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

IIC2343 – Arquitectura de Computadores (II/2011)

Solución Interrogación 2

Pregunta 1

a) Si la ROM de instrucciones del computador básico se cambia por una RAM, ¿es necesario modificar otro aspecto de la arquitectura para que sea autoprogramable? **(0,75 ptos.)**

Solución: Si, es necesario modificar la ISA, agregando nuevas instrucciones para transferir datos de/hacia la memoria de instrucciones.

b) Describa dos instrucciones que son soportadas por el computador básico pero que no existen en el assembly de este. **(0,75 ptos.)**

Solución: Existen muchísimas opciones, ya que de las 2^{17} palabras de control posibles, sólo se utilizan 68 opcodes. Dos ejemplos posibles son: (i) la instrucción que suma los registros A y B y luego almacena el resultado simultáneamente en los registros A y B, y (ii) la instrucción que suma los registros A y B y luego no almacena el resultado en ningún lado.

c) Un disco duro es conectado a un computador x86 y su comunicación se realiza mediante port I/O. Describa una manera de controlar este dispositivo, considerando cuantos puertos son necesarios para utilzarlo y la función de cada uno de estos. **(0,75 ptos.)**

Solución: Para controlar este dispositivo, se pueden utilizar 3 puertos. Uno para enviar comandos al disco, que serían recibidos por el controlador de este, un segundo puerto para verificar el estado de la operación actual del disco y un tercer puerto para el manejo de los datos, i.e., para poner las direcciones de memoria y luego el o los datos asociados.

d) En un computador con arquitectura x86, ¿cuál es contenido del stack y la posición a la que apuntan los registros BP y SP, después de ejecutar las siguientes instrucciones? Asuma que el stack comienza vacio. **(0,75 ptos.)**

```
MOV AX, 0xABCD
PUSH AX
MOV AL, 0x12
PUSH AX
CALL func
RET
func: PUSH BP
MOV BP, SP
SUB SP, 2
MOV BL, [BP+4]
MUL BL
PUSH AX
RET
```

Solución: En el diagrama, cada espacio entre líneas es de 16 bits. A su vez, cuando se almacenan datos, estos se dividen en dos partes de 8 bits, divididos por líneas punteadas, para hacer explícito el tamaño de las palabras en memoria. Cabe mencionar que en x86, los datos se almacenan usando little endian. Para el caso de las direcciones, no se utiliza la línea punteada. A pesar de que no es necesario que esté incluido en el diagrama, el valor de 16 bits que antecede a la posición apuntada por SP corresponde a 12h×12h = 0144h

	0x44
	0x01
SP →	
BP →	BP anterior
	IP+1
	0x12
	0xAB
	0xCD
	0xAB

e) ¿Es posible utilizar recursividad en el computador básico? ¿Son necesarias consideraciones adicionales? **(0,75 ptos.)**

Solución: La recursividad puede usarse en el computador básico, siempre y cuando no se utilizen parámetros de entrada ni valores de retorno mediante variables. Si quisieran usarse estos elementos, debería utilizarse el stack para almacenar los valores, similar a la convención stdcall, y emular el registro BP mediante la utilización de una posición fija de la memoria.

f) ¿Como se podría implementar en el computador básico la opción de que avise luego de realizar una operación cuando el resultado es par o impar? **(0,75 ptos.)**

Solución: Esto se puede implementar de manera similar a los flags Z o N. En este caso, el

bit P se implementaría verificando el bit menos significativo del número analizado. Si este es 1, P es 1, y si este bit es 0, P es cero.

g) ¿Qué modelo de interacción entre CPU, memoria e I/O recomendaría y por qué, para un disco duro, un mouse y una cámara de video? **(0,75 ptos.)**

Solución: Para un disco duro se recomienda el esquema de interrupciones con DMA, ya que un disco duro es un dispositivo con alto nivel de interacción con la CPU, cuyas transferencias son generalmente de gran tamaño. Para un mouse se recomienda el esquema de interrupciones sin DMA, ya que a pesar de que el mouse presenta un alto nivel de interacción, las transferencias de este son pequeñas. Finalmente, para la cámara de video, se recomienda el esquema de interrupciones con DMA, ya que el tamaño de las transferencias es muy alto y con bastante frecuencia.

h) Un programa en C es compilado para una ISA CISC y otra RISC. ¿Cual de las dos compilaciones tendrá como resultado un código de mayor tamaño y por qué ocurre esto? **(0,75 ptos.)**

Solución: El código para la ISA RISC tendrá un tamaño de compilación mayor, ya que RISC define sets de instrucciones muy simples, que redundan en una mayor cantidad de instrucciones que CISC para hacer una tarea específica.

Pregunta 2

a) Escriba en assembly x86, usando **stdcall**, un programa que calcule el producto de un vector y una matriz de tamaños arbitrarios, donde sus elementos son números enteros de 8 bits. El programa debe usar la subrutina `multiplicarVecMat`, que también debe ser implementada. Todos los detalles de implementación, por ej., como y donde se almacenan los datos, etc., deben ser claramente explicados. **(3 ptos.)**

Solución: Se asume que el vector se encuentra a la izquiera de la matriz, por lo que sólo puede ser un vector fila. También se asume que tanto matrices como vectores se almacenan en orden de filas, que sus direcciones de inicio corresponden a los labels vec y mat, que el número de columnas del vector se almancena en cols_vec y el de columnas de la matriz se almacena en cols_mat. Finalmente, se asume que el vector resultante se almacena en res.

```
start:
    PUSH res
    PUSH [cols_mat]
    PUSH mat
    PUSH [cols_vec]
    PUSH vec
    CALL multiplicarVecMat
    RET

multiplicarVecMat:
    PUSH BP
    MOV BP, SP
    SUB SP, 4
    MOV [BP-2], 0
    MOV [BP-4], 0
    for1:
        CMP [BP-2], [BP+10]
        JE end_for1
        for2:
            CMP [BP-4], [BP+6]
            JE end_for2
            MOV BX, [BP+4]
            MOV SI, [BP-4]
            MOV CL, [BX + SI]
            MOV AL, [BP-2]
            MUL [BP + 10]
            ADD AX, [BP + 8]
            ADD AX, [BP-4]
            MOV AL, [AX]
            MUL CL
            MOV BX, [BP+12]
            ADD BX, [BP-2]
            ADD [BX], CL
            INC [BP-4]
            JMP for2
        end_for2:
        INC [BP-2]
        JMP for1
    end_for1:
    ADD SP, 4
    RET 10
```

b) Usando la subrutina `multiplicarVecMat`, escriba un programa que calcule el producto entre dos matrices de tamaño arbitrario. **(2 ptos.)**

Solución: La manera más sencilla de utilizar la subrutina multiplicarVecMat es llamarla una vez por cada fila de la matriz de la izquiera del producto.

```
start:
    PUSH res
    PUSH [cols_mat2]
    PUSH mat2
    PUSH [cols_mat1]
    PUSH [rows_mat1]
    PUSH mat1
    CALL multiplicarMatMat
    RET

multiplicarMatMat:
    PUSH BP
    MOV BP, SP
    SUB SP, 2
    MOV [BP-2], 0
    for:
        CMP [BP-2], [BP+6]
        JE end_for

        MOV AL, [BP-2]
        MUL [BP+12]
        ADD AX, [BP + 14]
        PUSH AX

        PUSH [BP+12]
        PUSH [BP+10]
        PUSH [BP+8]

        MOV AL, [BP-2]
        MUL [BP + 8]
        ADD AX, [BP+4]
        PUSH AX

        CALL multiplicarVecMat

        INC [BP-2]
        JMP for
    end_for:
    ADD SP, 2
    RET 12
```

c) La ISA x86 tiene instrucciones avanzadas, como MUL, que facilitan bastante el desarrollo de programas. Describa una instrucción aritmética avanzada, implementable con la arquitectura x86, que facilite el cómputo del producto entre vector y matriz antes descrito. ¿Qué consideraciones habría que tener al usar esta instrucción? **(1 pto.)**

Solución: Una posible instrucción avanzada es el computo simultáneo de dos multiplicaciones con operandos de 8 bits que sean números enteros. Dado que los registros son de 16 bits, esta operación podría implementarse de manera muy sencilla en la ALU, con la consideración de que el resultado del producto siempre va a estar limitado a números de 8 bits.

Pregunta 3

a) Reescriba el siguiente código usando el assembly del computador básico, siguiendo la misma secuencia de operaciones. **(1 pto.)**

```
byte arreglo = new byte[]{6,7,3,4,5};
byte n = 5;
byte suma = 0;
byte i = 0;
for (i = 0; i < n; i++)
{
  suma += arreglo[i];
}
```

Solución:

```
DATA:

arreglo 6
        7
        4
        5
        3
n       5
i       0
suma    0

CODE:

for:
        MOV A, (n)
        MOV B, (i)
        CMP A, B
        JEQ end

        MOV A, arreglo
        ADD B, A
        MOV B, (B)
        MOV A, (suma)
        ADD (suma)

        MOV B, (i)
        INC B
        MOV (i), B
        JMP for
end:
```

b) En la posición 0x100 de memoria de un computador x86 se almacena un malware. Escriba un programa en assembly x86 que logre ejecutar este código maligno. **(2 ptos.)**

Solución: El objetivo de este problema es cargar el valor 0x100 en el registro IP (Instruction Pointer). Como no es posible modificarlo de manera directa, la única opción es usando la instrucción RET. De esta manera, una posible solución es la siguiente:

```
CALL func
RET

func:
MOV AX, 100h
PUSH AX
RET
```

c) Explique por qué es posible realizar este tipo de ataques en la arquitectura x86 y modifique esta arquitectura para que no puedan ocurrir estos ataques. **(3 ptos.)**

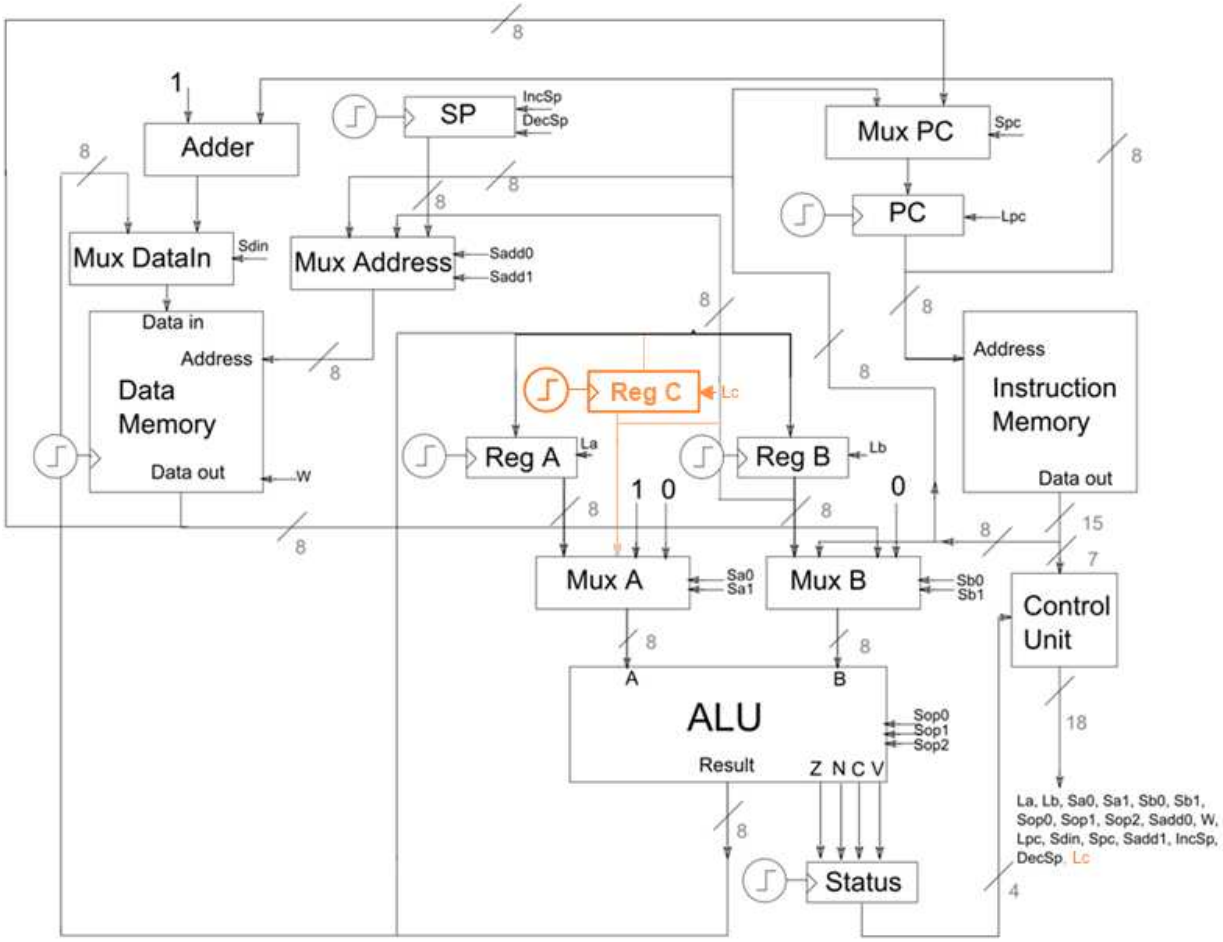
Solución: Estos ataques pueden realizarse debido a la incapacidad de las arquitecturas Von Neumann de diferenciar instrucciones de datos. Una posible solución es modificar la memoria principal, de manera que cada posición de memoria, además de contener una palabra, contenga además un bit que indique si la palabra almacenada corresponde a una instrucción o a un dato. Además de modificar la memoria, es necesario modificar las instrucciones de transferencia y salto. Las de transferencia deben diferenciar explícitamente cuando se quieren guardar datos o instrucciones en memoria y las de salto deben anularse cuando la dirección de llegada corresponda a una posición utilizada por un dato.

Pregunta 4

Se desea modificar el computador básico a nivel de microarquitectura e ISA. Para los siguientes puntos detalle las modificaciones que haría. Debe utilizar diagramas de componentes y conexiones y tablas de opcodes e instrucciones cuando corresponda. Las modificaciones son acumulativas.

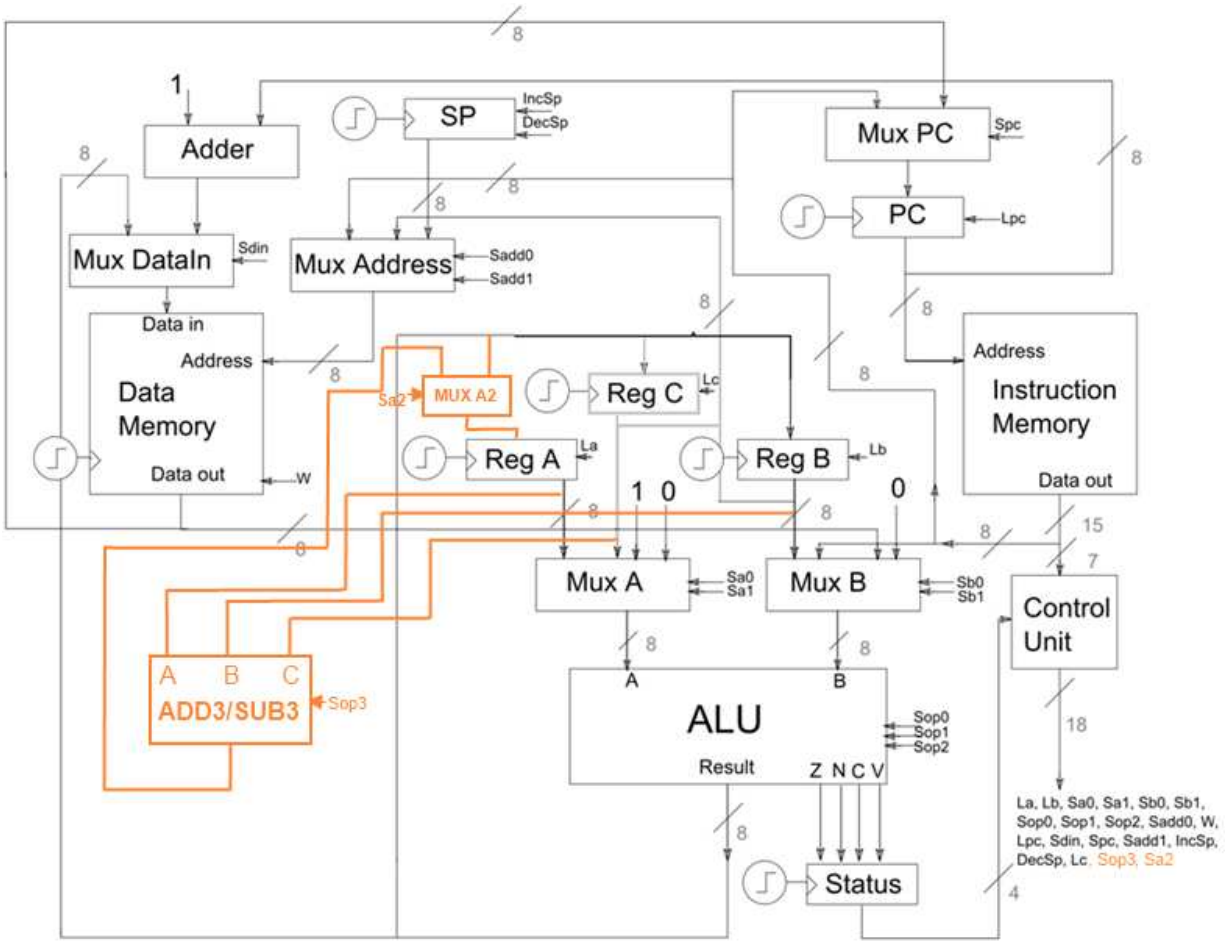
a) Agregar un tercer registro, que cumpla la función de acumulador de resultados y que pueda usarse para direccionamiento indirecto. **(1,5ptos.)**

Solución: En este caso, para permitir el paso de C a la ALU, se agrega la salida de este a Mux A.



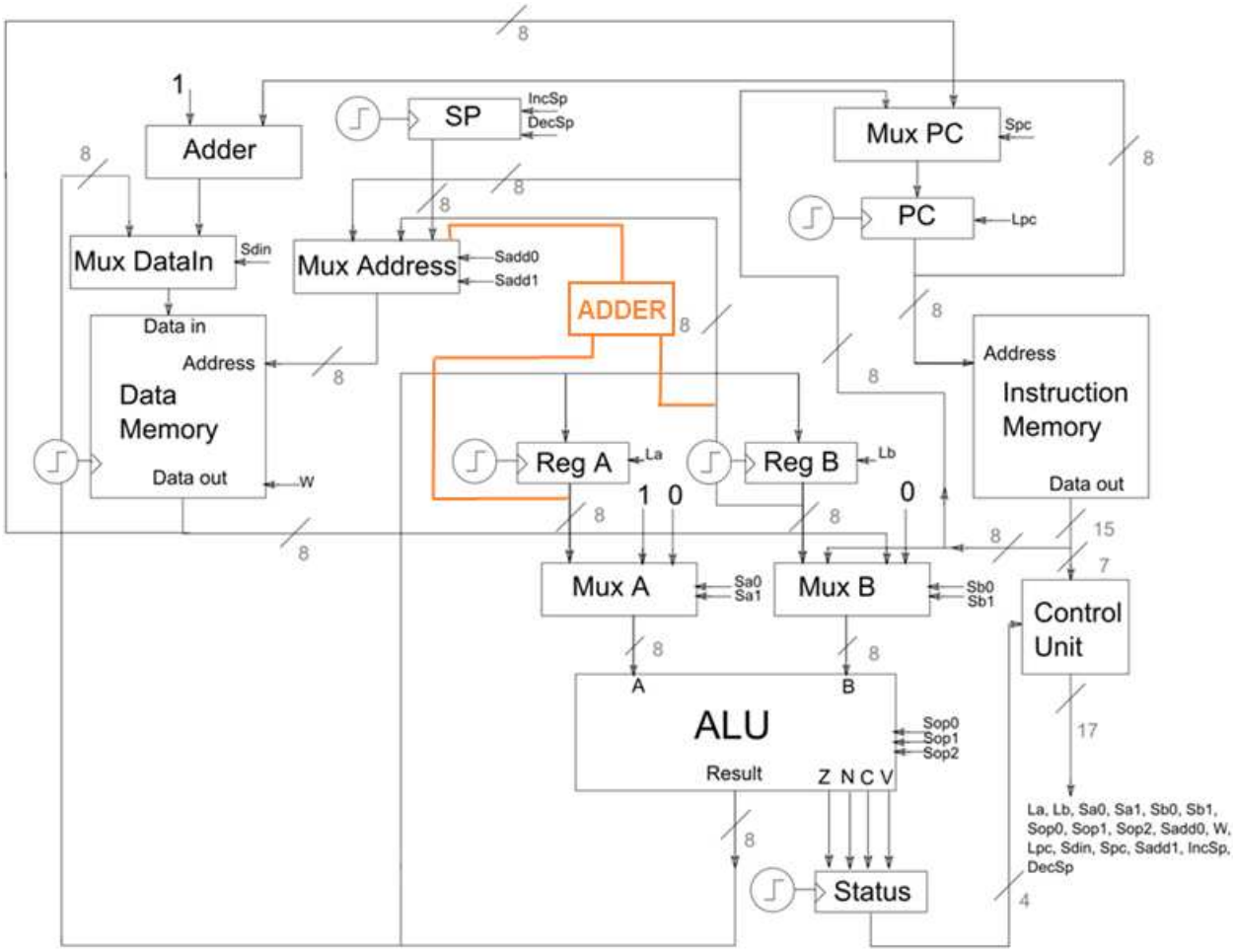
b) Agregar las instrucciones ADD3 reg1, reg2, reg3 y SUB3 reg1, reg2, reg3, que toman los valores en los registros reg1, reg2 y reg3, los suma/resta y los almacena en reg1. **(1,5 ptos.)**

Solución: Se asume que el orden de los operandos de las operaciones corresponden a A=reg1, B=reg2 y C=reg3. Además del diagrama, es necesario especificar el comportamiento de las nuevas señales de control. Sop3 es 0 cuando la operación elegida es la suma de 3 registros, mientras que es 1 cuando es la resta. Por otro lado, la señal Sa2 es 0 cuando se quiere almacenar en el registro A el resultado de la operación SUB3/ADD3, y es 1 cuando se quiere almacenar el resultado proveniente de la ALU. Dado que aun quedan opcodes de 7 bits disponibles, se asignan los opcodes 1001111 para ADD3 y 1010000 para SUB3.



c) Soportar direccionamiento indirecto con registro base y registro índice. **(1,5 ptos.)**

Solución: Para seleccionar como entrada de Mux Address el resultado de la suma de A y B, se utiliza el valor 1 tanto para Sadd0 y Sadd1. La instrucción de transferencia que hace uso de este direccionamiento será MOV A, [A+B], que tendrá el opcode 1010001 y la única diferencia con las señales de control de MOV A, [B] son los valores de Sadd0 y Sadd1.



d) Permitir la conexión de dispositivos externos con port I/O, usando 256 puertos. **(1,5 ptos.)**

Solución: Para utilizar la conexión de I/O, es necesario agregar las instrucciones IN B, port y OUT port, B, con opcodes 1010010 y 1010011 respectivamente. IN pone la señal Sb2 en 1, para que se almacene en el registro B el valor proveniente del I/O, y la señal EnIO en 1, para habilitar los dispositivos de I/O. Por otro lado, OUT sólo pone la señal EnIO en 1, ya que no necesita almacenar nada en el registro B. Cabe destacar que el puerto es extraido tal cual como los literales de las instrucciones y que la señal EnIO sólo se pone en 1 cuando se requiere interacción de los dispositivos de I/O. Se asume que los dispositivos de I/O entregan el resultado de manera inmediata, tal cual como una memoria.