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

IIC2343 – Arquitectura de Computadores (I/2015)

Solución Interrogación 2

Pregunta 1

a) Explique y ejemplifique la necesidad de la existencia de las convenciones de llamada. **(1 pto.)**
Solución: Cuando un programa debe interactuar con otro en un mismo computador a nivel de assembly, *i.e.* llamar subrutinas, es necesario que existan reglas que expliciten donde y cómo se ubicarán los parámetros y valores de retorno. Estas reglas son conocidas como convenciones de llamada. De no existir, no sería posible la comunicación y traspaso de datos entre programas, ya que ninguno sabría desde donde extraer los datos que entrega el otro. Un ejemplo clásico para esto es un programa que quiere llamar a una subrutina del sistema operativo.

b) Explique el paso de parámetros por valor y por referencia, desde el punto de vista de alguno de los assembly vistos en clases. De ejemplos para cada caso. **(1 pto.)**
Solución: Para el assembly x86, el paso de parámetros por valor consiste en introducir en el stack, previa llamada a la subrutina, directamente el valor del parámetro, *i.e.* MOV AX, 0x0010; PUSH AX. Esto implica que que cualquier cambio que se realice al parámetro dentro de la subrutina, no se verá reflejado fuera de esta. Para el paso por referencia, la idea es entregar no una copia del valor, sino la dirección de memoria de este, de manera que al ser modificado dentro de la subrutina, los cambios se vean reflejados fuera de ella, *i.e.* var dw 0x0010; LEA AX, var; PUSH AX.

c) Explique el comportamiento de la ley de Moore durante los últimos 10 años, en relación a la cantidad de transistores, velocidad de los procesadores, cantidad de núcleos, y relaciónelo con la evolución del consumo energético de los procesadores durante ese período. ¿Qué tipo de dispositivo ha sido el más beneficiado con este comportamiento? **(1 pto.)**
Solución: Durante los últimos 10 años, la ley de Moore ha seguido siendo cumplida desde el punto de vista del aumento de transistores. La diferencia con tiempos anteriores, es que este aumento no se ha visto reflejado en la velocidad del clock, sino que en la replicación de los núcleos de un procesador, dentro de un mismo circuito integrado. Esto permitió la creación de computadores que pueden alcanzar altos niveles de procesamiento de datos, si son usados en problemas paralelizables. Desde el punto de vista del consumo energético, durante los últimos diez años, la miniaturización ha avanzado de gran manera, permitiendo en desarrollo de procesadores que utilizan una superficie muy pequeña, lo que redunda en un menor uso energético. Todos estos cambios generaron el entorno propicio para la creación de procesadores ideales para dispositivos móviles de bajo consumo, como celulares o *tablets*.

d) Es posible agregar al assembly del computador básico la instrucción MOV A,(A+B), sin modificar la microarquitectura. Justifique su respuesta en cualquiera de los dos casos. **(1 pto.)**
Solución: Si es posible agregarla, utilizando una serie de instrucciones básicas para componerla, teniendo cuidado de no perder el valor previamente almacenado en el registro B: PUSH B; ADD B, A; MOV A, (B); POP B.

e) ¿Cuáles son los potenciales beneficios de una microarquitectura asíncrona? **(1 pto.)**
Solución: Los potenciales beneficios son mayores velocidades de proceso, menor consumo energético y menor interferencia electromagnética.

f) ¿Cuál es el máximo de instrucciones que podría tener un computador de 32 bits con arquitectura Von Neumann y 1 GB de RAM, si todas tuvieran largo de 1 palabra? (No considere los literales) **(1 pto.)**
Solución: Dado que los opcodes podrán utilizar los 32 bits completos, el máximo número de instrucciones será de 2^{32}.

Pregunta 2

a) Construya un computador OISC, que utilice la instrucción SBNZ a,b,c,d, definida de la siguiente manera:

- Mem[**c**] = Mem[**a**] - Mem[**b**]
- if(Mem[**c**] $\neq$ 0) goto **d**

Defina la arquitectura completa, *i.e.* microarquitectura e ISA. La instrucción puede ser implementada usando más de un ciclo. **(3 ptos.)**
Solución: La alternativa más simple para construir este computador, es utilizar la arquitectura del computador básico y construir un nuevo assembly, sobre el assembly original. Este nuevo assembly solo necesitaría implementar la instrucción **SBNZ**, mediante una secuencia de instrucciones del assembly del computador básico. Una posible solución es la siguiente:

Instrucción	Operandos	Nueva Instr.

SBNZ	a,b,c,d	MOV A, (a)
		MOV B, (b)
		SUB A, B
		MOV (c), A
		JNE d

Es interesante notar que no es necesario agregar la instrucción **CMP**, ya que la transferencia a la dirección de memoria **c** del dato almacenado en **A**, generará el flag.

b) El **throughput** de un procesador se define como la cantidad de instrucciones aritméticas o lógicas que pueden ejecutarse en una cierta cantidad de ciclos. Por ejemplo, si en el computador básico tenemos una sección de código que consiste en diez llamadas seguidas a la instrucción ADD, el **throughput** del procesador en esa sección será de 1.0. Teniendo esto en consideración, modifique la arquitectura del computador básico, para incluir al menos una instrucción que permita aumentar el **throughput** del procesador. **(3 ptos.)**
Solución: Para aumentar el **throughput**, basta con agregar una nueva ALU al computador básico, ALU2, e incluir una nueva instrucción que la utilice, como por ejemplo **ADD A,B,Lit**, que almacena en A la suma de los contenidos de **A** y **B** y el literal **Lit**. La ALU2 se debe conectar a continuación de la ALU original, recibiendo en su entrada A el valor de la ALU, y en la entrada B el valor del literal **Lit**, mientras que el resultado se conecta como estaba conectada la ALU anteriormente. Para mantener la compatibilidad con las instrucciones anteriores, es necesario que la entrada B de la ALU2 sea alimentada por un MUX, MUX_ALU2, que decide entre 0 y el literal **Lit**. Esto implica que es necesario agregar una nueva señal de control, que será 0 para todas las instrucciones originales, y 1 para la nueva instrucción **ADD A,B,Lit**. Finalmente, el opcode para la nueva instrucción será 1001110.

Pregunta 3

a) Se define como $H_{N,K}$ al conjunto de todos los vectores en $\Re^N$, que se encuentran en la superficie de una hiperesfera de radio K y que sólo tienen coeficientes enteros. Teniendo esto en consideración y dados N, K, un vector $\vec{h} \in H_{N,K}$ y una matriz H definida como $H = matriz(H_{N,K} - \{\vec{h}\})$, donde la función $matriz$ toma un conjunto de vectores y los transforma en una matriz de vectores, donde cada fila corresponde a un vector, escriba un programa en assembly x86 que encuentre en la matriz H el vector más cercano a $\vec{h}$, en base a la distancia **euclidiana**. En caso de usar subrutinas, use la convención *stdcall*. **(3 ptos.)**
Hint: recuerde que la norma de un vector $\vec{v}$ se calcula como $\|\vec{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} = \sqrt{\vec{v} \cdot \vec{v}}$.

Solución: Para responder esta pregunta existen dos caminos. El difícil, que consiste en calcular para cada vector de la matriz **H**, la distancia euclidiana con **h** (sin tomar en cuenta la raíz), y luego retornar el índice de aquel con menor distancia. Por otro lado, el camino fácil (?) requiere notar que al encontrarse todos los vectores sobre la misma hiperesfera, todos tendrán la misma norma igual a K. Esto implica que la norma de la diferencia entre 2 vectores, lo que es análogo a la distancia euclidiana entre ellos, se puede expresar de la siguiente manera: $\|(\vec{v_1} - \vec{v_2})\| = \sqrt{(\vec{v_1} - \vec{v_2}) \cdot (\vec{v_1} - \vec{v_2})} = \sqrt{2K^2 - \vec{v_1} \cdot \vec{v_2}}$. Esto significa que el vector más cercano es aquel con el mayor valor para el producto punto. La solución final se presenta a continuación:

```
    JMP    main
    h      db    ?
    H      db    ?
    N      dw    ?
    rows   dw    ?
    res    dw
    main:
    MOV    SI, 0       ;iterador para las filas
    MOV    DI, -32768 ;almacena el maximo prod. punto
    LEA    BX, H
    LEA    CX, h
  for:
    CMP    SI, rows
    JEQ    end
    PUSH   N
    PUSH   BX
    PUSH   CX
    CALL   prod_punto
    CMP    AX, max
    JLE    end_for
    MOV    max, AX
    MOV    res, SI
  end_for
    ADD    BX, N
    ADD    SI, 1
        JMP    for
  end:
    RET
    prod_punto:
        PUSH   BP
        MOV    BP, SP
    PUSHA
    MOV    DX, [BP-8]
        MOV    CX, [BP-6]
        MOV    BX, [BP-4]
    MOV    SI, 0
    MOV    DI, 0
  for_prod_punto:
    CMP    DX, SI
        JEQ    end_prod_punto
    MOV    AL, [CX+SI]
    IMUL   [DX+SI]
    ADD    DI, AX
    ADD    SI, 1
    JMP for_prod_punto
  end_prod_punto:
    POPA
        POP BP
    MOV AX, DI
        RET 6
```

b) I. 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 como 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. **(2 ptos.)**

Solución: Para esta pregunta, el análisis clave tiene que ver con qué instrucciones modifican el **PC**, sin contar lógicamente a **CALL**. Sólo existen 2: **RET** y las instrucciones de salto. El problema con estas últimas es que tienen la misma limitante que **CALL**, por lo que están descartadas. Esto nos deja sólo con **RET**, que, a pesar de que pueda parecer poco intuitivo, puede realizar justamente lo que necesitamos. **RET** carga en el **PC** el valor almacenado en el tope del stack, por lo que basta con cargar en el stack la dirección donde queremos saltar y luego llamar a **RET**. Para el retorno de una subrutina llamada de esta manera, basta con cargar en el stack, antes de introducir la dirección de la subrutina, la dirección donde se quiere retornar, de manera que el **RET** que es llamado dentro de la subrutina, extraiga desde el stack la dirección de retorno correcta. En resumen, si asumimos que la dirección a la que queremos saltar se encuentra almacenada en el registro **A**, la solución al problema es la siguiente:

```
    PUSH retorno
    PUSH A
    RET
retorno:
```

II. En la memoria de instrucciones del computador básico se encuentran almacenadas una gran cantidad subrutinas distintas y muy pequeñas, donde cada una no supera las 5 líneas. Estas subrutinas se encuentran en las direcciones 50, 55, 60, ..., 250, sin labels que las identifiquen. Basándose en lo obtenido en el ítem anterior, escriba en assembly del computador básico, un programa que dado un número natural n múltiplo de 5, llame a la subrutina ubicada en la dirección de memoria indicada por n, siempre que esté dentro del rango correcto. **(1 pto.)**

Solución:

```
    MOV A, (n)
    CMP A, 50
    JLT end
    CMP A, 250
    JGT end
    PUSH end
    PUSH A
    RET
end:
```