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

IIC2343 – Arquitectura de Computadores

Ayudantía 5

Profesores: Hans-Albert Löbel Díaz, Jurgen Dieter Heysen Palacios

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

Temas a tratar

Los temas a tratar dentro de esta guía son:

- Arquitectura `x86`

Preguntas

1. a. Indique la ISA del computador básico visto en el curso y la correspondiente a la arquitectura `x86`, explicando sus diferencias.

   b. **(I2 - I/2016)** ¿Por qué es necesaria la existencia de la instrucción `RET Lit` en un computador `x86`?

   c. **(I2 - I/2013)** Al iniciar el cuerpo de una subrutina, ¿por qué es necesario ejecutar las instrucciones `PUSH BP` y `MOV BP, SP`? ¿Qué pasa si no se ejecutan?

   d. **(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.

   e. **(I2 - I/2017)** ¿Cuántas palabras de la memoria son modificadas al ejecutar la instrucción `ADD [BH], AX`?

   f. **(I2 - I/2017)** ¿Cuántas llamadas recursivas a una función es posible hacer como máximo en un computador `x86` de 16 bits? Indique claramente sus supuestos.

   g. **(I2 - II/2016)** Describa un mecanismo para, en tiempo de ejecución, escribir el código de una subrutina y luego llamarla, utilizando el `Assembly x86` de 16 bits. Asuma que tiene disponible la especificación completa de la ISA.

   h. **(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.

2. *a*) **(I2 - I/2017)** Para el siguiente programa escrito en `Assembly x86-16`, indique los valores de los registros `SP` y `BP` y del *stack* completo, al momento de ingresar al *label* `set`.

```
        MOV BL,3
        PUSH BX
        CALL func
        RET
        func: PUSH BP
              MOV BP,SP
              MOV BL,[BP + 4]
              CMP BL,0
              JE set
              MOV CL,BL
              DEC CL
              PUSH CX
              CALL func
              MOV BL,[BP + 4]
              MUL BL
              JMP end

        set: MOV AX ,1

        end: POP BP
             RET 2
```

b) **(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`?

c) **(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 simultáneamente, utilizando el algoritmo de Euclides, descrito a continuación:

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