un disco duro para acceder a un dato. **(1 pto.)**
**Solución:** Para acceder a un dato, el disco duro gira el disco magnético y posiciona el cabezal en el radio adecuado. Luego, al pasar sobre la ubicación del dato, el cabezal lee la información.

e) ¿Qué beneficios podría traer el uso de procesador RISC para servidores? **(1 pto.)**
**Solución:** Al ser procesadores más simples, su uso de energía es menor. Esto resulta muy beneficioso para servidores que no realicen tareas muy complejas, ya que se podría tener muchos de estos, por una fracción del consumo energético de un grupo de servidores tradicionales.

f) Explique cómo funciona la transferencia de direcciones y datos desde/hacia los dispositivos mapeados a memoria. **(1 pto.)**
**Solución:** Las direcciones son redireccionadas mediante un *Address Decoder*, que decide si corresponde a un dispositivo de I/O o a la memoria. Luego, esta dirección llega al dispositivo correcto, que recibe/envía el dato por el bus de datos.

## Pregunta 2

a) Modifique el computador básico para que tenga soporte para las instrucciones IN y OUT. Describa los cambios a la microarquitectura y a la ISA. **(2 ptos.)**

**Solución:** Los cambios a realizar son los siguiente, asumiendo que los dispositivos pueden conectarse entre los puertos 1-255:

- Agregar una señal de control a la salida de la unidad de control. Esta señal será 1 sólo cuando la instrucción sea IN o OUT, el resto del tiempo será 0.
- Para OUT: Crear un nuevo bus de 8 bits, que sale del registro A, con dirección a los dispositivos de I/O. Además, sacar una copia del bus de literales, en dirección a los dispositivos de I/O. El bus que viene de A será el bus de datos de entrada a los dispositivos, mientras que el que sale del bus de literales será el de direcciones. Finalmente, este último bus debe estar controlado por un enabler, que se activa mediante la nueva señal de control agregada.
- Para IN: Crear un nuevo bus de 8 bits, que viene desde los dispositivos hasta el Mux A, aprovechando la entrada que no está ocupada. Además, para indicar el puerto se utiliza el bus de direcciones definido en el item anterior.

b) Conecte al computador modificado en el ítem anterior, un dispositivo de I/O que utilice los puertos 13 (comandos), 14 (estado) y 15 (datos). Diseñe las conexiones entre los buses de datos y direcciones y los registros de comandos, estado y datos del dispositivo. **(2 ptos.)**
**Solución:** Se asume que los registros de estado y datos son de sólo lectura, mientras que el de comandos es de sólo escritura.

- Para el registro de comandos, conectar la señal Load de este a la salida de un circuito lógico que compara el valor en el bus de direcciones con el número 13, entregando 1 si es igual y 0 en otro caso. El bus de datos de salida a los dispositivos de I/O se conecta a la entrada del registro de comandos.
- Para el registro de estado y datos, conectar la salida de estos a un mux de 4 entradas, donde sólo 3 se usan: registro de estado, registro de datos, literal 0. El control del mux se conecta con un circuito lógico que retorna 0 si el bus de direcciones contiene un 14 (estado), 1 si contiene un 15 (datos) y 2 en cualquier caso (literal 0). La salida del mux se conecta con el bus de salida de los dispositivos de I/O (definido anteriormente), que va al Mux A.

c) Asuma que el dispositivo de I/O conectado en el ítem anterior es una tarjeta de red. Escriba un programa, usando el assembly del computador básico extendido con IN y OUT, que lea los datos recibidos por la tarjeta y los escriba a partir de la dirección de memoria 0x40. Defina claramente los comandos y estados de la tarjeta que utilizará. **(2 ptos.)**
**Solución:** Se define el comanda que inicia el proceso de recepción como 0, el estado de dato recibido como 0.

```
DATA
CODE
    MOV   B, 0x40
    MOV   A, 0
    OUT   13, A
  for:
    IN    14, A
    CMP   A, 0
    JNE   for
    IN    15, A
    MOV   (B), A
    INC B
    JMP for
```

## Pregunta 3

Implemente en assembly x86 el algoritmo para ordenar arreglos de enteros *min-max sort*. Este algoritmo opera buscando el máximo y el mínimo de un arreglo y colocándolos en los extremos correspondientes, para luego repetir el proceso con el subarreglo que no contiene a los extremos. En caso de usar subrutinas, utilice la convención *stdcall*. Cualquier cosa que asuma debe quedar claramente explicada. **(6 ptos.)**

**Solución:** Se asume que el arreglo a ordenar es de números enteros de 8 bits.

```
  JMP   main
  N     dw   ?
  A     db   ?,?,?,...
main:
  PUSH  N
  LEA   BX, A
  PUSH  BX
  CALL  min_max_sort
  RET
min_max_sort
  PUSH  BP
  MOV   BP, SP
  MOV   BX, [BP+4]
  MOV   CX, [BP+6]
  MOV   DI, 0 ; indice del maximo
  MOV   AX, 0 ; indice del minimo
  MOV   DH, -128 ; valor maximo
  MOV   DL, 127 ; valor minimo
  MOV   SI, 0 ; contador
for:
  CMP   SI, CX
  JEQ   end_for
  CMP   [BX+SI], DL
  JLT   new_min
  CMP   [BX+SI], DH
  JGT   new_max
  JMP   next
new_min:
  MOV   DL, [BX+SI]
  MOV   AX, SI
  JMP   next
new_max:
  MOV   DH, [BX+SI]
  MOV   DI, SI
next
  ADD   SI, 1
  JMP   for
end_for:
  MOV   SI, AX
  MOV   AL, [BX]
  MOV   [BX], DL
  MOV   [BX+SI], AL
  MOV   SI, CX
  SUB   SI, 1
  MOV   AL, [BX+SI]
  MOV   [BX+SI], DH
  MOV   [BX+DI], AL
```

```
  CMP   CX, 2
  JEQ   return
  CMP   CX, 3
  JEQ   return
  SUB   CX, 2
  PUSH  CX
  ADD   BX, 1
  PUSH  BX
  CALL min_max_sort
return:
  POP BP
  RET   4
```