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

IIC2343 – Arquitectura de Computadores

Guía 3 – Repaso Examen I

Profesor: Yadran Francisco Eterovic Solano

Ayudante: Germán Leandro Contreras Sagredo (glcontreras@uc.cl)

Temas a tratar

Los temas a tratar dentro de esta guía son:

- Programabilidad, saltos y subrutinas.
- Arquitectura x86.
- Comunicación de `CPU` y Memoria con I/O.

Preguntas

1. a. **(Examen - II/2014)** Una máquina de *stack* es un computador que utiliza un *stack* en vez de registros para almacenar los resultados de las operaciones. Esto significa que cada instrucción aritmética o lógica de dos parámetros, toma los dos valores en el tope del *stack* y luego los elimina, sustituyéndolos por el valor de la operación recién realizada. Para el caso de las operaciones de un parámetro, por ejemplo `NOT`, el computador solo sustituye el valor en el tope del *stack* por el nuevo valor. Además, una máquina de *stack* es capaz de cargar valores literales en el tope del *stack* y también descartarlos.
    i. Diagrame la microarquitectura de una máquina de *stack* de 8 bits. Este computador debe ser capaz de realizar las mismas operaciones aritméticas y lógicas que el computador básico. No es necesario tener soporte para saltos.
    ii. Describa una `ISA` para la máquina del ítem anterior. Indique *opcodes*, señales de control y las instrucciones del *assembly* correspondientes.

   b. **(I1 - I/2016)** Una máquina `RAM` es un tipo de computador en el cual se utiliza solo la memoria `RAM` de datos para almacenar los resultados de las operaciones aritméticas y lógicas. Una máquina `RAM` puede tener más registros para uso interno, pero para un programador, solo se encuentran expuestos la memoria y los literales para realizar las operaciones. Por ejemplo, `ADD A,B` es sustituida por `ADD (Dir1),(Dir2)`. Construya una máquina `RAM` que posea las mismas funcionalidades que el computador básico. Especifique detalladamente el *hardware* y el formato de las instrucciones (señales de control, *opcodes*, *assembly*).

c. **(I1 - II/2016)** En esta pregunta deberá diseñar un computador especializado en el manejo de matrices. El computador debe ser capaz de: i) copiar una matriz desde la memoria de datos a un registro y viceversa, ii) sumar 2 matrices almacenadas en registros distintos y almacenar el resultado en un registro.

   I. Haga el diagrama del computador, considerando que las matrices pueden tener como máximo $N \times N$ elementos, cada uno de 1 byte.

   II. Diseñe el *assembly* del computador. Cada instrucción debe estar asociada a un *opcode* y estos a sus respectivas señales de control.

   III. Agregue tanto al *hardware* como al *assembly* soporte para una instrucción que permita modificar el valor de un elemento arbitrario de una matriz almacenada en la memoria de datos.

d. **(Examen - II/2012)** ¿Qué implicancia tiene en el tamaño de los programas el eliminar la conexión entre memoria de datos y `PC` (*program counter*) en el computador básico?

e. **(I1 - II/2015)** Modifique el diagrama del computador básico (sin saltos y subrutinas) de manera que soporte la ejecución de la instrucción `GOTO dir`, que fuerza que la siguiente instrucción en ejecutarse sea la ubicada en la dirección `dir`.

f. **(I2 - I/2015)** El soporte para subrutinas del computador básico tiene la limitación de que la instrucción `CALL` siempre tiene que llamarse con un *label* o un literal como parámetro. Esto significa que no es posible, por ejemplo, llamar a una subrutina mediante un número previamente calculado y almacenado en un registro, `CALL A` (una especie de direccionamiento indirecto de subrutinas). Describa cómo es posible saltarse esta limitación, *i.e.* permitir el llamado de subrutinas indicando su dirección mediante un número previamente calculado, usando el *assembly* del computador básico y sin modificar la arquitectura de ninguna manera.

g. **(I1 - I/2018)** Modifique el *hardware* del computador básico para que las instrucciones `RET` y `POP` tomen un solo ciclo.

2. a. **(I2 - II/2014)** Describa una convención de llamada para x86, que sea más rápida que *stdcall* al momento de leer y escribir parámetros y valores de retorno. Contrapese las posibles ventajas y desventajas.

   b. **(I2 - II/2014)** ¿Es posible emular el funcionamiento del registro `BP` en el computador básico, sin modificar la arquitectura? Si su respuesta es positiva, esboce la solución. Si es negativa, explique el motivo.

   c. **(Apuntes - Arquitectura x86)** El siguiente código en *Assembly* x86 obtiene una potencia mediante subrutinas.

```
MOV BL,exp
MOV CL,base
PUSH BX ; Paso de parametros (de derecha a izquierda)
PUSH CX
CALL potencia ; potencia (base,exp)
MOV pow,AL ; Retorno viene en AX
RET
potencia: ; Subrutina para el calculo de la potencia
    PUSH BP
    MOV BP,SP ; Actualizamos BP con valor del SP
    MOV CL,[BP + 4] ; Recuperamos los dos parametros
    MOV BL,[BP + 6]
    MOV AX,1 ;AX = 1
start:
    CMP BL,0 ; if exp <= 0 goto endpotencia
    JLE endpotencia
    MUL CL ; AX = AL * base
    DEC BL ; exp --
    JMP start
endpotencia:
    POP BP
    RET 4 ; Retornar , desplazando el SP en 4 bytes
base db 2
exp db 2
pow db 0
```

      I. Describa cómo se ejecuta este programa.

      II. ¿Cómo se modificaría este programa para hacer uso de variables locales? ¿Cómo cambia la dinámica desde el punto de vista de los registros `BP` y `SP`?

   d. **(I2 - II/2014)** Implemente, usando el *assembly* x86 y la convención *stdcall*, un programa que calcule el máximo común divisor de dos enteros no negativos, donde ambos no pueden ser 0 simultaneamente, utilizando el algoritmo de Euclides, descrito a continuación:

$$\text{Para } a, b \geq 0,\ mcd(a,b) = \begin{cases} a & \text{, si } b = 0 \\ mcd(b, a \bmod b) & \text{, en cualquier otro caso.} \end{cases}$$

3. a. **(I2 - II/2015)** Explique cómo funciona la transferencia de direcciones y datos desde/hacia los dispositivos mapeados a memoria.

   b. Describa las instrucciones de la `ISA` de un computador x86 que permiten acceder a dispositivos mediante *port* I/O.

   c. **(I2 - I/2017)** ¿Con qué tipo de dispositivo de I/O es preferible utilizar mapeo de memoria por sobre puertos?

   d. ¿Por qué es mejor hacer uso de interrupciones en vez de *polling*? Mencione un ejemplo.

   e. **(I2 - I/2016)** ¿Cuál es la función del vector de interrupciones? ¿Cuál es su contenido?

   f. **(I2 - II/2016)** Luego de recibir la señal `INTA`, ¿qué tarea(s) debe realizar un controlador de interrupciones?

   g. Detalle, paso a paso, cómo se manejaría una interrupción realizada por un dispositivo (llamémoslo IO_i) que se encuentra conectado a otro dispositivo (llamémoslo IO_j), donde la ISR de este último se encuentra almacenada en el vector de interrupciones del computador.

   h. Explique la diferencia entre las interrupciones realizadas por *hardware* y *software*, dando un ejemplo de cada una.

   i. **(I2 - II/2016)** Para los siguientes ejercicios, considere la siguiente tabla, que presenta el vector de interrupciones completo de un computador con `ISA` x86 de 16 bits. El vector de interrupciones se encuentra almacenado a partir de la dirección de memoria `0x0000`:

IRQ	**Dispositivo**	**Pos. en vector**
IRQ0	*Timer* del sistema	00
IRQ1	Disco Duro	01
IRQ2	Interfaz `USB`	02
IRQ3	Interrupción *software*	03

Cuadro 1: Vector de interrupciones del computador.

   I. ¿Cuantos dispositivos que generen solicitudes de interrupción pueden conectarse?

   II. Dos dispositivos, teclado y *mouse*, están conectados a la interfaz USB. Describa un mecanismo para ejecutar la `ISR` correspondiente al mouse, cuando este genera una interrupción.

   III. ¿Que ocurriría en este computador si se ejecuta la instrucción `MOV [0],AX`?

   IV. Proponga un esquema para permitir el acceso (lectura y escritura) controlado y centralizado al vector de interrupciones por parte de los programas, *i.e.*, el acceso solo puede realizarse a través de una interfaz entregada por el sistema operativo (o la `BIOS`).
   Hint: El esquema puede incluir cambios a la arquitectura del computador.

4. **(I3 - II/2012)** Suponga que se tiene un dispositivo de adquisición de imágenes térmicas conectado a un computador que tiene una microarquitectura especializada para la adquisición de imágenes, pero con `ISA` compatible con x86 de 16 bits. El computador tiene una memoria principal de 64 kilobytes, con el siguiente mapa de memoria para los primeros 4096 bytes:

Dirección	**Función asociada**
0-5	*Exception handlers*
6	Registro de comandos de la cámara
7	Registro de estado de la cámara
8-14	Vectores de interrupciones de *hardware*
15	Vector de interrupción de escritura en disco
16	Vector de interrupción de adquisición de imagen
17-31	Vectores de interrupciones de *software* de uso libre
32-123	Memoria de uso libre
124-1023	*Buffer* de adquisición de la cámara.
1024-4096	Espacio de memoria del disco.

Cuadro 2: Tabla que muestra el mapa de memoria del dispositivo.

Se desea escribir un programa que permita adquirir imágenes mediante la cámara y luego almacenarlas en disco. Las imágenes generadas por la cámara se encuentran en escala de grises de 8 bits, ordenadas por filas en una matriz cuadrada de 30x30.

a. Escriba una `ISR` para alguna interrupción de *software* disponible, que permita adquirir una imagen y luego escribirla en disco.
   La `ISR` de la cámara no recibe parámetros y retorna en su registro de estado información sobre la adquisición. Si la adquisición fue exitosa, el registro contendrá `0xFF` y la imagen se encontrará en el `buffer` de la cámara. En caso contrario, si la adquisición falló, el registro contendrá `0x00`. Durante la adquisición, el registro contendrá el valor `0xF0`.
   La `ISR` del disco utiliza internamente el controlador de `DMA`, por lo que necesita los siguientes parámetros en los siguientes registros:
   - La dirección de inicio del origen en el registro `AX`.
   - La dirección de inicio del destino en disco en el registro `BX`.
   - La cantidad de palabras a copiar en el registro `CX`.

   Puede utilizar la cantidad de parámetros que estime conveniente para su `ISR`, pero debe dejar explícitamente escrito qué significan y dónde se almacenan.

b. Escriba un programa que llame a la subrutina del ítem anterior para adquirir tres imágenes y almacenarlas de manera consecutiva en disco. Considere que la adquisición puede fallar y que se intentará esta un máximo de tres veces por imagen.