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

**IIC2343 — Arquitectura de Computadores (2020-2)**

# Ayudantía 6

### Dispositivos I/O

## 1 Precalentamiento

## Ejercicios Rápidos:

1. **Convenciones de llamada:** Explique y ejemplifique la necesidad de la existencia de las convenciones de llamada.

   **Solución:**

   Cuando un programa debe interactua con otro en un mismo computador a nivel de assembly (llamando a subrutinas), es necesario que existan reglas que expliciten dónde y cómo ubicar los parámetros y valores de retornos de éstas. Este conjunto de reglas se les conoce como convenciones de llamada. Si no existieran, no sería posible la comunicación y traspaso de datos entre programas, pues ninguno sabría desde donde extraer los datos que el otro le entrega. Por ejemplo, se tiene el caso de un programa que llama a una subrutina del sistema operativo.

2. **Stdcall:** Describa una convención de llamada para x86 que sea más rápida que *stdcall* al momento de escribir y leer parámetros y valores de retorno. Contrapese las posibles ventajas y desventajas.

   **Solución:**

   Una posible convención pordría utilizar los registros de la arquitectura x86 para almacenar los parámetros de entrada de las subrutinas. Si la cantidad de parámetros llega a exceder la cantidad de registros, se utiliza el stack para los parámetros restantes. De esta manera, comparada con *stdcall*, esta convención es más rápida para leer los argumentos, pero también es más compleja dado que no todos se ubican en el mismo lugar.

3. **Vector de interrupción:** ¿Cuál es la función del vector de interrupciones? ¿Cuál es su contenido?.

   **Solución:**

   El vector de interrupciones almacena la dirección de inicio de las distintas ISR registradas en el sistema. Al momento de procesar una IRQ, se debe buscar en el vector de interrupciones la dirección de la ISP asociada con el dispositivo que generó la IRQ.

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

**IIC2343 — Arquitectura de Computadores (2020-2)**

# Ayudantía 6

**Dispositivos I/O**

## 2 Subrutina x86 con *stdcall*

## Ejercicio 1: I2 2013/1

Escriba en *assembly* x86, usando al menos una subrutina y la convención *stdcall*, un programa que realice la búsqueda de un número en un arreglo ordenado, utilizando el algoritmo de búsqueda binaria. Para implementar este algoritmo, se compara el elemento a buscar con el elemento central del arreglo. Si el valor de este elemento es mayor que el del buscado, se repite el procedimiento en la parte del arreglo que va desde el inicio de éste hasta el elemento central. En caso contrario, se toma a la parte del arreglo que va desde el elemento central hasta el final.

## Solución:

Se asume que el arreglo tendrá un tamaño par y mayor que cero, y que el número buscado (target) siempre estará en el arreglo. Sea $n$ la cantidad de elementos del arreglo y *array* la dirección de memoria del primer elemento del arreglo.

```
        MOV AX, 0          // Inicializamos en cero.
        MOV AL, target     // cargamos AX con el número que buscamos (target)
        PUSH AX          //Se pasa el parámetro target
        MOV AL, n        // cargamos AX con la cantidad de elem del array (n)
        PUSH AX            //Se pasa el parámetro n
        LEA AX, array  // cargamos AX con la dirección del primer elemento del arreglo (array)
        PUSH AX           //Se pasa la dirección del array
        CALL binary_search   // Llamamos a la subrutina
        MOV res, AX
        RET

        binary_search:
        PUSH BP          //Guardamos valor original de BP
        MOV BP, SP    //Cargamos BP con valor de SP
        MOV BX, [BP + 4]      // BX = array
        MOV SI, [BP + 6]       // SI = n
        MOV CX, [BP + 8]     // CX = target

        start:
        SHR SI, 1   SI = SI/2
        CMP [BX + SI], CL // Vemos si en la posición BX+SI del array está target (ojo: se ocupa direccionamiento)
        JE end       // Encontramos el número (target)
        JG start    //Nos pasamos, hay que buscar en la primera mitad
        ADD BX, SI  // Hay que buscar en la mitad superior del array
        JMP start

        end:
        MOV AX, BX     // Cargamos AX con la posición relativa de target en el array
        ADD AX, SI  // Le sumamos el valor de referencia para pasar a posición absoluta
        POP BP    //Recuperamos el valor de BP
        RET 6    // Retornamos la subrutina, liberando 6 bytes del stack (3 variables)

        res     dw
        target  db     ?
        n       db     ?
        array   db     ?
```

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

**IIC2343 — Arquitectura de Computadores (2020-2)**

# Ayudantía 6

**Dispositivos I/O**

## 3 Dispositivos I/O e Interrupciones

## Ejercicio 1: I2 2015/2

Asuma un dispositivo I/O, una tarjeta de red, que se conecta al computador básico utilizando los puertos 13 (comandos), 14 (estados) y 15 (datos). 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.

### Solución:

Se define el comando que inicia el proceso de recepción como 0 y el estado de dato recibido como 0.

```
DATA
CODE
    MOV   B, 0x40   // Cargamos B con la dirección desde la cual podemos escribir (0x40)
    MOV   A, 0   // Cargamos A con 0
    OUT   13, A   // Se le indica a la tarjeta que se inicia el proceso de recepción
  for:
    IN    A ,14   // Recibimos de la tarjeta el estado y lo cargamos en A
    CMP   A, 0   // Vemos si se cumple el estado de dato recibido
    JNE   for   // Si no se cumple, esperamos
    IN    A,15   // Recibimos el dato
    MOV   (B), A  // Escribimos el dato en memoria
    INC B   // Pasamos a la siguiente dirección de memoria
    JMP for
```

## Ejercicio 2: P9 2020/1

En el caso de interrupciones, considere un computador con 3 dispositivos de I/O: una impresora, un disco y un modem, con prioridades 1, 2 y 4 (más alta) respectivamente. Además tenga en cuenta que procesar una interrupción (de cualquier tipo) toma 10 unidades de tiempo. En $t = 0$, hay un programa corriendo. En $t = 10$ ocurre una interrupción del modem; en $t = 15$, la impresora genera su propia interrupción y en $t = 17$, el disco produce su propia interrupción.

1. Dibuja una línea del tiempo y especifica cuál interupción está siendo procesada y cuándo, desde $t = 0$ hata que vuelve a ejecutarse el programa original.
2. ¿Qué hay en el stack en cada uno de los intervalos relevantes de tiempo?

## Solución:

1. Entre $t = 0$ y $t = 10$, se está ejecutando el programa y no hay interrupciones.
   En $t = 10$, se empieza a procesar la interrupción del modem.
   En $t = 15$, llega la interrupción de la impresora, pero como tiene menos prioridad que la del modem, queda en espera.
   En $t = 17$, llega la interrupción del disco, pero como tiene menos prioridad que la del modem, también queda en espera.
   En $t = 20$ termina de procesarse la interrupción del modem y empieza a procesarse la interrupción del disco, ya que tiene mayor prioridad que la de la impresora.
   En $t = 30$ termina de procesarse la interrupción del disco y empieza a procesarse la de la impresora.
   Ent $t = 40$ termina de procesarse la interrupción de la impresora y se reanuda la ejecución del programa.

2. Hasta $t = 10$ y desde $t = 40$, nada.
   Entre $t = 10$ y $t = 40$ se almacenan en el stack los datos relevantes para la ejecución del programa que son básicamente los registros de la CPU (PC, STATUS y registros generales) incluyendo los condition codes.