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



IIC2343 – Arquitectura de Computadores (II/2011)

Solución Interrogación 1

Pregunta 1

a) Se tienen dos números de punto flotante de precisión simple en formato IEEE754: $A = 0x3E200000$ y $B = 0x00000000$. ¿Cuál es el resultado, en formato IEEE754, de $A : B$? **(0,75 ptos.)**

Solución: A > 0 y B = 0, luego, dado que la división entre un número positivo y cero, en IEEE754, es $+\infty$, el resultado en correcto es $0x7F800000 = 01111111100000000000000000000000$.

b) ¿Cuántos relés se necesitan para representar cada una de las siguientes compuertas: AND, OR, NOT y XOR? **(0,75 ptos.)**

Solución: AND = 2, OR = 2, NOT = 1. Para el XOR hay dos opciones 1. $(\neg p \wedge q) \vee (p \wedge \neg q) = 8$, 2. $\neg(p \wedge q) \wedge (p \vee q) = 7$. Cualquier resultado que indique una cantidad diferente de componentes debe ir acompañado con un diagrama o una explicación.

c) Dados $A = 45$ y $B = 57$, ¿cuál es el resultado, en binario, de la operación $A - B$? **(0,75 ptos.)**

Solución: En complemento de 2 de 7 bits, $A = 0101101$, $B = 0111001$, luego, $A - B = -12 = 1110100$.

d) ¿Cual es la diferencia entre un latch y un flip-flop? **(0,75 ptos.)**

Solución: Un latch puede realizar cambios de estado mientras se encuentra con la señal de control en 1, mientras que el flip-flop sólo lo hace cuando está en uno de los dos flancos, ya sea de subida o de bajada.

e) Un programador desarrolla una aplicación bancaria de manejo de créditos. ¿Que consideraciones debe tomar en relación a los tipos de dato a usar? **(0,75 ptos.)**

Solución: El programador debería utilizar tipos de datos numéricos que usen base decimal y no binario, como el BigDecimal de Java.

f) Al leer 2 palabras contiguas de 1 byte de una memoria en orden big endian, se obtiene el número 272. ¿Qué número se obtendría al leer estas palabras en orden little endian? **(0,75 ptos.)**

Solución: 272 = 0x0110, luego, en little endian, el valor obtenido es 4097 = 0x1001.

g) ¿Cuántos bytes puede almacenar una memoria con palabras de 40 bits y bus de direcciones de 12 bits? **(0,75 ptos.)**

Solución: $\frac{2^{12} \times 40}{8} = 20480$.

h) Dada una representación de punto fijo, con 1 bit para el signo, 3 para la parte entera y 4 para la fraccional, y una de punto flotante con 1 bit para el signo del significante, 3 para el significante, 1 para el signo del exponente y 3 para el exponente. ¿Cuál de estas representaciones recomendaría para almacenar el número 0,10111101b y por qué? **(0,75 ptos.)**

Solución: Es mejor utilizar la representación de punto fijo, ya que pierde menos bits de la parte fraccional del número.

Pregunta 2

a) Describa una representación y/o mecanismo para obtener el inverso aditivo de un número en representación hexadecimal. Este proceso no debe incluir conversiones a otras representaciones numéricas, como binario o decimal. **(3 ptos.)**

Solución: Para obtener el inverso aditivo de un número hexadecimal se puede utilizar el siguiente proceso, llamado complemento a 16:

a) Complementar cada dígito del número con respecto a F. Ej.: $C(0x43) = 0\text{x}BC$.

b) Sumar uno al complemento

Para demostrar que este procedimiento es correcto, basta realizar la suma entre un número y su inverso: $0\text{x}43 + C16(0\text{x}43) = 0\text{x}43 + 0\text{x}BD = 0$.

b) Usando el proceso descrito anteriormente, calcule el resultado de $0\text{x}43 - 0\text{x}64$. Indique claramente cada una de las etapas del proceso. **(1,5 ptos.)**

Solución:

a) $0\text{x}43 - 0\text{x}64 = 0\text{x}43 + -0\text{x}64$

b) Obtener el complemento a 16 de 0x64 : $C16(0\text{x}64) = 0\text{x}9C$

c) Realizar la suma: $0\text{x}43 + 0\text{x}9C = 0\text{x}DF$

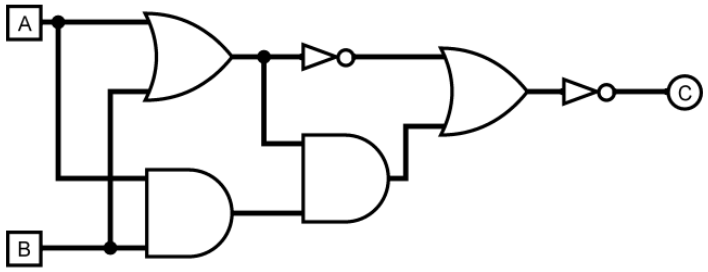
d) Dado que $C16(0\text{x}DF) = 0\text{x}21 = 33$, el resultado es correcto.

c) ¿Usando este esquema, cuál es el número hexadecimal con mayor valor absoluto que puede representarse usando sólo 2 dígitos?, ¿Por qué? **(1,5 ptos.)**

Solución: La cantidad de número hexadecimales representables con dos dígitos son 256 (8 bits). Luego, el número con mayor absoluto que puede representarse es $-128 = 0\text{x}80$.

Pregunta 3

a) Obtenga una fórmula de lógica booleana que represente el circuito de la figura. Utilice sólo los conectivos $\wedge$, $\vee$ y $\neg$. Deje por escrito el desarrollo que justifique su respuesta. **(2 ptos.)**



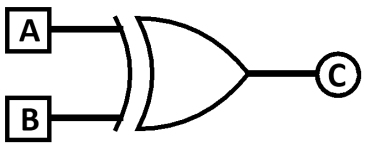
Solución: La tabla de verdad del circuito es la siguiente:

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Esta tabla corresponde a la del xor, luego, una fórmula de lógica booleana que satisface lo requerido es la siguiente: $(\neg A \wedge B) \vee (A \wedge \neg B)$.

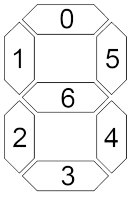
b) ¿Es posible construir un circuito equivalente al anterior, usando menos compuertas y no necesariamente las mismas? De ser así, diga cuantas son necesarias y dibuje el circuito. Justifique su respuesta en cualquiera de los dos casos. **(2 ptos.)**

Solución: Dado que el circuito es equivalente a una compuerta xor, el diagrama de éste es:

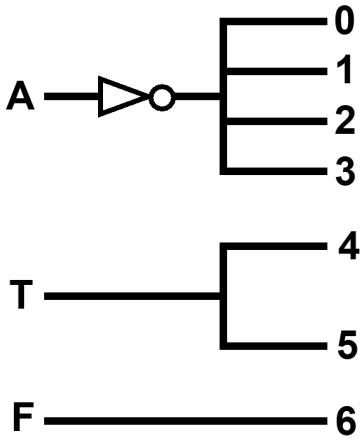


c) Construya con compuertas lógicas el controlador de un display de 7 segmentos. Este controlador recibe como input sólo un bit, i.e., el display de 7 segmentos sólo mostrará 0 ó 1. **(2 ptos.)**

Solución: Dado la siguiente enumeración de los segmentos del display:



Un circuito controlador del display puede ser el siguiente:



Donde T corresponde a una señal constante de 1 lógico (true) y F a una señal constante de 0 lógico (false).

Pregunta 4

Una imagen en escala de grises se define por su tamaño en pixeles y la profundidad de cada uno de estos, i.e., la cantidad de valores distintos que puede tomar un pixel. En base a estos, conteste las siguientes preguntas:

1. Dada una memoria con bus de direccionamiento y palabras de 1 byte, ¿cómo almacenaría en ella una imagen en escala de grises de 5×5 pixeles y 16 bits de profunidad? **(0.75 ptos.)**

   Solución: La imagen se puede representar correctamente mediante una matriz, que se almacena usando la secuencia de filas, donde cada uno de los elementos ocupa 2 palabras en la memoria.

2. Obtenga una expresión para la dirección de memoria del elemento (i, j) de la imagen del ejercicio anterior, donde $(1, 1)$ corresponde a las cordenadas del pixel superior izquierdo. Asumiendo que la imagen está almacenada en la memoria a partir de la dirección 0x6E, calcule la dirección de memoria del pixel ubicado en $(3, 4)$. **(0.75 ptos.)**

   Solución: La expresión para la dirección de memoria de un elemento de una matriz está dada por:

   $$dir(matriz[i, j]) = dir(matriz) + i \times sizeof(matriz[i, j]) \times cols + j \times sizeof(matriz[i, j])$$

   Dado que la imagen está con un desfase de un pixel por dimensión (comienza en (1,1)) con respecto a una matriz, sus dimensiones son 5×5 y cada elemento tiene un tamaño de dos palabras, la expresión anterior adaptada a la image queda de la siguiente manera:

   $$dir(img[i, j]) = dir(img) + (i - 1) \times 2 \times 5 + (j - 1) \times 2$$

   Evaluando esta expresión en el pixel (3,4) y en la dirección de inicio de la image, 0x6E, obtenemos la siguiente dirección de memoria:

   $$dir(img[3, 4]) = 0x6E + 2 \times 2 \times 5 + 3 \times 2 = 0x6E + 20 + 6 = 0x88 = 136$$

3. Una imagen en colores se diferencia de una imagen en escala de grises en que cada uno de los pixeles posee ahora 3 valores de intensidad distintos, uno para cada color (RGB). Esto es análogo a tener 3 imágenes de escala de grises distintas, donde cada una describe la distribución de intensidad para un color.
   ¿Como almacenaría una imagen en colores de 7×7 pixeles y 1 byte de profundidad por color en una memoria igual a la descrita anteriormente? **(1.5 ptos.)**

   Solución: Para almacenar una imagen en colores, basta con almacenar los 3 valores de color de cada pixel uno detrás de otro. De esta manera, almacenar esta imagen en específico es análogo a almacenar una matriz en orden de filas, donde cada uno de los elementos tiene 3 palabras de largo.

4. ¿Porqué no sirve esta misma memoria para almacenar una imagen en color de 7×7 pixeles y 2 bytes de profundidad por color? Indique el problema y una posible solución. **(1 pto.)**

Solución: Una imagen en colores de 7×7 pixeles y bytes bytes de profundidad por color, pesa $7 \times 7 \times 3 \times 2 = 294$ bytes. Dado que la memoria que se tiene es de sólo $2^8 \times 1$ bytes, la imagen no alcanza. La solución es utilizar una memoria de mayor tamaño, i.e., que tenga una mayor cantidad de palabras disponibles.

5. Obtenga una expresión para la dirección de memoria del elemento (i, j, k) de una imagen en colores, donde i, j es la ubicación del pixel y k es uno de los tres canales de color. **(2 ptos.)**

   Solución: Para obtener una expresión, basta sólo con adaptar la expresión obtenida anteriormente:

$$\begin{aligned} dir(img[i,j,k]) &= dir(img) + i \times sizeof(img[i,j,k]) \times 3 \times cols + \\ & \quad j \times sizeof(img[i,j,k]) \times 3 + k \times sizeof(img[i,j,k]) \end{aligned}$$

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

IIC2343 – Arquitectura de Computadores (I/2012)

Solución Interrogación 1

Pregunta 1

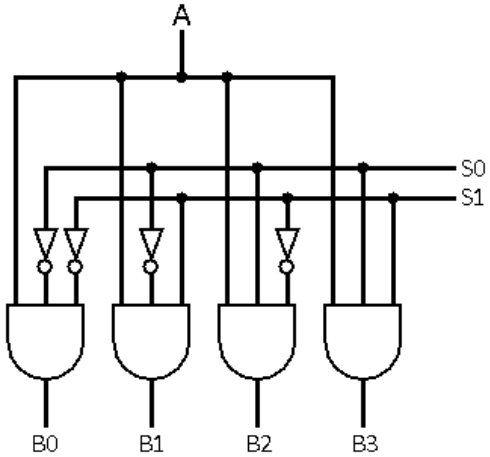
a) ¿Qué se puede decir en relación a las operaciones de suma y multiplicación en los números de punto flotante, con respecto a la pérdida de precisión? **(1 pto.)**

Solución: La suma, debido al proceso de equilibrio de exponentes, genera una mayor pérdida de precisión que la multiplicación. Luego, es preferible primero multiplicar y luego sumar.

b) ¿Que cosas se debe conocer de un arreglo para extraer e interpretar sus datos correctamente de una memoria?**(1 pto.)**

Solución: Se debe conocer la dirección de inicio del arreglo, su largo, el tipo de dato de sus elementos y su endiannes.

c) Diseñe un De-Multiplexor con bus de datos de 1 bit y bus de control de 2 bits.**(1 pto.)**
Solución:



d) ¿Que permite que un flip-flop D funcione en un flanco y no en un estado? **(1 pto.)**

Solución: Esto lo permite la conexión de 2 latches D en serie, donde la señal de control de uno es la negación de la del otro.

e) ¿Cuantos conectivos ternarios distintos se pueden definir en la lógica booleana?**(1 pto.)**

Solución: La tabla de verdad de cada conectivo binario tendrá $2^3 = 8$ entradas y cada entrada puede tomar 2 valores (lógica booleana). Luego, existen $2^8 = 256$ conectivos ternarios.

f) ¿Que consideración tiene el estándar IEEE754 con respecto al significante normalizado sin 1 y al número 0? **(1 pto.)**

Solución: Dado que el significante siempre contiene un 1, el estándar define al número 0 como el número que tiene sólo ceros en el exponente y el significante. El bit de signo del significante es libre, por lo que existe el +0 y el -0.

Pregunta 2

Dada una lógica ternaria con valores V, F y ?, y las siguientes tablas de verdad para los conectivos $\wedge$, $\vee$ y $\neg$:

A $\wedge$ B	V	F	?
V	V	F	?
F	F	F	F
?	?	F	?

A$\vee$B	V	F	?
V	V	V	V
F	V	F	?
?	V	?	?

A	$\neg$A
V	F
F	V
?	?

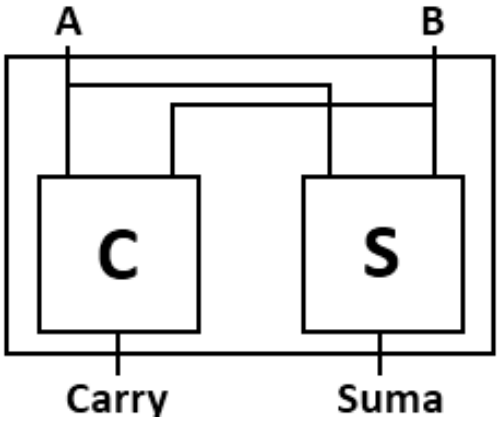
Conteste las siguientes preguntas:

a) Asigne los símbolos de la lógica ternaria antes descrita a los números 0, 1 y 2 y diseñe un full-adder que sume dos números de un trit (ternary digit). Puede utilizar los conectivos binarios de trits que estime conveniente para diseñar las compuertas. **(4 ptos.)**

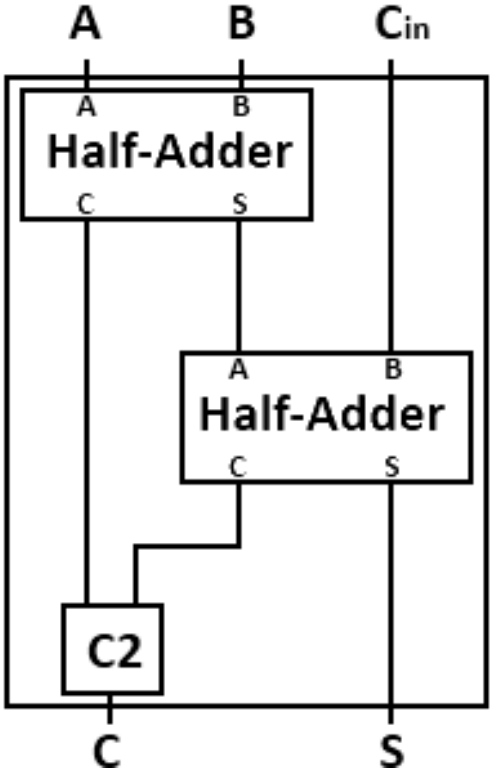
Solución: La asignación de símbolos a los números puede ser cualquiera, siempre y cuando las tablas de verdad usadas respeten esta asignación. En este caso, se utilizó la siguiente asignación: $? = 0$, $V = 1$, $F = 2$. Para construir un full-adder de trits, el primero paso consiste en construir un half-adder y luego, usando dos half-adders construir un full adder. Usando la asignación de símbolos a números anterior, la tabla de verdad para las salidas de carry (**C**) y suma (**S**) de un half-adder, dados trits en las entradas **A** y **B**, es la siguiente:

A	B	C	S
0	0	0	0
0	1	0	1
0	2	0	2
1	0	0	1
1	1	0	2
1	2	1	0
2	0	0	2
2	1	1	0
2	2	1	1

Luego, si creamos una compuerta de lógica ternaria para la columna **C** y otra para la columna **S**, el diagrama del circuito de un half-adder es el siguiente:



A continuación, para construir el full-adder basta con observar que el carry de un half adder toma como valor máximo un 1 y que además, al agregar la señal de carry-in a un full-adder, el resultado máximo de esta suma sigue necesitando 2 trits. Luego, podemos diseñar el full-adder ternario usando un esquema similar a un full-adder binario:



donde la compuerta **C2** tiene la siguiente tabla de verdad:

A	B	C2
0	0	0
0	1	1
1	0	1

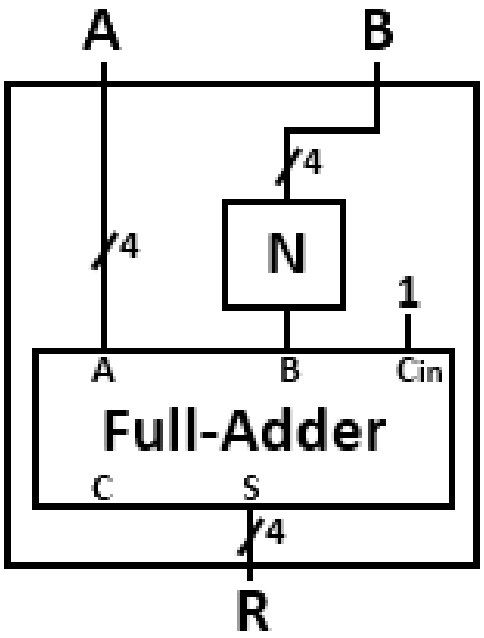
Cabe destacar que dado que nunca los carry que entegan los half-adders serán 1 al mismo tiempo, sólo es necesario definir el valor de **C2** para las tres combinaciones descritas en la tabla.

b) Utilizando lo desarrollado en el item anterior, diseñe un restador de dos números de cuatro trits cada uno. **(2 ptos.)**

Solución: Para construir un restador de trits utilizando un esquema similar al visto en clases para el restador de bits, es necesario que este funcione en complemento a 3, que es el análogo al complemento a 2 pero en ternario. Para esto, es necesario definir una compuerta que entregue el complemento del sustraendo (B), tal como el negador entrega el complemento de un número binario. La tabla para esta compuerta es la siguiente:

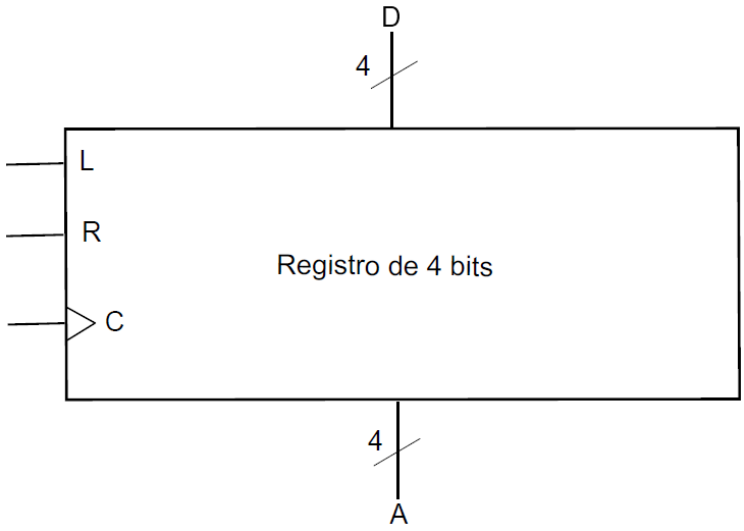
A	N
0	2
1	1
2	0

Teniendo definida la compuerta N, el diagrama para el restador de trits es el siguiente:

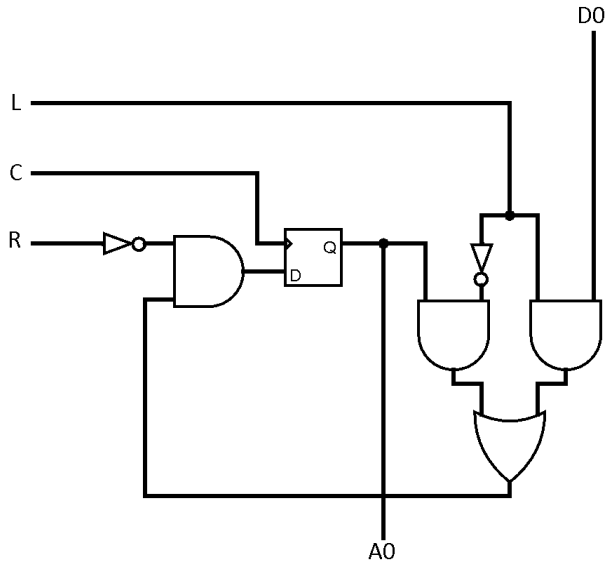


Pregunta 3

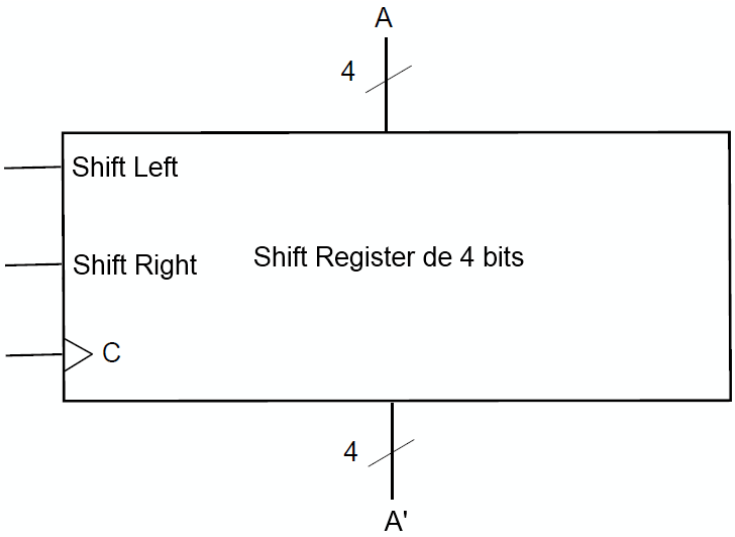
a) Implemente mediante compuertas lógicas y flip-flops tipo D, el registro de la figura, con señales de control (C), carga (Load) y reset (Reset), que funciona con flanco de subida. **(3 ptos.)**



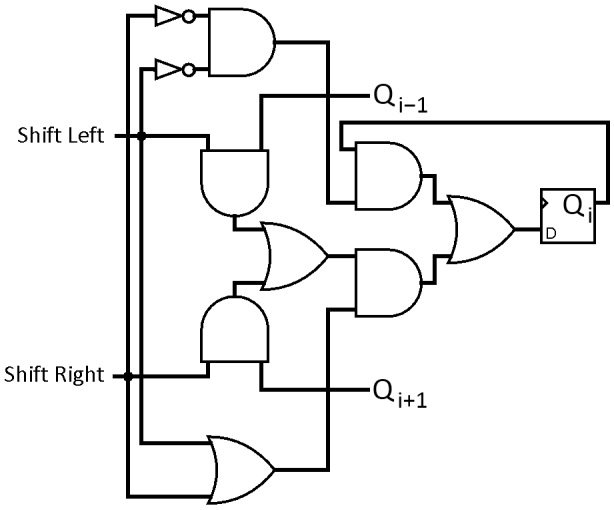
Solución: Por motivos de espacio, se presenta la solución para el bit menos significativo. Para el resto de los bits el diagrama es el mismo.



b) Implemente mediante compuertas lógicas y flip-flops tipo D, el shift-register de la figura. Este registro realiza shifting lógico con su contenido, de acuerdo a las señales Shift Left y Shift Right, cuando la señal C se encuentra en flanco de subida. **(3 ptos.)**



Solución: Por motivos de espacio, se presenta la solución para el bit i. Cuando $i = 0$ (bit menos significativo), $Q_{i-1} = Q_3$, mientras que si $i = 3$ (bit más significativo, $Q_{i+1} = Q_0$.



Pregunta 4

Dada la siguiente función, donde `var` representa un tipo básico desconocido:

```
var func(var a, var b)
{
    var c = 0, d = 0;
    var n = sizeof(var)*8;
    for (var i = n-1; i >= 0; i--)
    {
        d = d << 1;
        d(0) = a(i);
        if (d >= b)
        {
            c = c - b;
            c(i) = 1;
        }
    }
    return c;
}
```

Donde `x`(i) entrega el bit de `x` que está en la posición i. Suponga además que se tiene una memoria de 10 entradas, con palabras de 1 byte con los valores que se muestran a continuación:

Dirección	Palabra
0	$0x0A$
1	$0x00$
2	$0x01$
3	$0x00$
4	$0x12$
5	$0x05$
6	$0x10$
7	$0x00$
8	$0x00$
9	$0x06$

Calcule el valor de retorno de la función para los siguientes casos:

a) Las variables `a` y `b` son de tipo `byte`, `a` está almacenada a partir de la dirección 0 y `b` está almacenada a partir de la dirección 5. **(2 ptos.)**

b) Las variables `a` y `b` son de tipo `unsigned short` (entero sin signo de 2 bytes), `a` está almacenada a partir de la dirección 3 y `b` está almacenada a partir de la dirección 8. Ambas variables ocupan orden big endian. **(2 ptos.)**

c) Las variables `a` y `b` son de tipo `unsigned short`, `a` está almacenada a partir de la dirección 1 y `b` está almacenada a partir de la dirección 6. Ambas variables ocupan orden little endian. **(2 ptos.)**

Solución: El algoritmo copia el valor de la variable a en la variable d secuencialmente, desde los bits más significativos hasta los menos, mediante shifts a la izquierda de una posición. Esto significa que pone un bit en la posición 0 de d y luego lo mueve hacia la izquierda con shifts. El valor de retorno c se modifica sólo a partir del momento en que d es mayor que b. De ahí en adelante, por

cada iteración se le resta a c el valor de b. Como c parte en 0 y en los tres casos no tiene signo, esto es análogo a que c tome el valor $2^8 = 256$ cuando es del tipo *byte* y que tome el valor $2^16 = 65536$ cuando es *unsignedshort*. Además, junto con restar b, se pone en 1 el bit i de c.

a) $a = 10$ y $b = 5$, valor de retorno 247.

b) $a = 18$ y $b = 6$, valor de retorno 65525.

c) $a = 256$ y $b = 16$, valor de retorno 65471.

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



IIC2343 – Arquitectura de Computadores (II/2012)

Solución Interrogación 1

Pregunta 1

a) Describa un caso en que es mejor utilizar números en big-endian y otro en que sea mejor utilizarlos en little-endian. **(1 pto.)**

Solución: En una situación en que lleguen datos de manera secuencial, dependiendo del caso es más conveniente little-endian o big-endian. Si la operación a realizar con los datos es una suma, conviene que sea little-endian, ya que el carry se traspasa desde los bytes menos significativos a los más significativos. Por otro lado, si la operación a realizar es una división, es preferible usar big-endian, ya que la división se realiza desde los bytes más significativos a los menos significativos.

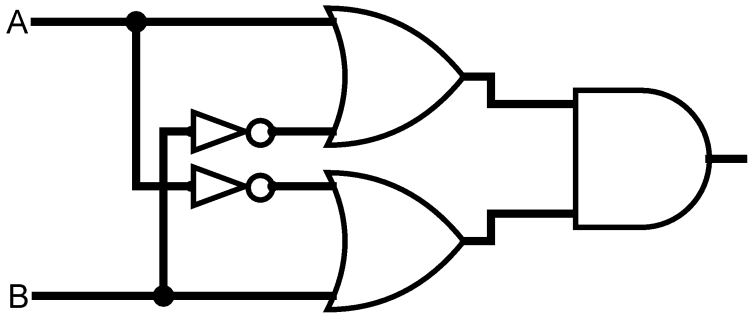
b) Escriba en formato float el número 16,375. Indique como se divide y que significa cada una de las partes del string de bits. **(1 pto.)**

Solución: En el formato float se asigna el primer bit al signo del significante, los siguientes 8 al exponente (desfasado en 128) y los restantes 23 al significante (normalizado). Luego, dado que $16,375 = 10000,011b = 1,0000011b \times 10b^{100b}$, el número en float se escribe de la siguiente manera: 0 10000011 00000110000000000000000

c) Implemente, utilizando sólo las compuertas lógicas **AND**, **OR** y **NOT**, el conectivo binario bicondicional ($\leftrightarrow$), que está definido por la siguiente tabla de verdad: **(1 pto.)**

A	B	A$\leftrightarrow$B
F	F	V
F	V	F
V	F	F
V	V	V

Solución: La tabla de verdad del conectivo bicondicional es analoga a la de la negación de un XOR, luego, se puede implementar de la siguiente manera con compuertas lógicas:



d) Un número entero de **16** bits en **little-endian** está almacenado en la posición 0x12 de una memoria RAM con palabras de 1 byte. Un programador extrae este dato de la memoria RAM y al no saber como interpretarlo prueba con un entero de **16** bits en **big-endian**. Si el número almacenado inicialmente era **255**, que valor obtuvo el programador? **(1 pto.)**

Solución: El número entero 255, se representa con 16 bits y little-endian cómo $0x00FF$. Luego, en big-endian, el número queda $0xFF00$, lo que corresponde al número -256 en decimal.

e) ¿Cuántas direcciones tiene una memoria RAM de 4.5 KB que utiliza palabras de 3 bytes? (1 KB = 1024 bytes) **(1 pto.)**

Solución: El tamaño total de la memoria es 4.5KB, que corresponde al producto entre el tamaño de las palabras y la cantidad de direcciones. Luego, la cantidad de direcciones de la memoria es $\frac{4{,}5\times1024}{3} = 1{,}5 \times 1024 = 1536$

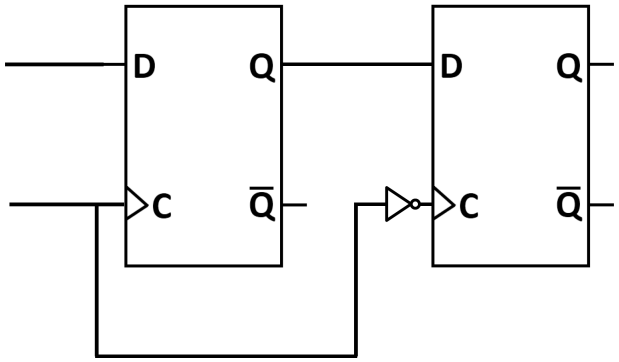
f) Un registro de 8 bits tiene almacenado el **número entero** 102. Que número se obtiene al realizar 2 operaciones **shift right** seguido de 3 operaciones **shift left**? **(1 pto.)**

Solución: El número 102 se escribe en binario como 01100110. Después de los 2 shift right, se obtiene 00011001, y después de los 3 shift left se obtiene finalmente 11001000, que es -56 en decimal.

Pregunta 2

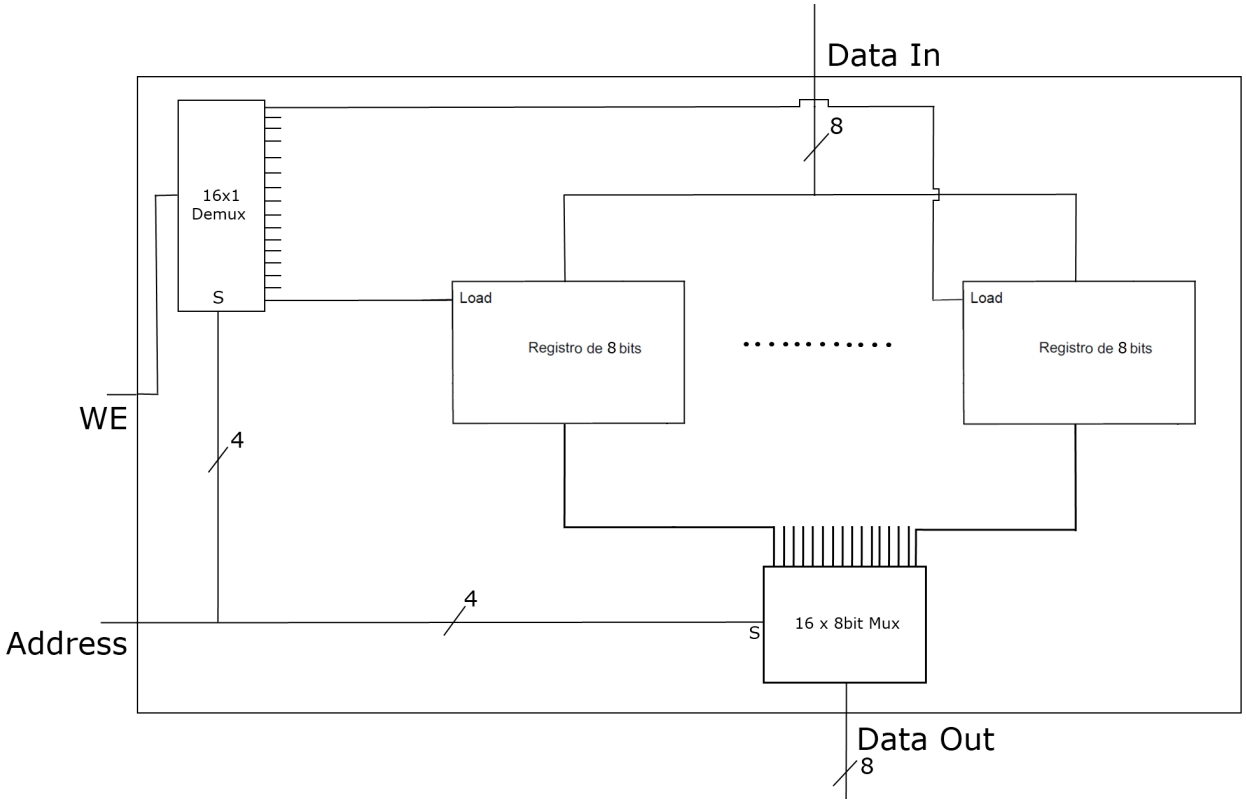
a) Implemente mediante compuertas lógicas, elementos de control y latches, un flip-flop tipo **D** que funcione con flanco de **bajada**. **(3 ptos.)**

Solución:



b) Implemente mediante compuertas lógicas, elementos de control y flip-flops, una memoria RAM de 16 palabras de 1 byte. **(3 ptos.)**

Solución:



Pregunta 3

Para los siguientes ejercicios, escriba su respuesta en el lenguaje de programación que más le acomode. Deje por escrito claramente cual es su elección.

a) Escriba un programa para convertir un número **natural** en base 12 a su valor en base decimal. Los digitos de un número en base 12 son: 0,1,2,3,4,5,6,7,8,9,A,B. Asuma que los números están almacenados en strings. **(3 ptos.)**

Solución:

```
public static int base12aBase10(char[] num)
{
    int res = 0;
    for (int i = num.length-1, factorBase = 1; i >= 0; i--, factorBase *=12)
    {
        int valor;
        char base12 = num[i];
        if (base12 == 'A')
        {
            valor = 10;
        }
        else if (base12 == 'B')
        {
            valor = 11;
        }
        else
        {
            valor = Character.getNumericValue(base12);
        }
        res += valor*factorBase;
    }
    return res;
```

b) Escriba un programa para obtener el inverso aditivo de un número en base 12, sin utilizar otra base como paso intermedio. Asuma que los números están almacenados en strings. **(2 ptos.)**

Solución:

```
public static char[] inversoAditivoBase12(char[] num)
{
    char[] res = new char[num.length];
    for (int i = 0; i < res.length; i+)
    {
        char c = num[i];
        if (c == '0')
        {
            res[i] = 'B';
        }
        else if (c == '1')
        {
            res[i] = 'A';
        }
        .
        .
        .
        .
        else if (c == 'A')
        {
            res[i] = '1';
        }
        else
        {
            res[i] = '0';
        }
    }
    //asumimos la existencia de un metodo que calcula
    //la suma de dos numeros en base 12
    res = sumaBase12(res, 1);
    return res;
```

c) Escriba un programa para obtener la dirección de memoria de un elemento de una matriz almacenada en orden de columnas. Asuma que el programa recibe los siguientes parámetros de entrada: la matriz, la fila del elemento y la columna del elemento. Puede utilizar los operadores para indexación vistos en clases. **(1 ptos.)**

Solución:

```
public static int obtenerDireccion(Matriz A, int i, int j)
{
    return dir(A) + j*A.filas*sizeof(A[i][j]) + i*sizeof(A[i][j]);
}
```

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

IIC2343 – Arquitectura de Computadores (I/2013)

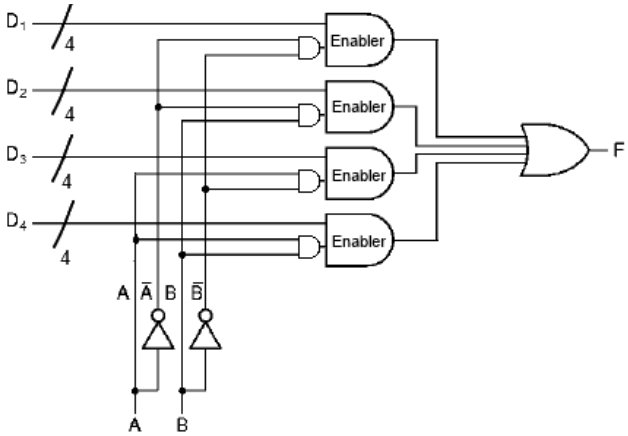
Interrogación 1

Pregunta 1

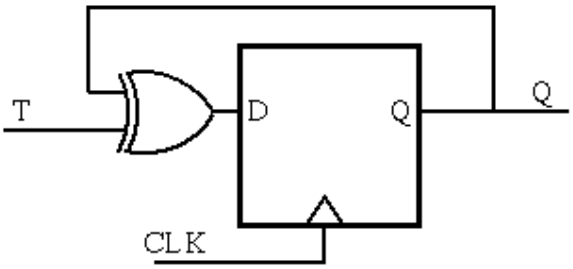
a) ¿Cuál es el valor del número 11000001100000000000000000000000, representado mediante el tipo de dato **float**? **(1 pto.)**
Solución: A simple vista se puede apreciar que el significante es cero y el signo es negativo, por lo que el número tiene la forma $-1 \times 2^{exp-127}$. Luego, como el exponente es igual a 131, el valor del número es $-1 \times 2^{131-127} = -1 \times 2^4 = -16$.

b) ¿Cuál es el mayor número positivo que puede representarse mediante un complemento a 2 de 11 bits? **(1 pto.) (1 pto.)**
Solución: Si se tienen 11 bits disponible, el mayor número positivo será $01111111111 = 2^{10}-1 = 1023$.

c) Diseñe un Mux de 4 entradas de 4 bits cada una. **(1 pto.)**
Solución:



d) Diseñe usando compuertas lógicas y flip-flops **D**, un flip-flop **T**. El comportamiento de este flip-flop consiste en invertir el valor de su salida **Q** si su señal de entrada **T** está en 1 y la señal de control **C** pasa de 0 a 1 (flanco de subida). En cualquier otro caso, la salida **Q** se mantiene igual. **(1 pto.)**
Solución:



e) ¿Cuántas entradas y salidas tiene un display controller de una calculadora que utiliza números de 4 bytes? **(1 pto.)**
Solución: Un display de siete segmentos puede mostrar númerosd de hasta 4 bits. Luego, necesitamos 8 displays para mostrar números de 4 bytes, por lo tanto, el display controller necesita 56 salidas y 32 entradas.

f) ¿Qué secuencia de números enteros es generada al realizar 8 operaciones **rotate left** a un registro de 8 bits que inicialmente almacena el número 79? **(1 pto.)**
Solución: La secuencia generada es la siguiente: -98, 61, 122, -12, -23, -45, -89, 79.

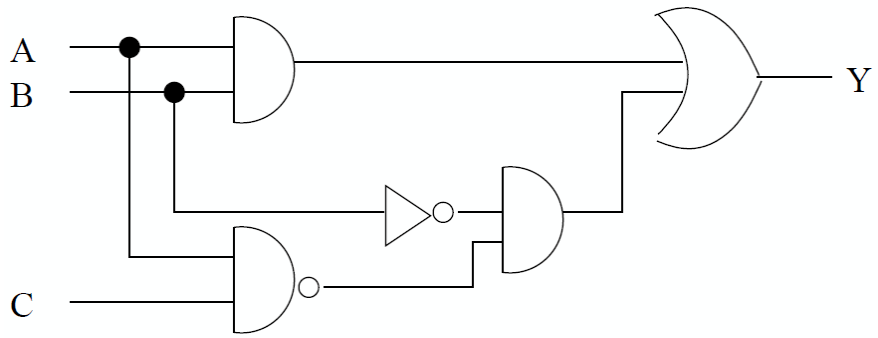
Pregunta 2

a) Considere la siguiente fórmula en lógica booleana:

$$Y = (A \wedge B) \vee \neg(A \wedge C) \wedge \neg B$$

i) Escriba la tabla de verdad correspondiente a la función anterior y diseñe un circuito lógico que la implemente. **(1 pto.)**
Solución: Esta es una de muchas posibles soluciones, dependiendo de la distribución (factorización) utilizada en la fórmula

A	B	C	Y
F	F	F	F
F	F	V	F
F	V	F	V
F	V	V	V
V	F	F	F
V	F	V	V
V	V	F	F
V	V	V	F



ii) A partir de la tabla de verdad, re-escriba la función como conjuntos de **ANDs** de las variables (o sus negaciones), combinados mediante **ORs**. **(1 pto.)**
Hint: $A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$
Solución: De la tabla de verdad, basta tomar los casos en que el valor de Y es V de la siguiente manera:

$$Y = (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge B \wedge C) \vee (A \wedge \neg B \wedge C)$$

b) Si dos números enteros (con signo), A y B, son sumados (restados) mediante un **adder** (**subtractor**) de n bits, es posible que el resultado sea incorrecto: la suma de dos números positivos puede ser un número negativo, la suma de dos números negativos puede ser un número positivo, etc. Estos errores son llamados *overflow*.
Para detectar automáticamente estas situaciones, se define una variable binaria **overflow** que toma el valor 1 cuando ocurre el error y 0 cuando no ocurre. Esta variable depende de otras

cuatro variables binarias: A_{n-1} (bit más significativo de A), B_{n-1} (bit más significativo de B), S_{n-1} (bit más significativo del resultado de la operación), Op (operación: suma o resta).
Escriba la tabla de verdad para la variable **overflow**, en base a los valores de las otras cuatro variables. **(2 ptos.)**
Solución: Consideraremos los casos de suma y resta de manera separada. Primero, para la suma (Op) tebemos tres casos:

- si los dos números son de signo opuesto, NO puede existir overflow.
- si los dos números son positivos ($A_{n-1} = 0$, $B_{n-1} = 0$) , sólo existe overflow el resultado es negativo ($S_{n-1} = 1$).
- si los dos números son negativos, ($A_{n-1} = 1$, $B_{n-1} = 1$), sólo existe overflow si el resultado no es negativo ($S_{n-1} = 0$).

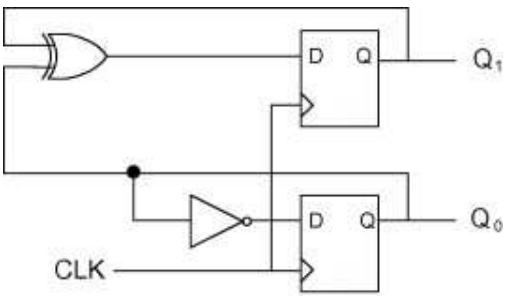
Para la resta el análisis es similar:

- si los dos números son del mismo signo, NO puede existir overflow.
- si A es positivo ($A_{n-1} = 0$) y B negativo ($B_{n-1} = 1$), sólo existe overflow cunado el resultado es negativo ($S_{n-1} = 1$).
- si A es negativo ($A_{n-1} = 1$) y B positivo ($B_{n-1} = 0$), sólo existe overflow si el resultado es positivo ($S_{n-1} = 0$).

Luego, la tabla toma la siguiente forma:

Op	A_{n-1}	B_{n-1}	S_{n-1}	overflow
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

c) Diseñe utilizando todos los elementos de circuitos lógicos vistos en clases, un contador secuencial de 2 bits, que se incrementa con cada flanco de subida de la señal de control. **(2 ptos.)**
Solución:



Pregunta 3

Para los siguientes ejercicios, escriba su respuesta en el lenguaje de programación que más le acomode. Deje por escrito claramente cual es su elección.

a) Escriba el código para el método `int[] sumaBinarios(int[] num1, int[] num2)`. Este método recibe dos números binarios de igual largo y retorna la suma. Los número se asumen que serán positivos y que la suma no sobrepasará la cantidad de bits de los sumandos. Los números binarios son representados como arreglos de enteros y cada posición del arreglo podrá tomar un valor 0 o 1 lo cual debe cumplirse también en el retorno. **(3ptos.)**
Solución:

```
public static int[] sumaBinarios(int[] num1, int[] num2)
{
    int[] resultado = new int[num1.length];
    int carry = 0;
    for (int i = num1.Length - 1; i >= 0; i--)
    {
        // Caso 3 unos.
        if (carry == 1 && num1[i] == 1 && num2[i] == 1)
        {
            carry = 1;
            resultado[i] = 1;
        }
        // Caso 2 unos.
        else if ((carry == 1 && num1[i] == 1) ||
                 (carry == 1 && num2[i] == 1) ||
                 (num1[i] == 1 && num2[i] == 1))
        {
            carry = 1;
            resultado[i] = 0;
        }
        // Caso 3 ceros.
        else if (num1[i] == 0 && num2[i] == 0 && carry == 0)
        {
            carry = 0;
            resultado[i] = 0;
        }
        // Caso 1 uno.
        else
        {
            carry = 0;
            resultado[i] = 1;
        }
    }
    return resultado;
}
```

b) Escriba el código para el método `float binarioAFloat(int[] single)`, el cual convierte un número binario (representado como arreglo de enteros) en su valor tipo `float`. El número binario representa un número del tipo «single precision floating point» (float de 32 bits) según las características del estándar IEEE754 visto en clases. En su conversión **no** debe considerar los casos especiales mencionados en los apuntes y en clases. **(3ptos.)**
Solución:

```
public static float BinarioAFloat(int[] single)
{
    //Posicion 0: signo
    //Posicion 1-8 = exponente
    //Posicion 9-31 = significante
    int exp = -127;
    float significante = 1;
    float resultado;
    int signo = single[0]*-1;

    for (int i = 7; i >= 0; i--)
    {
        exp += single[8-i] * (int)Math.pow(2, i);
    }
    for (int i = 9; i < 32; i++)
    {
        significante += single[i]*(float)Math.pow(2, -(i - 8));
    }
    resultado = significante * (float)Math.pow(2, exp);
    if (signo == -1)
    {
        resultado *= signo;
    }
    return resultado;
}
```

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

IIC2343 – Arquitectura de Computadores (II/2013)

Interrogación 1 - Pauta

Pregunta 1 (2,0 ptos)

El siguiente programa escrito en Java implementa un algoritmo de multiplicación de dos números en complemento a 2. Traduzca el programa al assembly del computador básico, cumpliendo lo siguiente:

- Su programa debe ser una traducción directa del programa en Java, es decir, además de traducir la funcionalidad (multiplicación) debe hacerlo implementando el mismo algoritmo descrito en el programa.
- Su programa debe definir (al menos) las variables `a`, `b` que se ocuparán como operandos de la multiplicación, y la variable `c` que almacenará el resultado.

Pregunta 1.1 (1,0 ptos)

```
public static void main(String[] args)
{
  //Declaracion de variables
  byte a = 12;
  byte b = 3;
  byte c = 0;
  //Loop principal
  while(true)
  {
     //And
     byte d = (byte)(b & 1);
     //If y acumulacion en c
     if(d != 0)
       c += a;
     //Shift right
     b >>= 1;
     //If y break
     if(b == 0)
       break;
     //Shift left
     a <<= 1;
  }
}
```

Solución

```
;Declaracion de variables
DATA:
    a       12
    b       3
    c       0

CODE:

    start:  MOV A, (b)    ;And
            AND A, 1      ;And
            CMP A, 0      ;If y acumulacion c
            JEQ else1     ;If y acumulacion c
            MOV A, (c)    ;If y acumulacion c
            MOV B, (a)    ;If y acumulacion c
            ADD A, B      ;If y acumulacion c
            MOV (c), A    ;If y acumulacion c
    else1:  MOV A, (b)    ;Shift right
            SHR A, A      ;Shift right
            MOV (b), A    ;Shift right
            CMP A, 0      ;If y break
            JNE else2     ;If y break
            JMP end       ;If y break
    else2:  MOV A, (a)    ;Shift left
            SHL A, A      ;Shift left
            MOV (a), A    ;Shift left
            JMP start     ;Loop principal
    end:
```

Puntajes

- Declaracion de variables: 0,1 ptos
- Loop principal: 0,1 ptos
- And: 0,1 ptos
- If y acummulacion c: 0,2 ptos
- Shift right: 0,15 ptos
- If y break: 0,2 ptos
- Shift left: 0,15 ptos

Pregunta 1.2 (1,0 ptos)

Para el siguiente programa escrito en el assembly computador básico, especifique:

- Los valores de los registros `A`, `B`, de las variables `var1`, `var2`, `var3` y del arreglo `arr1` cuando el programa llega a la instrucción asociada al label `label0` (antes que se ejecute esa instrucción).
- Los valores de los registros `A`, `B` y de las variables `var1`, `var2` la **segunda vez** que el programa llega a la instrucción asociada al label `label1` (antes que se ejecute esa instrucción).
- Indique si el programa llega al label `end` o no. En caso de que si llegue especifique los valores de los registros `A`, `B` y de las variables `var1`, `var2` al finalizar el programa.

```
DATA:
    var1    2
    var2    5
    var3    10
    arr1    0
            1
            2

CODE:
            MOV A, (var3)
            MOV B, (var1)
            SUB A, B
            MOV (var2), A
            MOV B, arr1
            MOV A, 2
            ADD A, B
            MOV B, A
            MOV A, (B)
            MOV (var1), A

    label0: MOV A,1
            MOV B,2
    label1: ADD A,1
            ADD B,1
            MOV (var1), B
            MOV (var2), A
            CMP A,B
            JGE label2
            MOV (var1), A
            MOV (var2), B
            JMP label3
    label2: MOV A,4
            CMP A,B
            JLT label1
            ADD B,A
            MOV (var1), A
            MOV (var2), A
            JMP end
    label3: ADD A,B
            JMP label1
    end:
```

Solucion y Puntajes

- **label0:**
  - A = 2 (0,1 ptos)
  - B = arr1 + 2 (0,1 ptos)
  - var1 = 2 (0,025 ptos)
  - var2 = 8 (0,025 ptos)
  - var3 = 10 (0,025 ptos)
  - arr = [0,1,2] (0,025 ptos)
- **label1:**
  - A = 5 (0,1 ptos)
  - B = 3 (0,1 ptos)
  - var1 = 2 (0,05 ptos)
  - var2 = 3 (0,05 ptos)
- **end:**
  - A = 4 (0,1 ptos)
  - B = 8 (0,1 ptos)
  - var1 = 4 (0,1 ptos)
  - var2 = 4 (0,1 ptos)

Pregunta 2 (2,0 ptos)

Pregunta 2.1 (1,0 ptos)

Para un determinado programa la *secuencia efectiva* de instrucciones corresponde a la secuencia de todas las instrucciones que se ejecutaron desde que comenzó hasta que terminó el programa, incluyendo las instrucciones que se repitieron por saltos o llamados a subrutinas.

A modo de ejemplo, para el siguiente programa:

```
CODE:
            MOV A, 0    //Instruccion 0
            MOV B, 2    //Instruccion 1
label1:     CMP A,B     //Instruccion 2
            JGE end     //Instruccion 3
            ADD A,1     //Instruccion 4
            JMP label1  //Instruccion 5
end:
```

La secuencia efectiva de instrucciones es:

```
MOV A, 0    //Instruccion 0
MOV B, 2    //Instruccion 1
CMP A,B     //Instruccion 2
JGE end     //Instruccion 3
ADD A,1     //Instruccion 4
JMP label1  //Instruccion 5
CMP A,B     //Instruccion 2
JGE end     //Instruccion 3
ADD A,1     //Instruccion 4
JMP label1  //Instruccion 5
CMP A,B     //Instruccion 2
JGE end     //Instruccion 3
```

Realice las modificaciones necesarias al computador básico de manera que se pueda implementar la instrucción `NUM_INSTRUCTIONS (var1)` la cual almacena en la variable `var1` la cantidad de instrucciones que efectivamente se ejecutaron hasta que se llamó a esta instrucción (no incluyendo a esta instrucción).

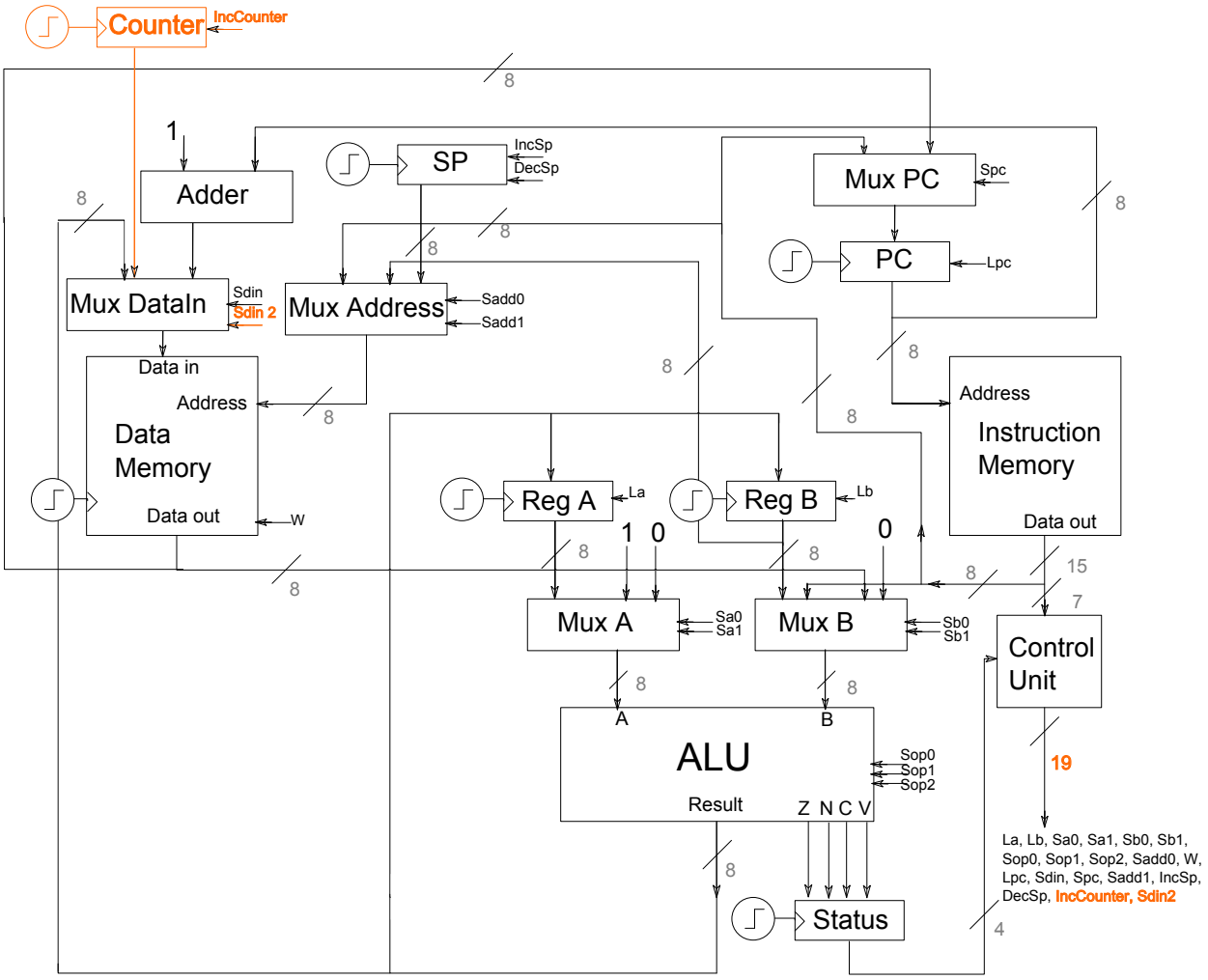
A modo de ejemplo, para el programa anterior, si se llama a esta instrucción luego del label `end` se debería almacenar el valor 12 en la variable `var1` dado que se ejecutaron 12 instrucciones.

Importante 1: Para esta pregunta puede asumir que todas las instrucciones se ejecutan en un ciclo del clock.

Importante 2: No puede agregar memorias para resolver esta pregunta.

Solución

Una posible solución es la siguiente:



- Se agrega un contador al computador con una señal de control IncCounter, que hace que el contador se increment si está habilitada.
- Se aumenta el tamaño del Mux de data in de la memoria de datos, y se agrega la salida del contador como entrada posible de este Mux.
- A todas las instrucciones se les agrega la señal de control IncCounter con valor 1, para que se incremente el contador luego de cada llamado, exceptuando a la nueva instrucción `NUM_INSTRUCTIONS (var1)` la cual no activa esta señal.
- Las señales de control relevantes de la nueva instrucción `NUM_INSTRUCTIONS (var1)` son:
  - La señal de selección del MUX Data In se elige para que se escoja el valor del contador como entrada a la memoria (Sdin = 10 por ejemplo).
  - Como dirección para la memoria de datos se elige Saddress = Literal.
  - La señal IncCounter se setea en 0.
  - Todas las otras señales de Load o Incremento/Decremento se setean en 0.

Puntaje

- Lograr que de alguna forma se cuenten todas las instrucciones que se ejecutaron, ya sea a medida que se ejecutan o después. No es válido ocupar el Program Counter, porque este solo incrementa si no hay instrucciones de salto, si las hay no tendrá almacenado el número total de instrucciones (0,5 ptos).
- Lograr que la instrucción `NUM_INSTRUCTIONS (var1)` configure bien la memoria:
  - Agrandar mux de data in para que acepte una entrada más, y seleccionar esta entrada que debería tener el número de instrucciones. (0,2 ptos)
  - Setear el mux de address para que se elija como dirección el literal de la instrucción. (0,1 ptos)
  - Setear la memoria en modo escritura (W = 1). (0,1 ptos)
  - Asegurarse que todo el resto de lase señales de carga esten en 0 (LoadA, LoadB, LoadPC, IncSP, DecSP).(0,1 ptos)
- Descuento si las modificaciones rompen alguna de las instrucciones existentes (-0,3 ptos).

Pregunta 2.2 (1,0 ptos)

Uno de los modos de direccionamiento existentes en el computador básico es el **direccionamiento directo** el cual, por ejemplo, se ocupa al usar la instrucción `MOV A, (var1)`. Esta instrucción carga en el registro A el valor que **directamente** está ubicado en la dirección de memoria datos asociada a `var1`. A modo de ejemplo, en la tabla de memoria que se encuentra más abajo, si la dirección asociada a `var1` fuese 2, al ejecutar la instrucción `MOV A, (var1)`, el registro A quedaría con el valor 5, o sea el valor de la palabra de memoria ubicado en la dirección 2.

Dirección	Palabra
0	5
1	4
2	5
3	0
4	0
5	7
6	1
7	1

Se desea agregar al computador básico la capacidad de realizar **direccionamiento indirecto**, en particular usted debe permitir que el computador básico soporte la instrucción `MOV A, ((var1))`. Esta instrucción realiza lo siguiente:

1. Obtiene la palabra almacenada en la dirección asociada a `var1`. Siguiendo el ejemplo anterior, suponiendo que la dirección asociada a `var1` es 2, el valor que se obtiene sería 5
2. Ocupa este valor obtenido para **volver a direccionar la memoria de datos**, y obtener una nueva palabra de memoria. En el ejemplo el valor 5 se ocupa como dirección, y se obtiene el valor que se encuentra en la dirección 5 de la memoria, o sea 7.

3. Se almacena este último valor en el registro A. En el ejemplo A = 7.

Solución

Una posible solución es implementar esta instrucción sin modificar el hardware. La siguiente secuencia de instrucciones implementa la instrucción:

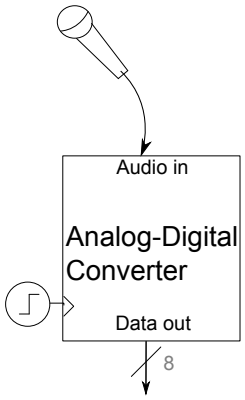
```
PUSH B
MOV B, (var1)
MOV A, (B)
POP B
```

Puntaje

- Lograr la funcionalidad asociada a la instrucción (direccionamiento indirecto), ya sea sin modificaciones al computador, o agregando alguna modificación que sea util (1,0 ptos).
- Descuento si se implementa por software, pero no se conserva el estado del computador, por ejemplo si se ocupa el registro B, pero no se hace un backup y restoración previa, o si se ocupa la memoria de datos para escribir algun valor potencialmente borrando lo que había antes (-0.5 ptos).
- Descuento si las modificaciones rompen alguna de las instrucciones existentes (-0,3 ptos).

Pregunta 3 (2,0 ptos)

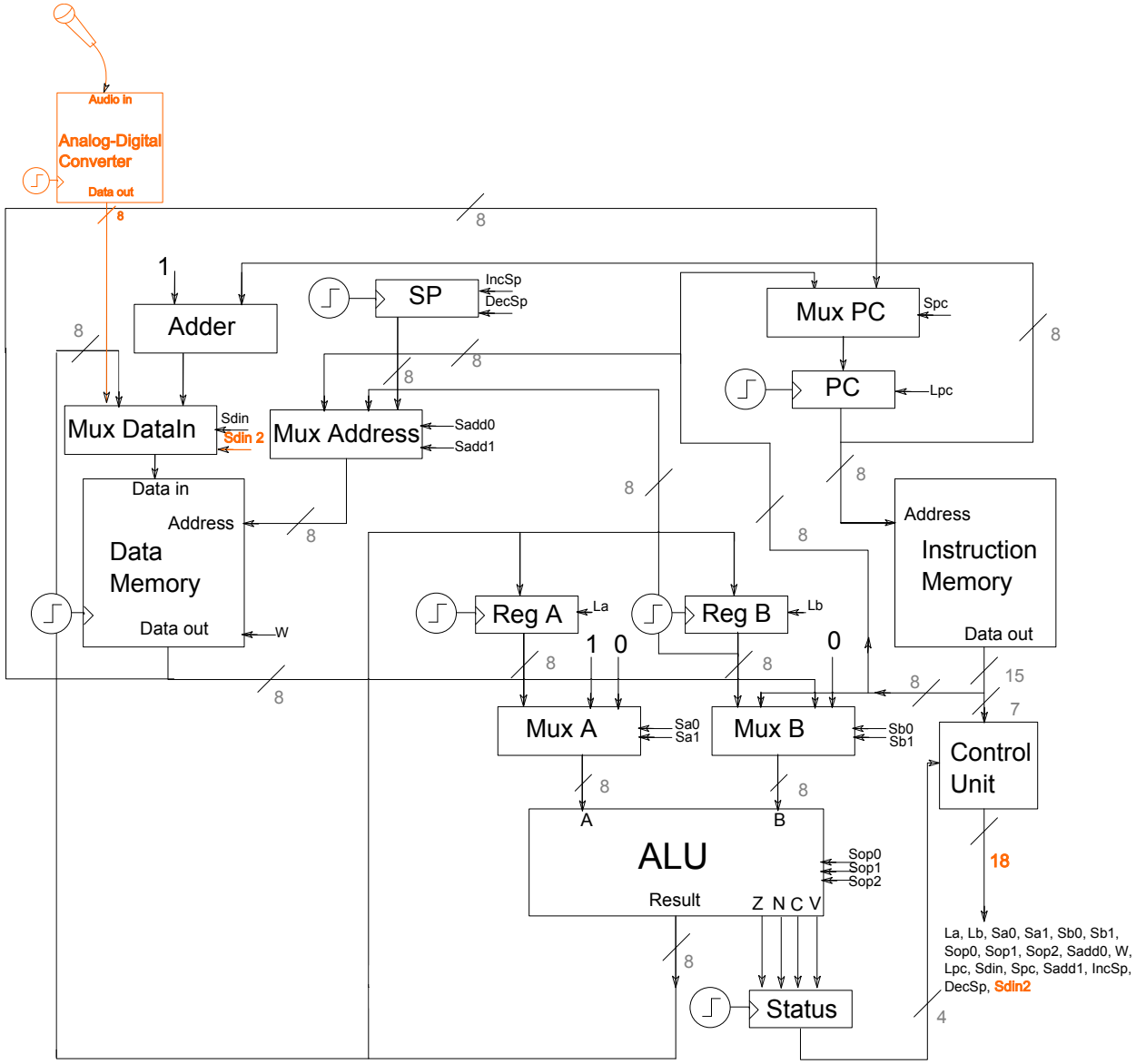
Se requiere agregar al computador básico la capacidad de procesar y almacenar audio desde un micrófono. Para esto se cuenta con un conversor análogo-digital, el cual recibe como entrada una señal continua que representa las variaciones de voltaje captadas por el micrófono y entrega como salida muestras de 8 bits cada cierto tiempo T (período de muestreo), de manera automática. El conversor está contectado al mismo clock del computador básico y se sabe que el período de muestreo T es 8 veces el tiempo que toma un ciclo del clock, lo que permite mantener sincronización. A continuación se muestra el diagrama del conversor análogo-digital:



1. Modifique el computador básico agregando un conversor análogo-digital de manera de que permita almacenar la muestra en memoria. Para esto debe agregar la instruccion `READ_ADC (B)`, la cual almacena una muestra del conversor análogo-digital, en la dirección de memoria de datos apuntada por el valor almacenado en el registro `B`. Conecte el conversor al computador básico de manera de que se pueda ejecutar esta instrucción e indique que señales de control se deben activar para ejecutar la instrucción.

2. Escriba un programa en el assembly del computador básico que implemente un loop infinito que vaya almacenando de manera sucesiva las muestras que van llegando desde el conversor análogo-digital en el arreglo almacenado en la dirección memoria apuntada por el label `vector`. Asegurese de obtener todas las muestras y no repetir muestras.

Solución



La instrucción `READ_ADC (B)` configura la memoria para que la entrada de datos sea el ADC y la dirección el registro B.

```
        MOV B, vector    ; Inicializar B con posicion inicial del arreglo vector
loop:   READ_ADC (B)     ; Leer muestra
        INC B            ; Incrementar en 1 posicion del vector
        CMP A,B          ; Ejecutar 5 instrucciones que no hagan nada util para
        CMP A,B          ; sincronizar programa con el periodo de muestreo
        CMP A,B
        CMP A,B
        CMP A,B
        JMP loop         ; Loop inifinto
```

Puntaje

- Conectar ADC a mux de entrada de datos de la memoria (0.5 ptos).
- Lograr que la instrucción `READ_ADC (B)` configure bien la memoria:
  - Agrandar mux de data in para que acepte una entrada más, y seleccionar esta entrada que debería tener el ADC. (0,2 ptos)
  - Setear el mux de address para que se elija como dirección el registro B. (0,1 ptos)
  - Setear la memoria en modo escritura (W = 1). (0,1 ptos)
  - Asegurarse que todo el resto de lase señales de carga esten en 0 (LoadA, LoadB, LoadPC, IncSP, DecSP).(0,1 ptos)
- El programa que lee las muestras debe cumplir con:
  - Inicializar B con posicion inicial del arreglo vector (0.1 ptos)
  - Leer muestra llamando a instrucción `READ_ADC (B)` (0.2 ptos)
  - Incrementar en 1 posicion del vector (0.1 ptos)
  - Ejecutar 5 instrucciones que no hagan nada util para sincronizar programa con el periodo de muestreo (0.5 ptos)
  - Loop inifinto (0.1 ptos)
- Descuento si las modificaciones rompen alguna de las instrucciones existentes (-0,3 ptos).

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
16 de Mayo de 2014
IIC2343-1 Arquitectura de Computadores

Pauta P1

1. Realice las siguientes operaciones utilizando aritmética de complemento de 2, utilizando 6 bits (5 bits + 1 bit de signo). Indicar si se produce overflow.

   a) $31 + 5$.

   Tenemos que:

   $$31 = 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 11111b = 011111_{comp_2}$$

   $$5 = 0 \cdot 2^4 + 0 \cdot 2^3 + 2^2 + 0 \cdot 2^1 + 2^0 = 00101b = 000101_{comp_2}$$

   Luego:

   $$011111_{comp_2} + 000101_{comp_2} = 100100_{comp_2}$$

   Se produce overflow porque obtuvimos un resultado negativo al sumar dos números positivos.

   b) $-15 - 17$.

   Tenemos que:

   $$15 = 0 \cdot 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 01111b = 001111_{comp_2} \Longrightarrow -15 = 110001_{comp_2}$$

   $$17 = 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 2^0 = 010001_{comp_2} \Longrightarrow -17 = 101111_{comp_2}$$

   Luego:

   $$-15 - 17 = 110001_{comp_2} + 101111_{comp_2} = 1100000_{comp_2} = 100000_{comp_2}(6bits)$$

   Así, no hay overflow.

c) $-7 + 27$.

Tenemos que:

$$7 = 000111_{comp_2} \Longrightarrow -7 = 111001_{comp_2}$$
$$27 = 011011_{comp_2}$$

Luego,

$$-7 + 20 = 111001_{comp_2} + 011011_{comp_2} = 1010100_{comp_2} = 010100_{comp_2}(6bits)$$

Así, no hay overflow.

Cada punto tiene el mismo valor y se utilizará la siguiente distribución:

- Cada transformación a complemento 2 vale 30 %.
- La realización de la operación vale 30 %.
- Identificación de overflow vale 10 %.

2. Considere el estándar IEEE 754 para la representación de números de punto flotante de precisión simple (32 bits). ¿Cuántos números se pueden representar con este estándar, sin considerar $\pm\infty$ ni NaN?

Según el estándar, podemos representar un total de 2^{32} combinaciones posibles. De esas combinaciones, tenemos que 2 corresponden a la representación de los infinitos y $2^{23} - 1$ corresponden a NaN (2^{23} porque la diferenciación se produce con los bits del significante, considerando que alguno debe ser distinto de 0, y por eso se le resta el caso en que todos son 0's). Además, considerar que existen dos representaciones del 0, por lo que quitamos una de ellas, obteniendo así una cantidad de $2^{32} - 2 - (2^{23} - 1) - 1 = 4286578686$ combinaciones posibles.

Distribución:

- Combinaciones totales 20 %.
- Infinitos 20 %.
- NaN 20 %.
- 0's 20 %.
- Resultado Final con Distribución de bits 20 %.

3. Escriba en formato float el número -48. Indique cómo se compone y qué significa cada una de las partes de la secuencia de bits.

   El formato float, según el estándar IEEE 754, se representa con 32 bits, divididos de la siguiente forma:

   - 1 bit de signo
   - 8 bits para el exponente (desplazado en +127)
   - 23 bits para el significante normalizado

   Así, la representación se vería de la siguiente forma:

Signo	Exponente	Significante
1 bit	8 bits	23 bits

   Por lo tanto, tenemos que:

   - Dado que el número es negativo, el bit de signo valdrá 1.
   - Tenemos que $48 = 110000b$. A continuación, debemos normalizar el número binario. Tenemos que $110000b = 1{,}10000 \cdot 2^5 b$. Por lo tanto, se tie que el $significante = 10000000000000000000000$.
   - Finalmente, debemos determinar el exponente. Del procedimiento anterior, obtuvimos que el exponente es 5. Pero es necesario desplazarlo en +127. Es decir, $exp = 5 + 127 = 132 = 10000100b$.

   Así, juntando todo, tenemos que la representación sería:

S	Exp	Significante
1	10000100	10000000000000000000000

   Distribución:

   - Bit signo 10 %.
   - Significante 30 %.
   - Exponente 30 %.
   - Resultado Final con Distribución de bits 30 %.

4. Un codificador de prioridad (priority encoder) tiene 2^N entradas, una salida Q de N-bits y una salida NONE de 1 bit. La salida Q indica la posición del bit más significativo del input que es 1, o vale 0 si es que ninguno de los bits del input es 1. La salida NONE vale 1 si ninguno de los inputs es 1. Utilizando las compuertas lógicas básicas, diseñe un 2-input (N = 1) priority encoder con inputs A_1 y A_0, en donde A_1 es el input más significativo, y outputs Q y NONE. De esta forma, si es que $A_1 = 0$ y $A_0 = 1$, entonces tanto Q como NONE debería ser 0, pero si $A_1 = 1$, entonces Q debería valer 1.

   Tenemos que:

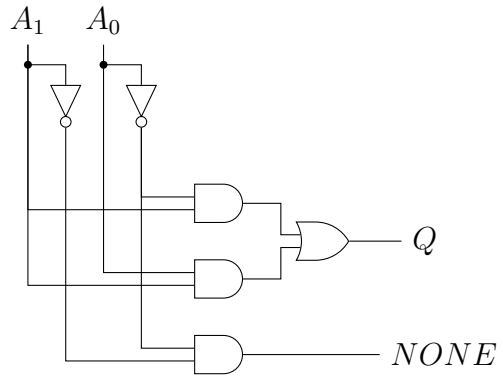
   - Tabla de verdad:

A_1	A_0	Q	NONE
0	0	0	1
0	1	0	0
1	0	1	0
1	1	1	0

   - Ecuaciones booleanas:

$$Q = A_1\overline{A_0} + A_1A_0$$
$$NONE = \overline{A_1A_0}$$

   - Dibujo del circuito:



   Distribución:

   - Tabla de verdad 33 %.
   - Ecuaciones booleanas 33 %.
   - Diagrama 33 %.

Pauta P2

Modifique la microarquitectura de la CPU Básica para poder realizar lo siguiente:

a Direccionamiento indirecto por registro base + offset

b Además del direccionamiento anterior, agregar direccionamiento indirecto por registro base + registro índice

c Agregar instrucción `MUL A, B`, la cual multiplica las entradas del mux A y mux B.

d Condition code I, que indica cuándo un número es impar. Indique cómo se debe procesar el output de la ALU para obtener este condition code. Este condition code debe incluirse en el Status Register.

Para cada uno de estos casos se deben señalar los bits de control, buses, componentes digitales y cualquier otra modificación que estime pertinente.

Respuesta

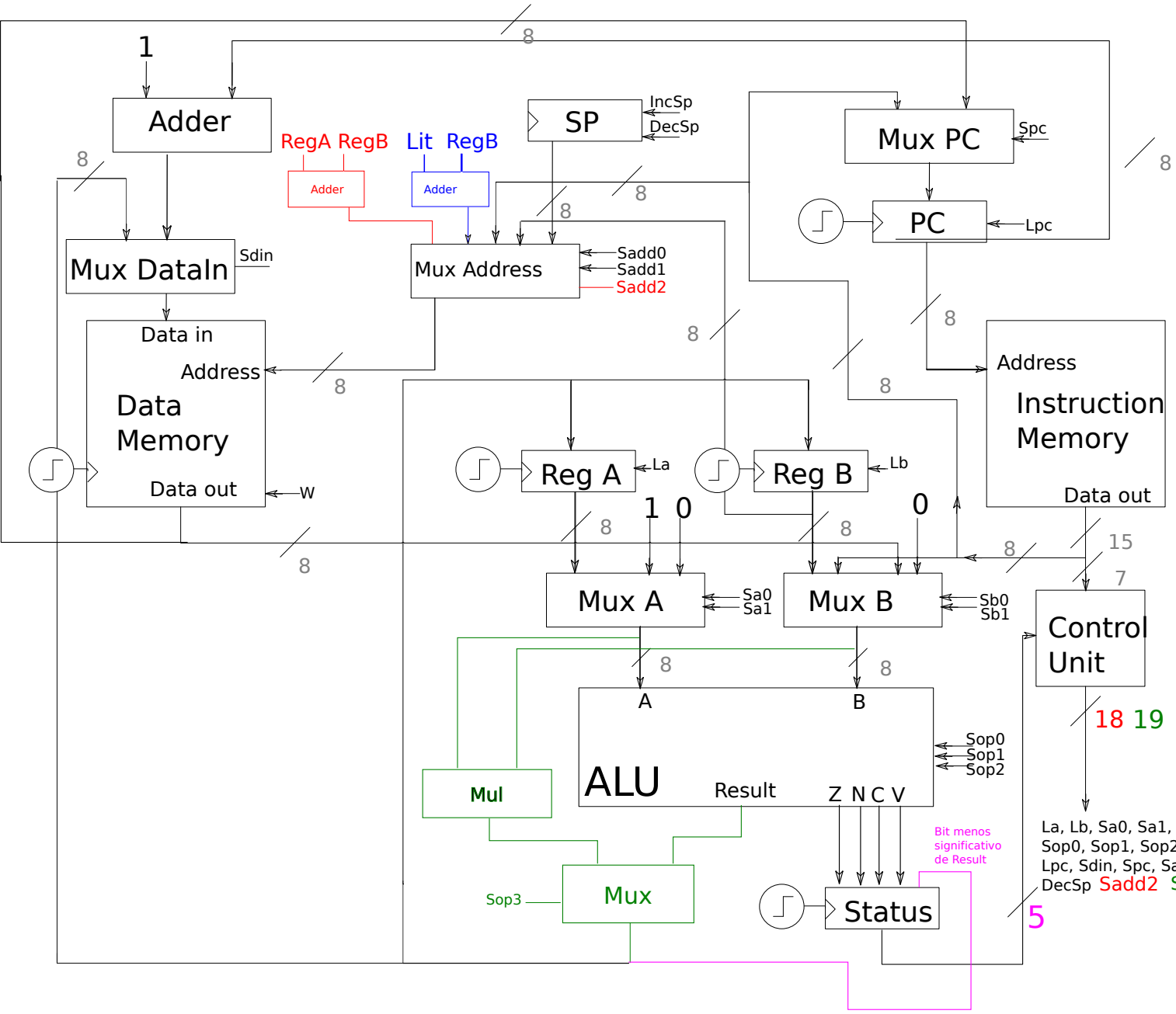
Revisar PDF adjunto, se utilizaron los siguientes colores para detallar los cambios:

a Azul

b Rojo

c Verde

d Morado


1
8
Adder
RegA RegB
Adder
Lit RegB
Adder
SP
IncSp
DecSp
Mux PC
Spc
8
8
8
8
PC
Lpc
Mux DataIn
Sdin
Mux Address
Sadd0
Sadd1
Sadd2
8
8
Data in
Address
Data
Memory
Data out
W
8
8
8
Address
Instruction
Memory
Data out
Reg A
La
Reg B
Lb
1 0
0
8
8
8
15
7
Mux A
Sa0
Sa1
Mux B
Sb0
Sb1
Control
Unit
8
8
A
B
18 19
Sop0
Sop1
Sop2
ALU
Result
Z N C V
Mul
Bit menos
significativo
de Result
Sop3
Mux
Status
La, Lb, Sa0, Sa1, Sb0, Sb1,
Sop0, Sop1, Sop2, Sadd0, W,
Lpc, Sdin, Spc, Sadd1, IncSp,
DecSp Sadd2 Sop3
5

Cada uno de los puntos tiene el mismo valor en el puntaje final. Se utilizará la siguiente distribución de puntos:

a
- Uso de registro y literal **50%**
- Uso de adder **50%**

b
- Uso de ambos registros **50%**
- Bit de control **50%**

c
- Correcta conexión de salida de muxes **33%**
- Bit de control **33%**
- Multiplexor de resultado de ALU y MUL **33%**. En este caso se puede asumir que el MUL es interno a la ALU, pero se debe haber especificado el bit de control de todas formas.

d
- Especificar bit menos significativo como condition code **50%**
- Conexión a Status register y expensión de bus a Control Unit **50%**

Pauta P3

Escriba una subrutina en Assembly x86 llamada `bin2num` que transforma un string de unos y ceros a un número entero no negativo.

Parámetros de entrada:

* `input` es un arreglo de caracteres '0' y '1' * `len` es la longitud del arreglo de caracteres

Parámetros de salida:

* Un número entero no negativo de 16 bits.

Ejemplo:

Input, len -¿ Output

"1101", 4 -¿ 13

La dirección de memoria de `input` y `len` deben pasarse en el stack, y el resultado debe ser retornado según la ABI de x86. Asuma que la arquitectura es de 16 bits.

Respuesta

La respuesta a esta pregunta estaba en el repositorio de ejemplos de Assembly, sólo había que adaptarla de 64 bits a 16.

```
        section .text
bin2num:
        push bp
        mov bp, sp
        mov ax, 0 ; Result
        mov di, 0 ; index
```

```
        mov dx, word [bp + 6]     ; count
        mov cx, word [bp + 4]     ; input
bin2num_loop:
        cmp di, dx                ; Check if there are characters left
        jge bin2num_end           ; If not, jump to the end
        shl ax, 1                 ; Multiply the result by two
        cmp byte [cx + di], '0'   ; Check if next char is a zero
        jz bin2num_no_add         ; If '0', no add
        add ax, 1                 ; If '1', add and go to loop
bin2num_no_add:
        inc di                    ; Increment the array's index
        jmp bin2num_loop
bin2num_end:
        pop bp
        ret

; No era necesario especificar un 'main'
main:
        push bp
        mov bp, sp
        push word [len]
        push input
        call bin2num
        pop bp
        ret

        section .data
input db "001101",0
len dw 6
```

Puntajes:

- Manejo correcto del base pointer **10%**
- Resultado en AX **10%**
- Paso de parámetros por el stack **20%**
- Lógica de la subrutina **60%**

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



IIC2343 – Arquitectura de Computadores (II/2014)

Solución Interrogación 1

Pregunta 1

a) ¿Cuál es el problema del *shift right* con los números binarios con signo?¿Es posible solucionarlo? **(1 pto.)**
Solución: Al interpretar el shift right como división por 2 en números binarios negativos, el signo se pierde ya que se agrega un cero a la izquierda. Esto se puede solucionar agregando explicitamente 1s en vez de 0s cuando se ejecuta esta operación.

b) ¿Que relación existe entre el tamaño de los elementos de memoria y su velocidad? Explique claramente su respuesta. **(1 pto.)**
Solución: Como todos los elementos electrónicos, mientras mayor es su velocidad, mayor es el costo por unidad de memoria (mega, giga, tera, etc). Por este motivo, los elementos de memoria más veloces son vendidos en tamaños mucho más pequeños que los elementos de memoria más lentos.

c) Describa el valor decimal del número 0x94A6, si este se interpreta como binario con signo. **(1 pto.)**
Solución: 0x94A6 = 1001010010100110 $\Rightarrow$ $C_2(1001010010100110) = 0110101101011010 = 27482$. Luego, el valor buscado es -27482

d) ¿Es posible utilizar una ALU de 8 bits, como la vista en clases, para realizar en paralelo dos sumas de números de 4 bits? Si la respuesta es positiva, describa el proceso. Si la respuesta es negativa, explique el motivo. **(1 pto.)**
Solución: Si asumimos que 2 números de 4 bits entran por cada bus y que la división lógica (no física) de estos se hace usando los 4 bits más significativos para uno de los números y los 4 menos siginificativos para el otro, la ALU calculará en paralelo la suma de los dos números de manera correcta, a menos que existe un carry entre el bit 3 y el bit 4, que es la posición donde se dividen los números. Este carry es en realidad overflow para los número de 4 bits, por lo que no debería tomarse en cuenta para el cálculo.

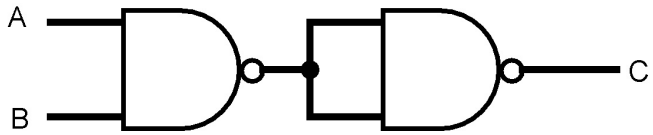
e) Describa un procedimiento para discretizar la señal bidimensional continua generada cuando se escanea un fotografía. **(1 pto.)**
Solución: Para realizar este proceso, dado que es una señal sin componente temporal, basta con discretizar sólo dos cantidades. La primera es la dimensión de la imagen, que se realiza dividiendo el alto y ancho de estas en pequeñas unidades indivisibles, llamadas pixels (picture elements). Luego, el valor de color de cada canal de uno de estos pixels es cuantizado, generalmente entre 0 y 255. De esta manera, una imagen digital consiste en una serie de pixels ordenados, donde cada uno almacena 3 valores enteros entre 0 y 255.

f) ¿Qué tipos de endianness se ocupan para leer datos mayores a 1 byte desde los **registros**? **(1 pto.)**
Solución: Los registros no utilizan endianness de ningún tipo, ya que su tamaño no está dividido en palabras de memoria. El endianness se utiliza exclusivamente en elementos de memoria que almacenan una palabra de memoria por cada dirección.

Pregunta 2

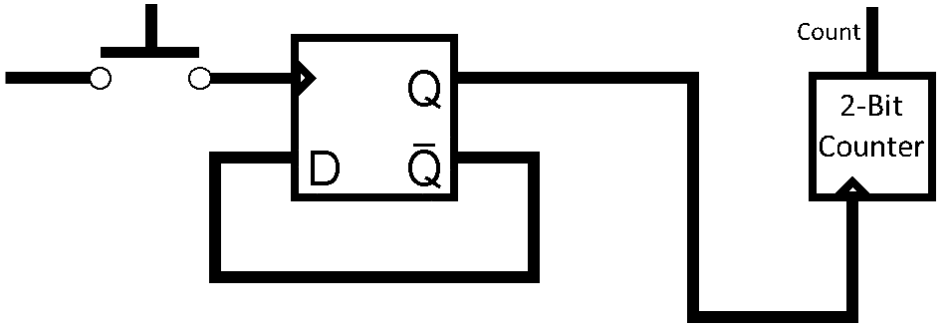
a) Implemente la compuerta AND sólo utilizando compuertas NAND (negación del AND). **(3 ptos.)**

Solución: Una posible solución se presenta en la siguiente figura:



b) Diseñe, utilizando todos los elementos de circuitos lógicos vistos en clases que necesite, un contador secuencial circular ascendente de 2 bits, que se incrementa cada **dos flancos** de subida de la señal de control. **(3 ptos.)**

Solución: La siguiente solución asume que el Flip-Flop parte con un 1 almacenado:



Pregunta 3

a) Demuestre que los números de punto flotante del tipo *float* del estándar IEEE754 **no** cumplen el principio de asociatividad de la suma, *i.e.*, $x + (y + z) = (x + y) + z$. **(3 ptos.)**

Solución: Los números del tipo float tienen problemas cuando las sumas incluyen números extremadamente dispares, *i.e.*, modulos muy grandes y muy chicos. Esto se debe a que el proceso de suma implica igualar los exponentes, proceso que puede llevar a que se trunquen elementos del número de menor módulo. Teniendo esto en cuenta, si se seleccionan los valores $x = -1{,}5 \times 10^{38}, y = 1{,}5 \times 10^{38}, z = 1{,}0$, tenemos lo siguiente en el lado izquierdo de la igualdad:

$$x + (y + z) = -1{,}5 \times 10^{38} + (1{,}5 \times 10^{38} + 1{,}0) = -1{,}5 \times 10^{38} + 1{,}5 \times 10^{38} = 0$$

En este caso, al igualar los exponentes en la suma de la derecha, el valor 1,0 se pierde, ya que el significante no tiene la cantidad de cifras necesarias. Si analizamos ahora el lado derecho de la igualdad tenemos:

$$(x + y) + z = (-1{,}5 \times 10^{38} + 1{,}5 \times 10^{38}) + 1{,}0 = 0 + 1{,}0 = 1{,}0$$

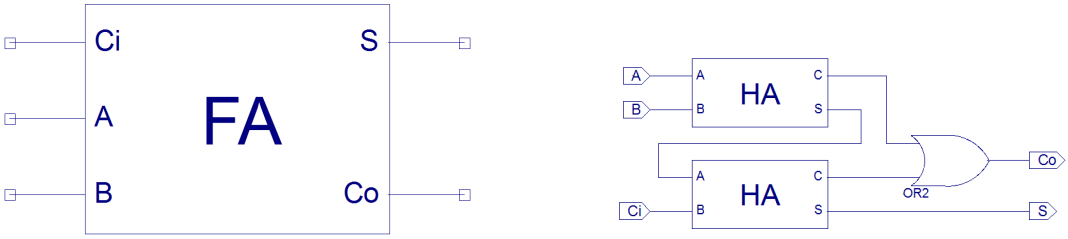
A diferencia del caso anterior, acá no se pierde información, ya que la primera operación se realiza con números con exponente similar.

b) Si hay algún problema eléctrico, como un alza de voltaje, es muy fácil corromper datos almacenados en binario. Por ejemplo, el número 10 (1010b) puede transformarse en 14 (1110b), con tan sólo modificar un bit. Describa una codificación binaria para los números 0 y 1, de manera que ésta permita detectar y corregir errores de a lo sumo 1 bit, *i.e.*, un bit de la codificación se ve alterado. **(3 ptos.)**

Solución: Del enunciado se obtienen claramente, que codificar un número usando sólo un bit no es suficiente para corregir errores. Si tomamos 2 bits y representamos el 0 como “00” y el 1 como “11”, también nos encontramos con un problema, ya que con un error, la codificación del 0 puede quedar como “01”, mientas que la del 1 podría quedar también como “01”. Siguiendo el mismo análisis, es fácil notar que si usamos tres bits y codificamos al 0 como “000” y al 1 como “111”, ningún error de 1 bit será suficiente para confundir las codificaciones.

Pregunta 4

Para todas las preguntas de esta sección, considere el componente de la Figura 1a) y su versión detallada en la Figura 1b):



(a) Full Adder (b) Full Adder detallado

Figura 1

a) Construya la tabla de verdad del Full Adder, considerando las entradas A, B y Ci, y las salidas Cout y S. **(1 pto.)**

Solución: Una posible solución se presenta en la siguiente tabla:

Ci	A	B	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

b) ¿Qué proceso(s) en necesario realizar en Xilinx para que un componente recién fabricado esté disponible para usarse en nuevos esquemáticos, como por ejemplo, en el Main? **(1 pto.)**

Solución: Para que un componente esté disponible para su uso se deben realizar los siguientes pasos en la pestaña de diseño:

- Guardar el esquemático
- Chequear las reglas del diseño (Check Design Rules)
- Crear el símbolo del esquemático (Create Schematic Symbol)

Luego de dichos pasos, el componente puede ser usado en nuevos esquemáticos.

c) Utilizando el componente del enunciado, la simbología de Xilinx y el área de trabajo adjunta, construya un sumador con tres entradas de 4 bits cada una, de forma que al procesarlo se vea como el de la Figura 2. Para esto puede basarse en la construcción realizada para el proyecto del curso. **(2 ptos.)**

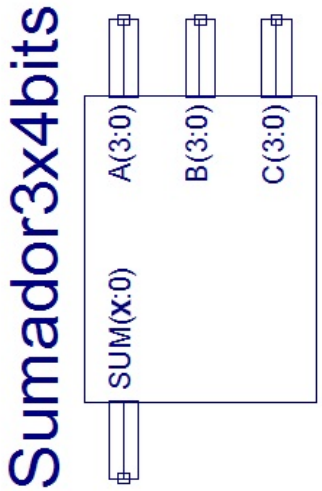
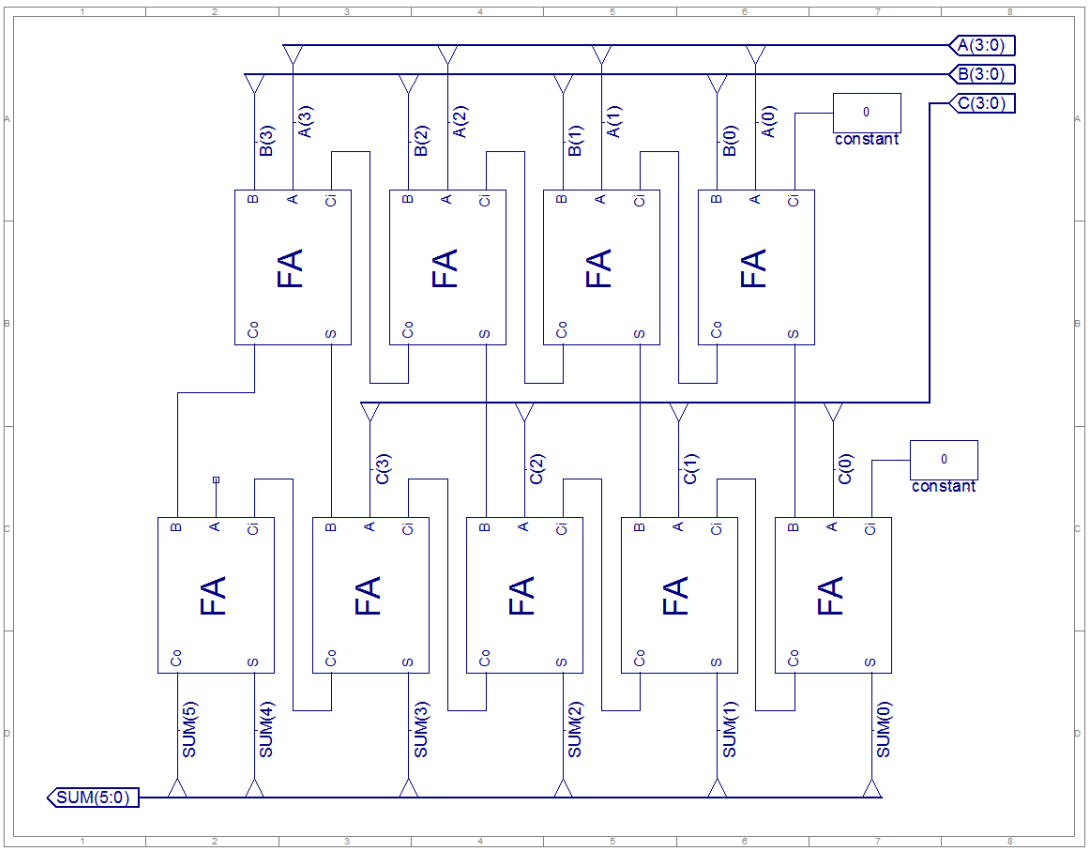


Figura 2: Sumador con tres entradas de 4 bits.

Solución: Una posible solución se presenta en la siguiente figura:

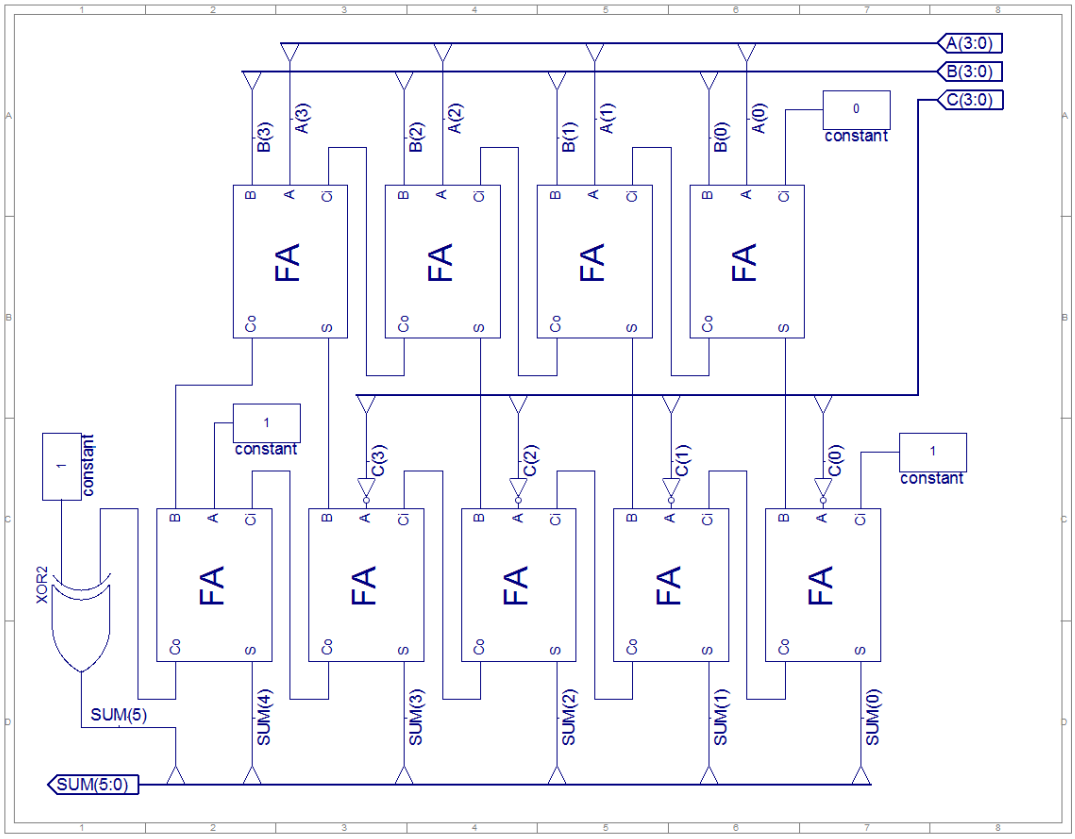


d) Considerando la salida SUM(X:0) del componente de la pregunta anterior, ¿cuál es el largo en bits de la palabra que sale de esta? Justifique su respuesta. **(1 pto.)**

Solución: El largo de la palabra es de 6 bits, pues en el peor caso $A = B = C = 1111$. Luego la suma $A + B + C = 101101$.

e) Comente, ¿qué modificaciones haría a su sumador para que ahora realizara la operación A + B – C? Incluya en su respuesta los supuestos que haga. **(1 pto.)**

Solución: Supongamos que $A + B > C$ y llamemos T_0 a la salida S del Full Adder que toma A_0 y B_0, entonces para realizar la operación mencionada habría que invertir los inputs de C y agregar un 1 en el Ci del Full Adder que toma C_0 y T_0. De esta forma la resta se realiza correctamente.



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



IIC2343 – Arquitectura de Computadores (I/2015)

Solución Interrogación 1

Pregunta 1

a) Describa una codificación para números enteros binarios distinta al complemento a 2, que utilize bit de signo y tenga una única representación para el cero. No es necesario que en esta codificación se cumpla que $A + -A = 0$ **(1 pto.)**
Solución: Una posible solución es la codificación ZigZag. Este esquema utiliza los números naturales para codificar de manera alternada los números negativos y positivos, *i.e.*: 0 ⇒ 0, -1 ⇒ 1, 1 ⇒ 2, -2 ⇒ 3, 2 ⇒ 4, etc. De esta manera, los números negativos siempre tendrán un 1 como bit menos significativo y los positivos tendrán un 0, y el cero será representado únicamente como 0.

b) Un reloj digital fue programado para hacer sonar su alarma a los 30 minutos, pero esta sonó a los 7680 minutos. ¿Por qué se produjo este problema? **(1 pto.)**
Solución: El problema ocurrió porque el número fue interpretado en un endianness incorrecto. El número 30 en hexadecimal, usando 2 bytes, se representa como 0x001E. Luego si el número fue leído en un endianness incorrecto, será interpretado como 0x1E00 = 7680

c) ¿Qué consideración hay que tener en una ALU para que esta pueda operar números con y sin signo (no simultáneamente)? **(1 pto.)**
Solución: No hay que tener ninguna consideración, la ALU opera números binarios sin importar si son positivos o negativos.

d) En la ALU vista en clases, el principal factor que limita la velocidad es la propagación del carry. ¿Es posible minimizar este problema? Fundamente su respuesta, o de ejemplos, en ambos casos. **(1 pto.)**
Solución: Para solucionar este problema, es posible utilizar una capa de lógica previa al sumador de ALU, que se encargue de predecir el valor del carry para cada *Full-Adder*, en función de las entradas anteriores. Esta técnica se conoce como carry-lookahead.

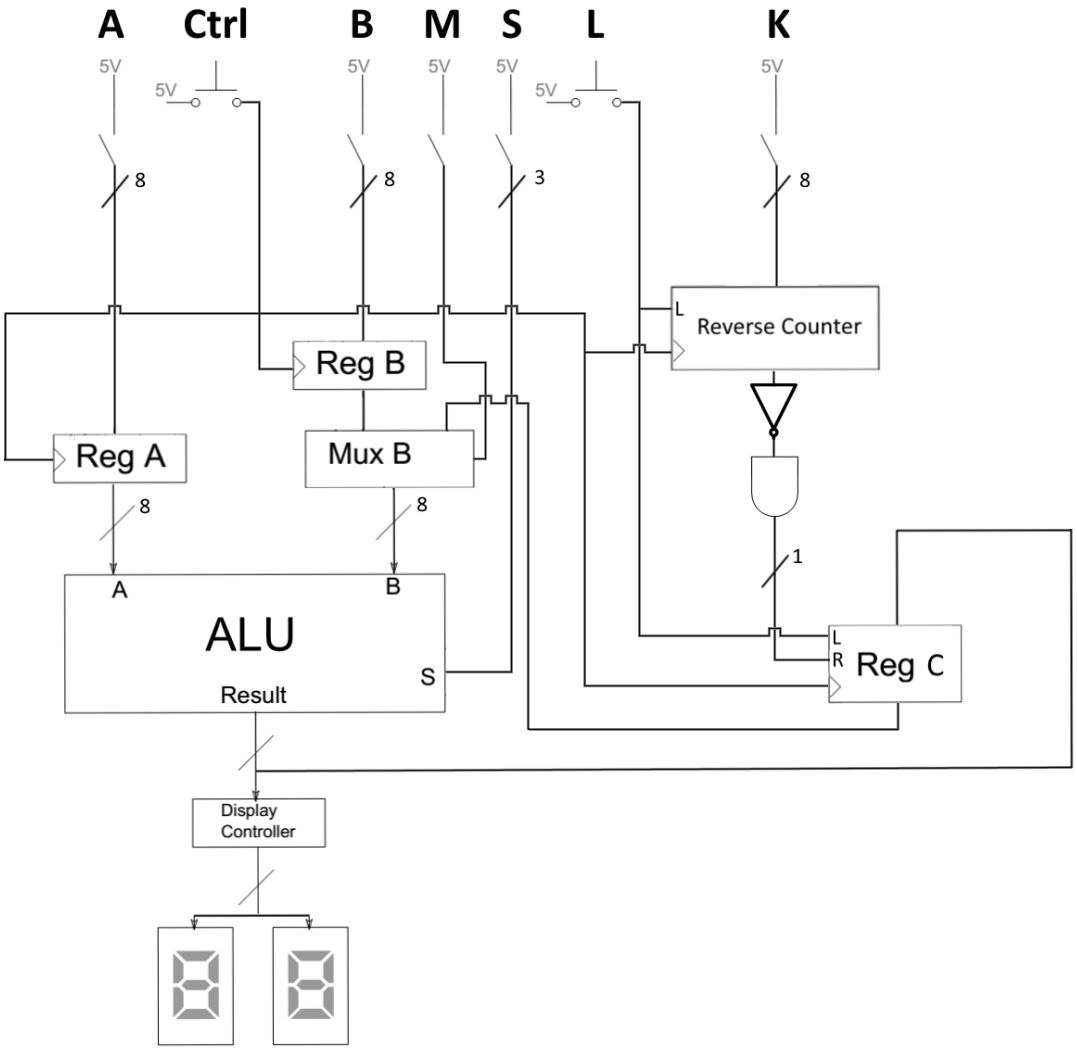
e) ¿Por qué es más lento realizar operaciones sobre números de punto flotante de base decimal, que sobre los de base binaria? **(1 pto.)**
Solución: Un posible motivo es la imposibilidad de realizar los cálculos directamente en binario, debido al uso de otra base, lo que implica, entre otras cosas, que el punto (coma) no puede moverse de manera simple al momento de realizar multiplicaciones o sumas.

f) Si una memoria RAM es construida utilizando los elementos de memoria vistos en clases, ¿a qué tipo de memoria RAM corresponde? **(1 pto.)**
Solución: Corresponde a una SRAM (Static RAM).

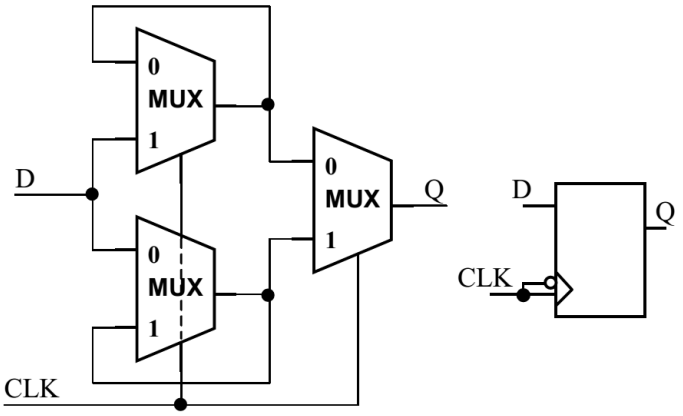
Pregunta 2

a) Diseñe una calculadora de 8 bits y 8 operaciones, que permita visualizar los resultados de las operaciones. Además, esta calculadora debe permitir respaldar el resultado de **una** operación, para ser usado posteriormente. Este resultado almacenado sólo estará disponible durante las K siguientes operaciones, donde K es un número natural definido por el usuario, no mayor a 255. Este resultado almacenado podrá ser sobrescrito por el usuario, durante cualquier de las K operaciones siguientes, en cuyo caso, el proceso comenzará nuevamente de cero. **(3 ptos.)**

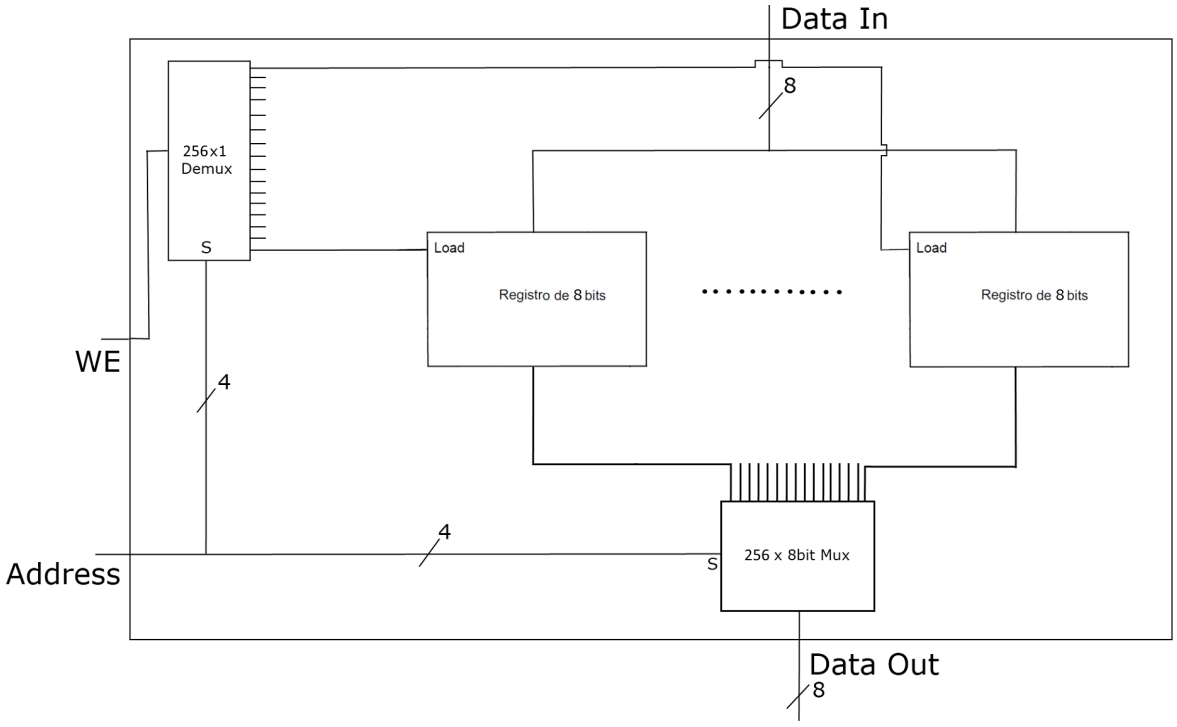
Solución:



b) Diseñe una memoria RAM DDR de 256B. Sólo puede utilizar, como elementos de memoria, flip-flops y registros. **(3 ptos.)**
Solución: Este problema puede dividirse en 2 partes. La primera parte, y más compleja, es diseñar un flip-flop que se active en ambos flancos (2 ptos.). La segunda, y más sencilla, es diseñar una memoria RAM utilizando esos flip-flops (1 pto).
Una posibilidad de diseño para el flip-flop activado en ambos flancos es la siguiente:



En la figura, a la izquierda se muestra la construcción del flip-flop activado en ambos flancos, mientras que a la derecha, el símbolo de este flip-flop. Luego, utilizando este flip-flop, construir un registro que se active en ambos flancos es trivial (igual que para el caso activado en el flanco de subida).
Finalmente, una posible solución para la construcción de la memoria, que utiliza registros activados en ambos flancos, es la siguiente:



Pregunta 3

a) Implemente, utilizando el lenguaje que más le acomode, la clase **Float**, que emula el comportamiento del tipo de datos *float* (IEEE754). Su implementación debe cumplir lo siguiente:

- incluir métodos para sumar, restar y multiplicar.
- un constructor que reciba como entrada un string que describe el número en formato decimal (no notación científica).
- métodos para obtener el signo, el exponente y el significante.
- manejar los casos especiales.

Internamente, la clase **NO** puede utilizar tipos de dato de punto flotante (float, double, etc.). Asuma la existencia de funciones que realizan la transformación entre strings, números binarios, decimales, etc. **(4 ptos.)**

Solución:

```
public class Float
{
  public int sign = 0;
  public int exp = 127;
  public String significant = toBinaryString(0, 23);
  public String decimal;

  // funcion toBinaryString(String number) retorna un string con la
  // representacion binaria del numero racional contenido en number.

  // funcion toBinaryString(int number, int length) transforma un
  // numero entero su representacion binaria de largo length

  public Float(String number)
  {
    this.decimal = number;
    String binaryString;
    if (number.charAt(0) != '-')
    {
      this.sign = 0;
      binaryString = toBinaryString(number);
    }
    else
    {
      this.sign = 1;
      binaryString = toBinaryString(number.substring(1));
    }
    if (binaryString.charAt(0) == 0)
    {
      if (binaryString.length() == 1)
      {
        this.exp = 0;
        this.significant = toBinaryString(0, 23);
        return;
      }
      else
      {
```

```
          int i = 2;
          for (; binaryString.charAt(i) != '1'; i++)
          {
            this.exp--;
          }
          i++;
          for (int j = 0; i < binaryString.length() || i < 25; i++,j++)
          {
            this.significant.charAt(j) = binaryString.chaAt(i);
          }
          return;
        }
      }
      else
      {
        int i = 1;
        for (; binaryString.charAt(i) != '.'; i++)
        {
          this.exp++;
          this.significant.charAt(i-1) = binaryString.charAt(i);
        }
        i++;
        for (int j = 0; i < binaryString.length() || i < 25; i++,j++)
        {
          this.significant.charAt(j) = binaryString.chaAt(i);
        }
        return;
      }
    }
    public Float add(Float f)
    {
      return new Float(addFloat(this.decimal, f.decimal));
    }
    public Float sub(Float f)
    {
      return new Float(subFloat(this.decimal, f.decimal));
    }
    public Float mul(Float f)
    {
      return new Float(mulFloat(this.decimal, f.decimal));
    }
    public String getSign()
    {
      return String.valueOf(this.sign);
    }
    public String getExp()
    {
      return toBinaryString(this.exp, 8);
    }
    public String getSignificant()
    {
      return this.significant;
    }
  }
```

b) Describa un procedimiento para realizar la multiplicación de dos números naturales binarios de 8 bits, usando sólo las operaciones disponibles en la ALU vista en clases. **(2 ptos.)**
Solución: Asumiendo que *Producto* = *Multiplicando* * *Multiplicador*, el procedimiento se resume de la siguiente manera:

a) Inicialmente se tiene almacenado el multiplicando y el multiplicador y se tiene espacio reservado para almacenar el producto, inicializado en cero.

b) Verificar si bit menos significativo del multiplicador es 1.

- Si es 1, sumar el multiplicando al producto.

c) Hacer shift-left del multiplicando.

d) Hacer shift-right del multiplicador.

e) Verificar si este proceso se ha repetido 8 veces

- Si no se ha repetido 8 veces, volver al paso *b)*.
- Si se ha repetido las 8 veces, se termina el proceso.

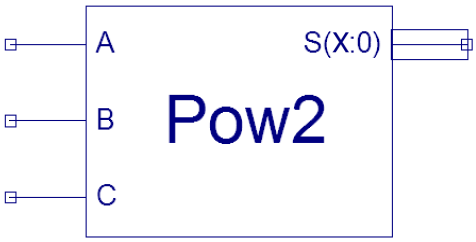
Pregunta 4

Para las preguntas a) y b) de esta sección, considere el componente de la Figura 1):

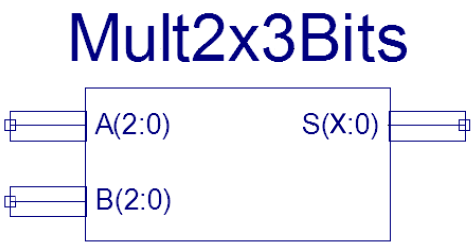


Figura 1: Constante 1 de 3 entradas de 1 bit.

Para las preguntas d), e), f) y g) de esta sección, considere los componentes de la Figura 2a) y 2b):



(a) Eleva al cuadrado 1 entrada de 3 bits.



(b) Multiplicador de 2 entradas de 3 bits.

Figura 2

a) Construya la tabla de verdad del componente Const1, considerando las entradas A, B y C, y la salida S. **(0.5 ptos.)**

Solución: Una posible solución se presenta en la siguiente tabla:

A	B	C	S
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Figura 3: Tabla de verdad Const1.

b) Utilizando el componente de la Figura 1) como referencia, la simbología de Xilinx y el área de trabajo adjunta, construya una posible implementación del mismo utilizando sólo compuertas lógicas. No puede

utilizar el símbolo de constante y debe usar todas las entradas. **(0.5 ptos.)**

Solución: Una posible solución se presenta en la siguiente figura:

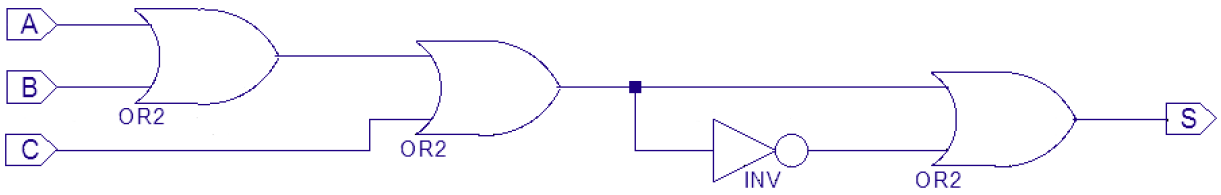


Figura 4: Posible implementación Const1.

c) En Xilinx, ¿Qué es y para qué sirve un Bus Tap?, ¿Cuáles son los requisitos, sobre un Bus de Datos y los Bus Tap, para su correcto funcionamiento? **(0.5 ptos.)**

Solución: Un Bus Tap es un símbolo de Xilinx que permite interactuar con los Buses de Datos, introduciendo o sacando bits, específicos o rangos de ellos, de los mismos. Para su correcto funcionamiento es importante considerar el tamaño del Bus de Datos con el que se está trabajando, que los nombres de ambas estructuras coincidan y el orden en que entran o salen los bits del Bus de Datos.

d) Considerando las salidas S(X:0) de los componentes de la Figura 2), ¿cuáles son los largos en bits de las palabras que salen de cada uno? Justifique sus respuestas. **(0.5 ptos.)**

Solución: Para el componente Pow2, el largo de la palabra es de 6 bits, pues en el peor caso $A = B = C = 1$. Dado lo anterior, esto se interpretaía como el número 7, luego su cuadrado es $49 = 110001$. Para el componente Mult2x3Bits, el largo de la palabra también es 6 bits, pues en el peor caso $A = B = 111$. Luego la multiplicación $A * B = 49 = 110001$.

e) Construya la tabla de verdad del componente Pow2, considerando las entradas A, B y C, y la salida S. **(0.5 ptos.)**

Solución: Considerando que A es el bit más significativo y C es el menos significativo, un posible solución se presenta en la siguiente tabla:

A	B	C	S5	S4	S3	S2	S1	S0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	0	0	1

Figura 5: Tabla de verdad Pow2.

f) Utilizando el componente de la Figura 2a) como referencia, la simbología de Xilinx y el área de trabajo adjunta, construya un componente capaz de elevar al cuadrado una entrada de 3 bits, de forma que al procesarlo se vea como el de la Figura 2a). Para esto no puede utilizar el componente de la Figura 2b). **(1.5 ptos.)**

Solución: Una posible solución se presenta en la siguiente figura:

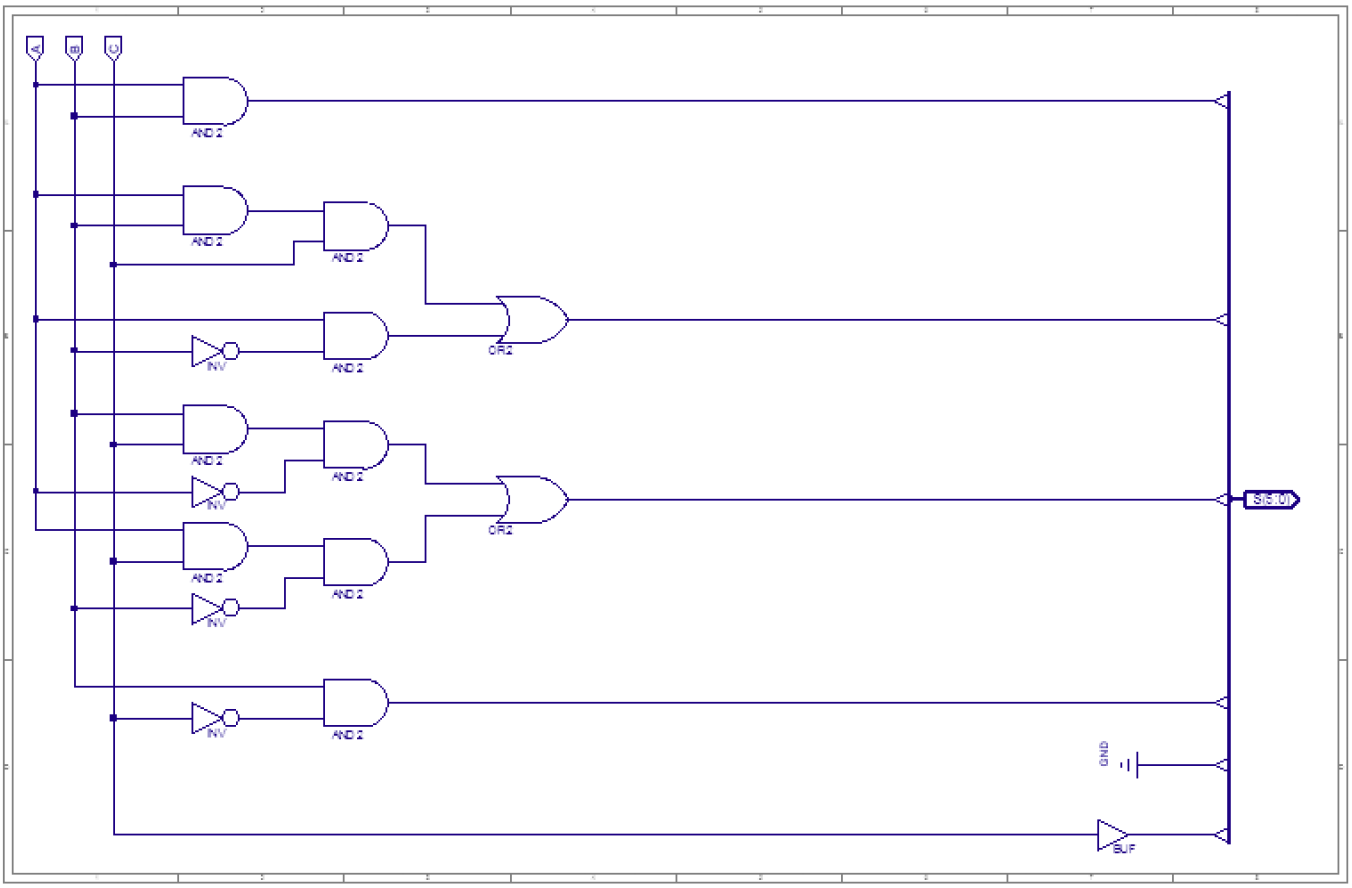


Figura 6: Posible implementación Pow2.

g) Utilizando los componentes de la Figura 2) como referencia, la simbología de Xilinx y el área de trabajo adjunta, construya un nuevo símbolo con 2 entradas de 3 bits cada una, A y B, y una única salida S, tal que $S = (A + B)^2$. Para esto puede basarse en la construcción realizada para el proyecto del curso. **(2 ptos.)**

Solución: Es necesario notar que no se puede realizar la suma primero, pues no se podría utilizar el Pow2 con el resultado de esta. Luego es necesario utilizar la igualdad $(A + B)^2 = A^2 + 2 * A * B + B^2$. Una posible solución se presenta en la siguiente figura:

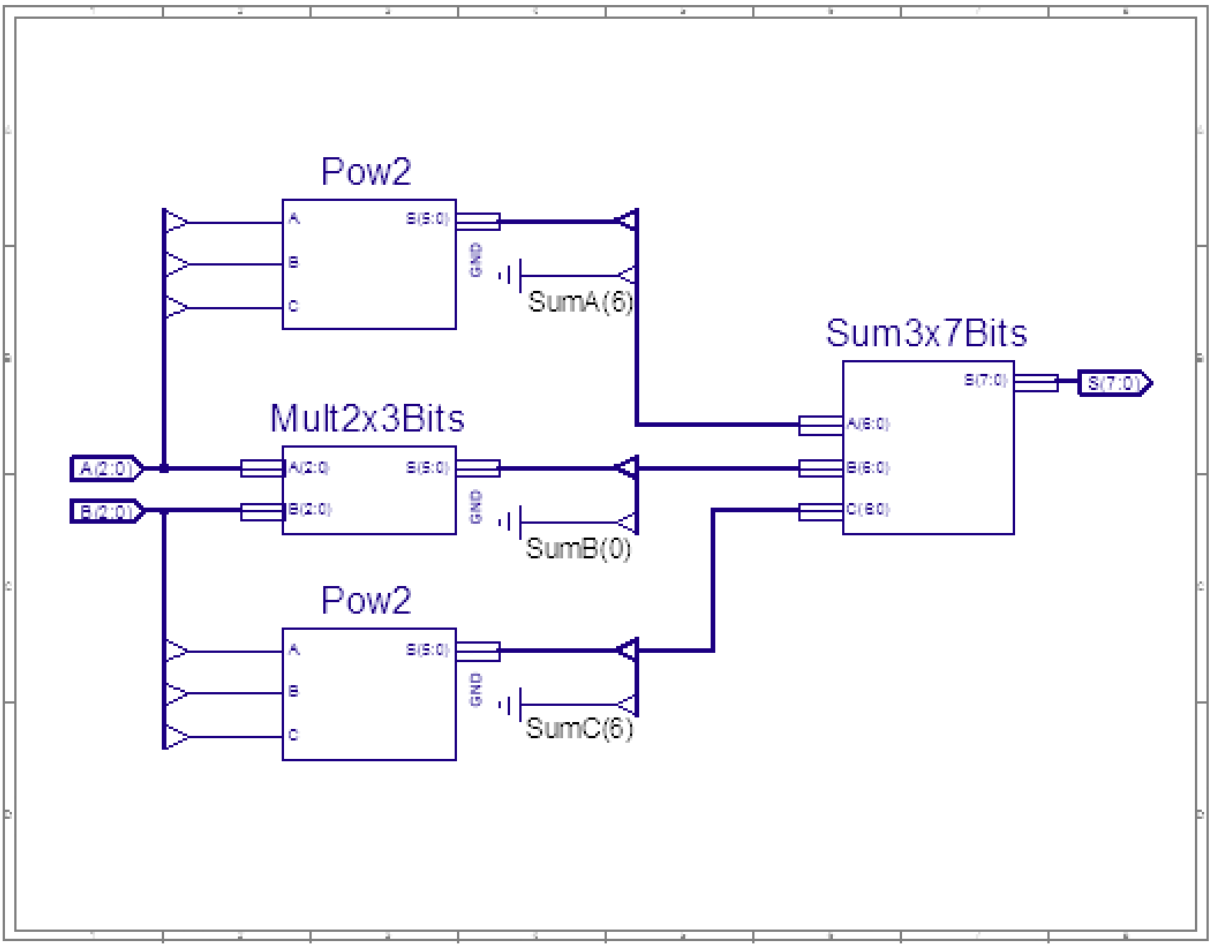


Figura 7: Posible implementación para $S = (A + B)^2$.

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



IIC2343 – Arquitectura de Computadores (II/2015)

Interrogación 1

Pregunta 1

a) Describa como es posible intercambiar el valor de dos variables binarias de 4 bits, usando sólo la operación XOR. **(1 pto.)**
Solución: Basta con aplicar tres veces seguidas la operación XOR ($\oplus$), *i.e.*, $X = X \oplus Y, Y = X \oplus Y, X = X \oplus Y$.

b) Dada una cantidad acotada de piedras (**N**), ¿qué representación posicional permite representar más números usando a lo más **N** piedras? Asuma que en cada posición, la cifra está representada por la cantidad de piedras. **(1 pto.)**
Solución: Siempre será la representación de base 2, ya que al usar sólo una piedra por posición, alcanza a llegar a un exponente mayor.

c) ¿Cómo implementaría en la máquina programable una instrucción que no haga nada, *i.e.*, que no modifique el estado del computador? Describa las señales de control. **(1 pto.)**
Solución: Basta cualquier conjunto de señales de control, tal que L_A y L_B sean iguales a 0.

d) Asuma que a una memoria RAM con un bus de direcciones de 12 bits, se le conecta un clock de 4KHz. ¿Cuánto tiempo toma cómo mínimo llenar esta memoria? **(1 pto.)**
Solución: Demorará 1 segundo, ya que $4K = 2^{12}$, luego en un segundo se podrán escribir 2^{12} palabras, que es justamente la capacidad de la memoria.

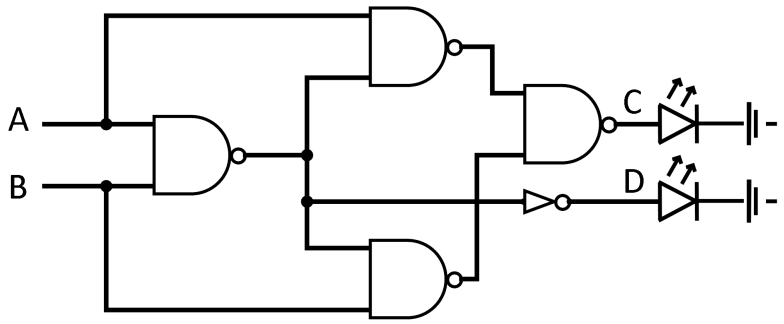
e) Muestre que con la compuerta NAND se pueden representar todas las compuertas binarias de la lógica booleana. **(1 pto.)**
Solución: Basta con mostrar como se pueden formar AND ($\wedge$), OR ($\vee$) y NOT ($\neg$) usando solo NAND ($\overline{\wedge}$):

$$\begin{aligned} \neg A &= A \overline{\wedge} A \\ A \wedge B &= (A \overline{\wedge} B) \overline{\wedge} (A \overline{\wedge} B) \\ A \vee B &= (A \overline{\wedge} A) \overline{\wedge} (B \overline{\wedge} B) \end{aligned}$$

f) ¿Cuántos caracteres puede tener como máximo un archivo de texto en castellano de 1KB no comprimido, donde sólo se utilizan como caracteres las letras mayúsculas, el punto, la coma y el espacio? **(1 pto.)**
Solución: Dado que se utilizan en total 30 caracteres, se necesitan 5 bits por carácter. Luego, la cantidad máxima de caracteres es $\lfloor \frac{1024 \times 8}{5} \rfloor = 3222$.

Pregunta 2

a) Describa con una tabla de verdad y con palabras que es lo que realiza el siguiente circuito: **(1 pto.)**



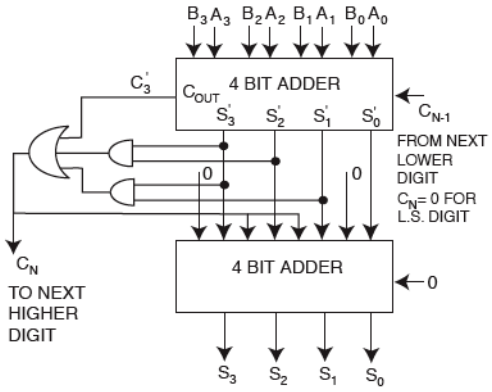
Solución: Si ambas entradas son 0, los LEDs permanecen apagador. Si sólo una de las entradas está en 1, se prende el LED superior. Finalmente, si ambas entradas están prendidas, se prende el LED inferior.

A	B	C	D
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

b) Describa como representaría en un computador, a nivel de hardware, el conjunto completo de los números racionales. Considere además de la representación, como se implementarían las operaciones aritméticas básicas $(+, -, \times, /)$ **(2 ptos.)**
Solución: Dado que todos los números racionales se escriben como $\mathbb{Q} = \frac{a}{b}, a \in \mathbb{Z}, b \in \mathbb{N}$, es posible representarlos usando dos registros, donde uno representa el numerador y el otro el denominador. Para operarlos se utilizan las reglas básicas de los racionales:

- Multiplicación: Se multiplica numerador con numerador y denominador con denominador. Este producto se puede implementar sin problemas con compuertas lógicas.
- División: Se intercambia en el divisor, el contenido de los registros del numerador y denominador. Luego, se realiza el producto como en el caso anterior.
- Suma y resta: Se igualan los denominadores y luego se ejecuta la operación. La igualación se realiza mediante multiplicación entre números enteros y naturales, que es fácilmente implementada con compuertas lógicas, lo mismo que la posterior suma.

c) El BCD (binary coded decimal), es una codificación que representa cada cifra de un número natural decimal utilizando una cantidad fija de bits. Utilizando las compuertas lógicas vistas en clases, diseñe un sumador que toma como entradas dos números naturales de 4 cifras, representados en BCD. **(3 ptos.)**
Solución: La siguiente figura muestra la solución para una cifra. Para 4 cifras, se usan 4 como el de la figura, conectados en cascada.



- ADD **0110** WHEN C_N**=1**
- ADD **0000** WHEN C_N**=0**

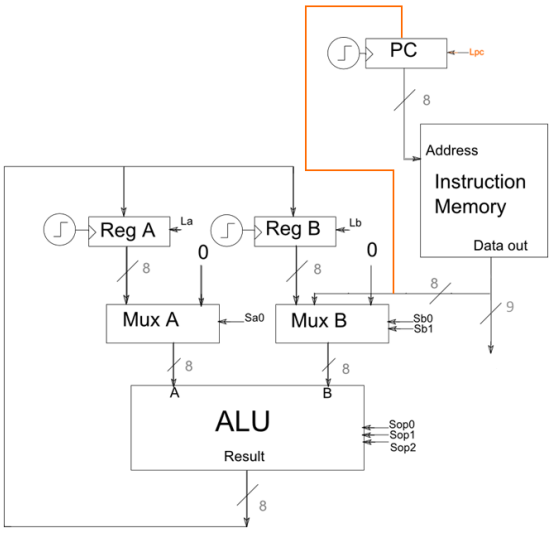
Pregunta 3

a) Dada la siguiente secuencia de instrucciones de la máquina programable vista en clases: A = 8, B = 5, A = A+B, A = A-B, ¿a qué secuencia de números corresponden estas instrucciones, si son interpretadas como floats? Interprete los números usando little endian. **(1 pto.)**
Solución: Utilizando el mismo orden que el de la máquina vista en clases, *i.e.*, L_A, L_B, S_A, S_B, ALU, más el literal a continuación, las señales de control de las instrucciones quedan de la siguiente manera: Como cada par (instrucción,literal) utiliza 16, se necesitan 2 instrucciones para poder codificar un float. El

L_A	L_B	S_A	S_B	ALU	Lit	Instr.
1	0	1	01	000	00001000	A=8
0	1	1	01	000	00000101	B=5
1	0	0	00	000	00000000	A=A+B
1	0	0	00	001	00000000	A=A-B

estándar float ordena sus bits, de más a menos significativo, de la siguiente manera: signo (1), exponente (8), significante (23). Luego, asumiendo little endian (orden de los bytes) la representación binaria y decimal de los números generados por las instrucciones son: 00000101 01101000 00001000 10101000 = $1{,}09101775 \times 10^{-35}$, 00000000 10000001 00000000 10000000 = $1{,}1846958 \times 10^{-38}$.

b) Modifique el diagrama de la máquina programable, de manera que soporte la ejecución de la instrucción GOTO **Dir**, que fuerza que la siguiente instrucciones en ejecutarse sea la ubicada en la dirección **Dir**. **(2 ptos.)**
Solución:



c) Asuma que dos máquinas programables de 8 bits están conectadas mediante un bus de datos unidireccional, donde los datos van desde la ALU de la primera (máquina **X**) hasta el registro A de la segunda (máquina **Y**). Considere además que el clock de la máquina **X** tiene el doble de frecuencia que el de **Y**. Para compensar por esta diferencia de velocidad, el clock de la máquina **X** se detiene después de **K** ciclos, durante un período de tiempo también igual a **K** ciclos, luego de lo cual vuelve a funcionar, repitiendo el proceso. Teniendo esto en consideración, diseñe el bus de datos, de tal manera que la máquina **Y** no pierda información. **(3 ptos.)**
Solución: El bus de datos se puede modelar fácilmente utilizando un arreglo de K registros (asumimos

K potencia de 2), donde la máquina **X** escribe secuencialmente en ellos en cada ciclo, desde 0 a K-1, mientras que por el lado de **Y** esta lee de estos registros también de 0 a K-1. Las salidas de los K registros están conectadas a un mux de K entradas por el lado de **Y**, que se encarga de seleccionar el registro a leer, y su bus de control se conecta a un contador ascendente de $\log_2(K)$ bits, que utiliza el clock de la máquina **Y**. Las entradas de los K registros están conectadas a la salida de la ALU de **X**. Para decidir en que registro escribir, *i.e.* que señal de Load activar, se utiliza un esquema similar al de las salidas, pero el contador utiliza el clock de **X** y se utiliza un demultiplexor (inverso de un multiplexor) en lugar de un multiplexor.

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



IIC2343 – Arquitectura de Computadores (I/2016)

Interrogación 1

Pregunta 1

a) ¿En qué casos es posible soportar la instrucciones ADD B,Lit en el computador básico, sin modificar su hardware ni sobrescribir datos? Para los casos negativos, indique que modificaciones al hardware y/o assembly se deberían hacer para soportarla. **(1 pto.)**
Solución: Sólo podrían estar soportadas, sin modificaciones, la instrucciones ADD B,1 y ADD B,0. Si se quiere tener soporte completo, se debe agregar al MUX A una entrada que venga desde el bus de literales.

b) ¿Por qué existen más números negativos que positivos en el complemento a 2? ¿Cómo se podría modificar el esquema para que hubiera más números positivos que negativos? **(1 pto.)**
Solución: Al seleccionar el rango de los número que comienzan con 0 como el de los positivos, la inclusión del 0 implica automáticamente que tendran un número menos que los negativos. Si los negativos se representaran con un 0 al principio, entonces los positivos tendrían un número más.

c) ¿Que pasaría sin se quita el registro STATUS del computador básico y se conectaran directamente las señales ZNCV a la unidad de control? **(1 pto.)**
Solución: Al eliminar este registro, se elimina también la sincronización con el flanco de subida, por lo que se corre el riesto que las señales ZNCV que se utilizan en la unidad de control para decir el salto, no correspondan a la operación en la cual se basa el salto, generalmente un CMP.

d) Indique la codificación en complemento a 2 de 8 bits del número decimal -128. **(1 pto.)**
Solución: -128 corresponde al número con mayor valor absoluto en complemento a 2 de 8 bits. Por lo tanto, su codificación es 10000000. Cabe destacar que no existe la codificación en complemento a 2 de 8 bits para el número 128.

e) ¿Cuál es la frecuencia máxima que puede tener el clock del computador básico? ¿Qué pasa si un clock con una frecuencia mayor a la máxima es conectado al computador básico? **(1 pto.)**
Solución: La frecuencia máxima es aquella tal que su período es igual a la duración de la instrucción más lenta del computador. Si un clock con una mayor frecuencia es conectado al computador básico, se perderá la sincronía cada vez que se ejecute esta instrucción, lo que implica que el resultado será incorrecto.

f) Diseñe un circuito que multiplique por 4 un número natural de 8 bits. Asuma que no ocurrirá overflow. **(1 pto.)**
Solución: Basta diagramar un componente con 8 entradas, donde las 6 menos significativas se conectan, manteniendo el orden, a las 6 salidas más significativas, y al resto se le conectan 0s. De manera más formal, si A es el bus de entrada y S el de salida, el componente está dado por:

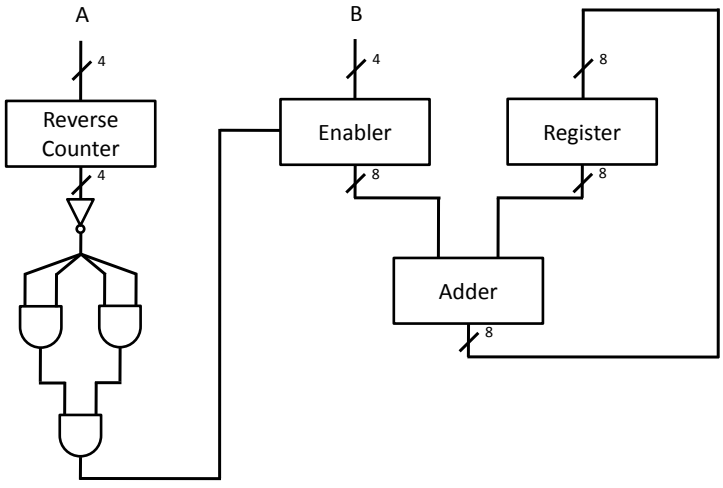
$$S_i = \begin{cases} A_{i-2} & \forall i \geq 2 \\ 0 & \text{en otro caso} \end{cases}.$$

Pregunta 2

a) El estándar IEEE754 de 32 bits (float) utiliza 1 bit para el signo, 23 bits para el significante y 8 para el exponente. Dos números codificados con este estándar, difieren en su significante en la unidad mínima. Asumiendo que ambos números tienen el mismo exponente, ¿cuál será la diferencia en el valor decimal de estos números? Parametrice esta diferencia en función de alguno de los elementos del estándar y comente con respecto a la significancia de este valor. **(2 ptos.)**
Solución: Dado que el significante sólo tiene 23 bits, la menor diferencia entre dos de estos se dará en el bit menos significativo, por ejemplo, 1.00000000000000000000000b y 1.00000000000000000000001b. Luego, dado que ambos números tienen el mismo exponente, la diferencia en el valor decimal entre estos dos números dependerá justamente de este exponente y está dada por 2^{E-23}, donde E es el exponente. Esto implica que mientras más grande sea este, mayor será esta diferencia. Para cuantificar, si $E = 0$, la diferencia será de 2^{-23} lo que parece razonable, pero si $E = 50$, esta diferencia será de $2^{50-23} = 2^{27} =$ 134,217,728.

b) Diagrame y explique una posible implementación de la instrucción JLE Dir (opcode 1000011) al interior de la unidad de control. **(2 ptos.)**
Solución: El primer paso consiste en verificar que el opcode de la instrucción actual corresponda al de JLE, para lo cual se realiza un NXOR con una máscara que tenga el valor del opcode y luego se verifica que todos los bits del resultado sean 1. El output de esta última verificación será llamado A_0. De manera independiente, se debe conectar a un OR las salidas de Z y N. La salida de esta OR se llamará A_1. Finalmente, el valor de L_{PC}, se obtiene conectando A_0 y A_1 a una compuerta AND.

c) Construya un circuito que permita multiplicar dos números naturales de 4 bits. **(2 ptos.)**
Solución: En el siguiente circuito se asume que los factores de la multiplicación son A y B. Además, en el tiempo 0, el contador es cargado con el valor de A y el registro con el valor 0. Finalmente, la transformación de 4 a 8 bits posterior al enabler, se realizar llenando con 0s los 4 bits más significativos.



Pregunta 3

a) Una máquina RAM, es un tipo de computador en el cual se utiliza sólo 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, sólo 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). **(3 ptos.)**
Solución: Para construir esta máquina basta con utilizar el hardware del computador básico, manteniendo además todos los opcodes, pero modificando el assembly, de tal manera que las operaciones que utilizan registros sean sustituidas por las que utilizan las direcciones de memoria. La implementación de cada una de estas instrucciones se puede realizar trivialmente utilizando una secuencia de opcodes adecuada, que implemente con instrucciones del computador básico la misma funcionalidad. Por ejemplo, la secuencia para ADD (Dir1),(Dir2) serían los opcodes correspondientes a la siguiente secuencia de instrucciones del computador básico: MOV A, (Dir1) ; MOV B, (Dir2) ; ADD A, B ; MOV (Dir1), A.

b) Modifique el computador básico para dar soporte a la instrucción CLEAR, que setea en 0 el valor de todos los registros (excepto PC y STATUS) y el de todas las palabras de la memoria de datos. Indique el(los) opcode(s) y señal(es) de control de la nueva instrucción. **(3 ptos.)**
Solución: La solución más simple y directa es agregar a los registros A y B y a la memoria de datos, una señal CLR, que setea en 0 el valor almacenado, tal como el registro visto en cátedra. Esta señal se agrega al conjunto de señales de control generada por la unidad de control, y se seteará en 1 sólo cuando la unidad de control reciba el opcode de la instrucción CLEAR (1001110).

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



IIC2343 – Arquitectura de Computadores (II/2016)

Solución Interrogación 1

Pregunta 1

a) Explique en detalle que pasaría en el computador básico si el registro PC cambia su funcionamiento de flanco de subida a flanco de bajada. **(1 pto.)**
Solución: Al cambiar la sincronización, el almacenamiento en los registros o memoria, no coincidiría necesariamente con lo que la instrucción espera almacenar. Dependiendo de la velocidad de ejecución de la instrucción, algunas instrucciones funcionarán de la misma manera, mientras que otras almacenarán resultados incorrectos.

b) ¿Qué tipo de dato puede representar más valores distintos, **int** de 32 bits o **float** de 32 bits? **(1 pto.)**
Solución: El **int** puede almacenar 2^{32} posible valores distintos, mientras que en **float**, al tener varias combinaciones de bits que representan lo mismo (NaN), tiene menos de 2^{32} combinaciones posibles disponibles.

c) Diseñe un circuito que calcule el resultado de la multiplicación de dos números de 1 bit. **(1 pto.)**
Solución: La multiplicación de dos números de 1 bit correponde a combinar las dos señales con una compuerta AND.

d) Describa un mecanismo para determinar el endianness de un computador. **(1 pto.)**
Solución: Una posible opción es escribir en memoria un **int** con valor 1 a partir de la posición 0 y luego leer el contenido de esa posición. Si lo leído es 0, el computador utiliza **big endian**, mientras que si es 1, es **little endian**.

e) Describa en detalle los pasos que debe realizar un **assembler** para procesar código assembly del computador básico. **(1 pto.)**
Solución: El primer paso es registrar todos los *labels* definidos en el segmento **DATA**. Luego, en el segmento **CODE**, sustituir las referencias a estos labels por el literal correspondiente (dir. de memoria). Finalmente, se debe transformar cada instrucción del segmento **CODE** a un opcode.

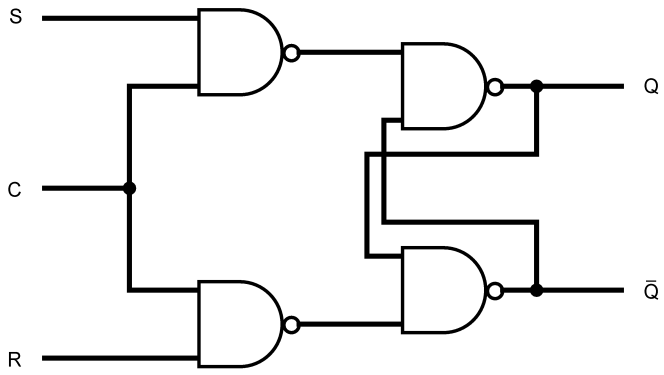
f) Demuestre que el complemento a 2 del complemento a 2 de un número x es igual a x, *i.e.*, $x = C_2(C_2(x))$. **Hint**: asuma que $C_2(x + y) = C_2(x) + C_2(y)$. **(1 pto.)**
Solución:

$$\begin{aligned} x + C_2(x) &= 0 && / \ C_2 \\ C_2(x + C_2(x)) &= 0 && / \text{ utilizamos el hint} \\ C_2(x) + C_2(C_2(x)) &= 0 \\ C_2(x) + C_2(C_2(x)) &= x + C_2(x) \\ C_2(C_2(x)) &= x \end{aligned}$$

Pregunta 2

a) Dados números naturales K, N, T, donde $K \geq 2 \wedge N > T > 0$, describa un algoritmo para transformar de manera **eficiente** números naturales codificados en base K^N a base K^T. No es válido utilizar la fórmula general de transformación de bases $\sum_{k=0}^{n-1} s_k \times b^k$. **(2 ptos.)**
Solución: Un algoritmo eficiente puede realizarse en dos pasos: i) Transformar cada cifra del número original en N cifras que representen el mismo número en base K, mantiendo el orden correlativo entre cifras. ii) Agrupar las cifras del número expandido en grupos de T y transformar cada grupo en un número en base K^T, nuevamente manteniendo el orden correlativo.

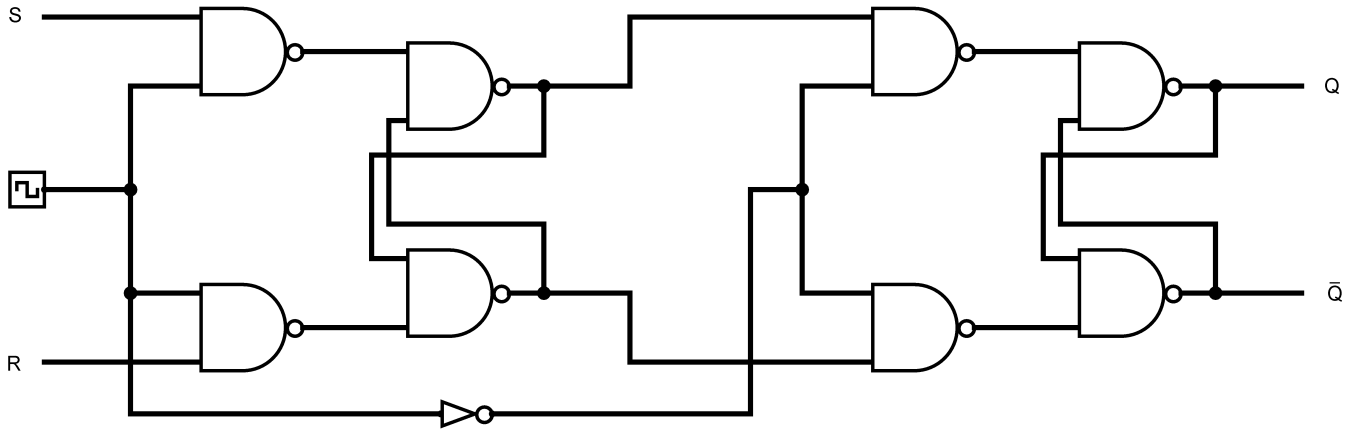
b) i) Modifique un latch tipo RS agregando una señal de control C, tal que los cambios en el estado del latch sólo se realicen cuando $C = 1$. **(2 ptos.)**
Solución:



ii) Diseñe un flip-flop tipo RS, dado por la siguiente tabla de verdad: **(2 ptos.)**

S	R	C	Q	$\overline{Q}$
0	0	↑	Q	$\overline{Q}$
0	1	↑	0	1
1	0	↑	1	0
1	1	↑	?	?
X	X	0,1,↓	Q	$\overline{Q}$

Solución:



Pregunta 3

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.

a) Haga el diagrama del computador, considerando que las matrices pueden tener como máximo $N \times N$ elementos, cada uno de 1 byte. **(3 ptos.)**
Solucion: Basta con utilizar el diagrama del computador básico, aumentando el tamaño de los registros, buses de datos, ALU y las palabras de memoria, a N^2 bytes.

b) Diseñe el assembly del computador. Cada instrucción debe estar asociada a un opcode y estos a sus respectivas señales de control **(3 ptos.)**
Solucion: En base al diagrama anterior, basta utilizar el assembly del computador básico. La única diferencia se da en los literales, donde existen dos opciones: i) separar cada uno de los elementos de una matriz con coma (,) y ii) declarar la matriz completa como un único número de tamaño N^2 bytes.

c) **Bonus**: 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. **(3 ptos.)**
Solución: Una posible solución es agregar unidades de shifting para aislar el elemento buscado y modificarlo y luego realizar operaciones lógicas para integrar este valor con la matriz completa.

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



IIC2343 – Arquitectura de Computadores (I/2017)

Solución Interrogación 1

Pregunta 1

a) ¿Qué número entero es generado al realizar cuatro operaciones **shift right** seguidas de cinco operaciones **rotate left** a un registro de 8 bits que inicialmente almacena el número entero 79? **(1 pto.)**
Solución: 79: `01001111` $\xrightarrow{4xSHR}$ `00000100` $\xrightarrow{5xRTL}$ `10000000` :-128

b) ¿Cuál es la cantidad máxima de pixeles que puede tener una imagen en blanco y negro de 1KB no comprimida? Asuma que cada pixel solo almacena su valor de color y que el archivo de la imagen sólo almacena pixeles, por lo que no es necesario considerar el encabezado. **(1 pto.)**
Solución: 1 pixel = 1b (0: blanco, 1: negro) → 1KB = 1024*8b = 8192b → 8192 pixeles

c) Sea P, el conjunto de representaciones de punto flotante de $s + e + 2$ bits, con s bits para el significante normalizado, 1 bit para el signo de este, e bits para el exponente (no desplazado) y 1 bit para el signo de este.

Considere ademas x y $\tilde{x}$, números pertenecientes a $\mathbb{Z}$, ambos codificados usando una representación perteneciente a P, tales que:

- $sucesor(x) \neq x + 1$
- $\forall \tilde{x} < x, sucesor(\tilde{x}) = \tilde{x} + 1$

Donde $sucesor(y)$ es una función que retorna el siguiente número entero mayor que y (el sucesor de y).

En base a esto, responda las siguientes preguntas:

i. Indique cuál es el valor de x en función de los parámetros s y e y muestre que dado x, siempre existe un número $x' \in \mathbb{Z}$, codificado en la misma representación que x, tal que $x' + 1 = x$. **(1 pto.)**
Solución: `x` corresponde sencillamente a aquel número donde la representación obliga a realizar un salto de a 2 si se quiere seguir aumentando lo registrado, lo cual ocurre cuando ya no se pueda registrar el bit menos significativo del número entero (si hay parte decimal, aún quedan bits por descartar sin incurrir en un salto no unitario).
De lo anterior, se ve claramente que $x \propto s$. Además, considerando que la notación de punto flotante implica omitir el bit más significativo, el cual marca 1, se tiene inmediatamente:

$$x = 2^{s+1} = \underbrace{0}_{\text{se}}\overbrace{bin(s+1)}^{\text{e}}\underbrace{0}_{\text{ss}}\overbrace{zero(s)}^{\text{s}}$$

En cuanto a la demostración, basta con considerar que al encontrarse aquel x por definición se tiene la función sucesor, la cual en particular dispone de un $\tilde{x}$ tal que $sucesor(\tilde{x}) = x \equiv \tilde{x} + 1 \longleftrightarrow x - 1 = \tilde{x}$, que es lo que se pedía mostrar.

ii. ¿Existe un número con las características de x en el estándar IEEE754 de 32 bits? En caso positivo, indique su valor, y en caso negativo indique por qué no es posible encontrar este número. **(1 pto.)**
Solución: Adaptando directamente la respuesta obtenida en la *parte i*, se tiene que:

$$x_{IEEE754(32b)} = 2^{23+1} = \underbrace{0}_{\text{s}}\overbrace{bin(23+1+127)}^{\text{e=8}}\overbrace{zero(23)}^{\text{s=23}}$$

iii. Caracterice, en función de los parámetros s y e, el conjunto de representaciones de punto flotante $\tilde{P} \subset P$, tales que no contienen un número con las características de x. **(2 ptos.)**
Solución: Puede deducirse fácilmente de la *parte i* que si no es posible representar $bin(s+1)$ en el exponente, la representación sencillamente no llegará hasta el punto de existencia de x. Luego, $\tilde{\mathbb{P}}$ es el conjunto de aquellas representaciones donde $2^s - 1 \geq e$.

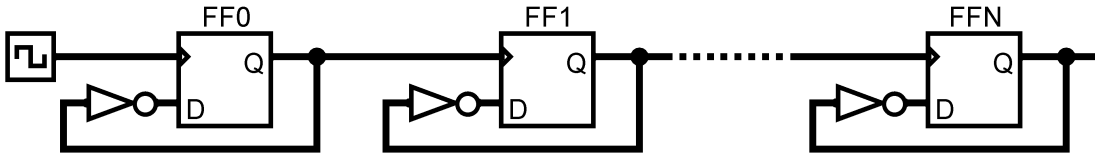
Pregunta 2

a) Dado un número entero n de 32 bits almacenado en una memoria con palabras de 1 byte, determine el valor de n tal que el error en valor absoluto es máximo si se confunde el endiannes del número al ser leído e interpretado. **(1 pto.)**
Solución: Se espera mostrar uno de los 2 valores en los cuales el error en valor absoluto es máxima, considerando 2 casos:

- El alumno entendió error en valor absoluto como $|a - b|$.
- El alumno entendió error en valor absoluto como $|a| - |b|$.

Junto a esto también escribir una pequeña explicación de por qué el error generado es máximo.

b) En la siguiente figura, si la frecuencia del clock que entra al *flip-flop* **FF0** es **F** Hz, ¿cuál es la frecuencia del clock del *flip-flop* **FFN**? **(1 pto.)** **Solución:** Mostrar el valor de la frecuencia del clock en el flip-flip



N ($\frac{F}{2^N}$) junto con la razón de este valor (cada vez que se pasa al otro flip-flop la frecuencia disminuye a la mitad).

c) Construya un circuito que permita detectar la ocurrencia de *overflow* al sumar o restar dos números enteros de 8 bits en una ALU. **(2 ptos.)**
Solución: Se debían mostrar los 4 casos posibles en los que ocurría overflow (2 al sumar y 2 al restar). Lo más sencillo que se esperaban era que construyeran un circuito para cada caso y después los unieran con OR's.

d) Diseñe una memoria RAM que permita acceder (lectura y escritura) de manera individual a cada uno de los bits de una palabra. Tenga en consideración que los buses de datos de entrada y salida deben mantener su tamaño de una (1) palabra. **(2 ptos.)**
Solución: Se esperaba al menos los siguiente:

- Agregar una nueva entrada a la RAM (de tamaño log (tamañodelapalabra) o del tamaño de la palabra) para seleccionar el de la palabra de memoria bit con el que se operará.
- Crear mecanismos para la lectura y escritura de palabras de memoria en la RAM (utilización de Mux's y Demux's). Se puede tanto diagramar la nueva RAM o explicar detalladamente su funcionamiento.

Pregunta 3

a) Indique cuál es la velocidad máxima que puede tener el clock del computador básico, en función de la propagación de la señal al interior de este. Mencione todos los componentes involucrados. **Hint**: considere qué elementos participan en el intervalo entre flancos de subida y como estos se relacionan. **(1 pto.)**
Solución: Se esperaba considerar los diferentes caminos que recorre el flujo de las señales al interior del computador básico en una operación, mencionando tanto los buses que van a la memoria de instrucciones, unidad de control, muxs, ALU, etc.. como los buses que van a la memoria de datos, registros, etc.

b) Si se elimina la instrucción **CMP** del computador básico, ¿cómo deben modificarse las instrucciones de salto, sin alterar el hardware, para que estas no dependan del resultado de la última instrucción ejecutada? Escriba detalladamente todas las modificaciones necesarias y sus implicancias. Asuma que sólo es necesario resolver el caso de la comparación de los registros **A** y **B** y que no es posible sobreescribir los registros para realizar la comparación. **(2 ptos.)**
Solución: Incluir en la instrucción de salto la operación del CMP. Esto implica que la nueva instrucción de salto se compone de dos opcodes y, por ende, dos ciclos. El primero que hace la resta entre los registros A y B y no guarda el resultado, sino que solo se actualizan los flags del status. El segundo que hace lo mismo que la instrucción de salto sin modificaciones.

c) Diseñe, mediante un diagrama, un computador que soporte de manera **nativa** el uso de números complejos ($\mathbb{C}$). Detalle el funcionamiento de este computador e indique la manera en que los datos son codificados y almacenados. Considere que este computador soporta control de flujo, subrutinas y las mismas ocho operaciones aritméticas del computador básico. En relación a estas últimas, se espera que no tomen más de un ciclo de clock. **(3 ptos.)**
Solución: Señalar la necesidad de soportar 2 dimensiones de un número (2 registros para cada número, y el doble de espacio de uso por cada número en la memoria de datos y algunos buses de datos). Cada dimensión como un número perteneciente a R (Reales), por lo que se necesita definir alguna representación de tipo float y soporte de operaciones con este tipo de números (FPU).

Arquitectura de Computadores – IIC2343

I1

14 septiembre 2017

1. Complemento de 2

a) [**1 pt.**] ¿Cómo se detecta *overflow* después de una operación aritmética?, tanto para números positivos como para números negativos. Justifica.

El resultado es del signo opuesto a los operandos, o, equivalentemente, el bit de signo es incorrecto: un 1 (a la izquierda) cuando el número es positivo; un 0, cuando el número es negativo.

El número positivo de mayor valor es 011…1; si sumamos 1 a este número, obtenemos 100…0, pero éste es un número negativo en complemento de 2

b) [**1 pt.**] Sea x un número binario de 8 bits; y sea $\tilde{x}$ una secuencia de 8 bits tal que cada bit de $\tilde{x}$ es la negación del correspondiente bit de x; p.ej., si x = 01101001, entonces $\tilde{x}$ = 10010110. ¿Cuál es la relación aritmética/algebraica entre $-x$ y $\tilde{x}$?

Para negar un número x *en complemento de 2, negamos cada bit —* ~x *— y a este valor le sumamos 1. Es decir,* –x = ~x + 1 => x + ~x = –1. (*Otra manera de ver esto, es que a partir de esta última ecuación se deduce la "regla" para determinar* –x *a partir de* x *en complemento de 2.*)

c) [**1 pt.**] Sea x un número binario de 8 bits; ¿cómo se lo lleva a 16 bits?, tanto para números positivos como para números negativos. Justifica.

Para números positivos (el bit de más a izquierda es 0), simplemente rellenamos con ocho 0's; es (más o menos) obvio que el valor numérico no cambia.

Para números negativos (el bit de más a la izquierda es 1) … rellenamos con 1's ! Esto se puede comprobar, ya que en el número original (de 8 bits) el 1 de más a la izquierda se interpreta como $1 \times -2^7 = -128$*; en cambio, en el nuevo número (de 16 bits) este mismo 1 se iterpreta como* $1 \times 2^7 = 128$*. Si a este valor sumamos el valor de los nuevos ocho 1's a su izquierda (recordando que el nuevo 1 de más a la izquierda ahora vale* 1×-2^{15} *), obtenemos nuevamente –128. Es decir, el valor numérico de los 8 bits originales* ***no cambia*** *al agregar ocho 1's a la izquierda.*

d) [**3 pts.**] He aquí la multiplicación de 1000_{10} por 1001_{10} :

multiplicando		$1\,0\,0\,0_{10}$
multiplicador	×	$1\,0\,0\,1_{10}$
		$1\,0\,0\,0$
		$0\,0\,0\,0$
		$0\,0\,0\,0$
		$1\,0\,0\,0$
producto		$1\,0\,0\,1\,0\,0\,0_{10}$

Es decir, tomamos los dígitos del multiplicador uno a uno de derecha a izquierda, multiplicamos el multiplicando por el dígito del multiplicador, y desplazamos el producto intermedio un dígito a la izquierda con respecto al producto intermedio anterior. Cuando los dígitos involucrados son solo 0 y 1 (como en el ejemplo y como en el caso de los números binarios), vemos que cada paso del proceso es simple:

i) Escribir una copia del multiplicando en la posición correcta si el dígito del multiplicador es 1; o
ii) Escribir 0 en la posición correcta si el dígito es 0.

Dibuja el diagrama de bloques del hardware para multiplicación de números de 32 bits. Inicialmente, el multiplicando y el multiplicador están en registros , y usamos un tercer registro para el producto, inicializado en 0; además, contamos con un sumador y con una unidad de control que controla a cada registro y al sumador. Explica tu diagrama en función del algoritmo de multiplicación sugerido.

Además, responde y justifica: Para cada uno de los tres registros, ¿cuántos bits tiene que tener? y ¿qué operación tiene que poder hacerse en el registro?

El registro del multiplicando debe tener la capacidad de hacer shift a la izquierda sin perder sus bits originales —es decir, con cada shift multiplica su valor por 2— por lo que debe tener 64 bits; la señal de cuándo hacer shift a la izquierda proviene de la unidad de control.

Este valor del multiplicando ingresa como uno de los dos operandos al sumador de 64 bits, cuando el bit que se está mirando del multiplicador es 1 (por i). En cambio, cuando el bit que se está mirando del multiplicador es 0, entra un 0 al sumador (por ii). Esto significa que debe ser posible mirar uno por uno los bits del multiplicador, de derecha a izquierda. Por lo tanto, el registro del multiplicador debe tener la capacidad de hacer shift a la derecha y "enviar" a la unidad de control su bit de más a la derecha (que va cambiando con cada shift). La señal de cuándo hacer shift a la derecha proviene de la misma unidad de control. Este registro puede ser de solo 32 bits.

La unidad de control, cada vez que recibe el bit de más a la derecha del multiplicador (bit que va cambiando con cada shift a la derecha del registro del multiplicador), decide si el valor del multiplicando o un 0 debe ingresar como operando al sumador.

Finalmente, la salida del sumador va al registro de 64 bits que almacena el producto; por lo tanto, este registro debe tener no solo 64 bits, sino también la capacidad de ser escrito. La señal de cuándo este registro debe actua-lizar su valor según la salida del sumador proviene también de la unidad de control. Además, el valor de este registro es a su vez el otro operando del sumador.

2. Considera el siguiente diagrama de bloques del computador básico, aún incompleto.

a) Explica el rol del multiplexor *Address*; da un ejemplo de su funcionamiento

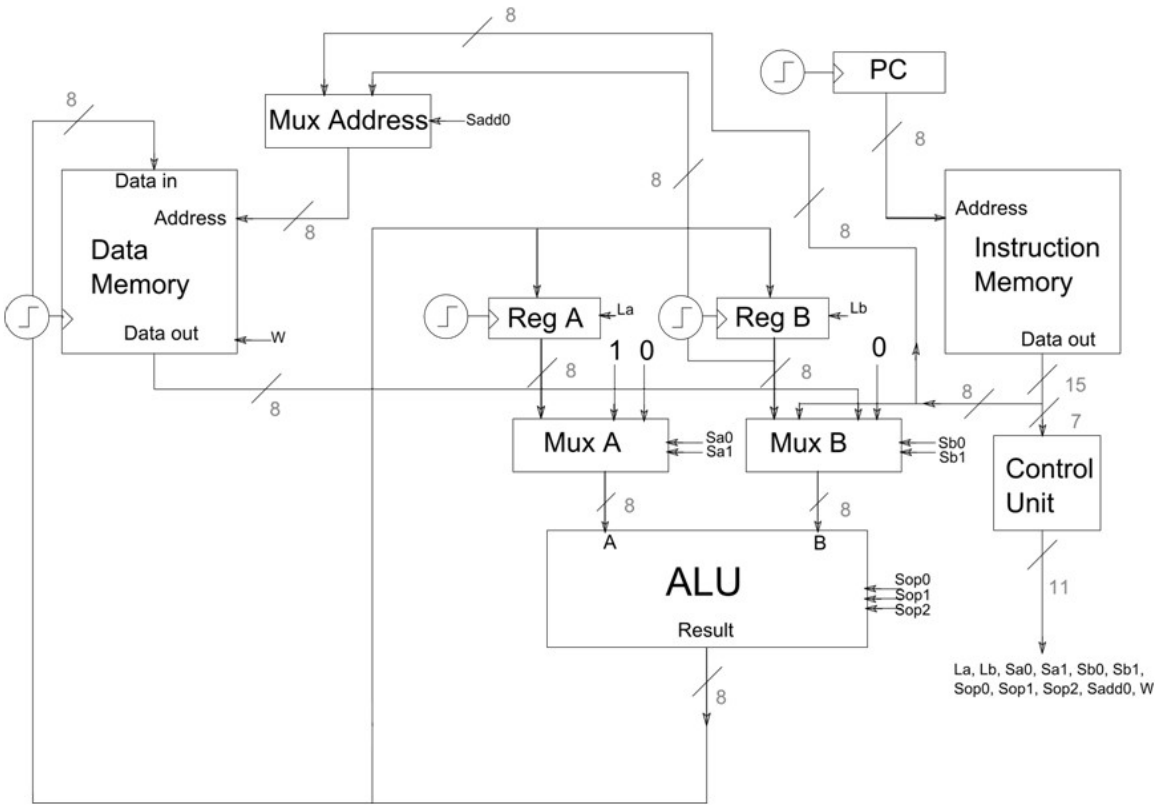
Decidir cuál dirección se "ingresa" a la memoria: la que viene especificada como literal en la instrucción (desde la instruction memory*) o la que está almecenada en el registro B.* ***Falta el ejemplo.***

b) Explica qué es necesario agregar para permitir instrucciones del tipo saltos incondicionales; y explica cómo funcionaría en este caso un salto incondicional.

Para saltos incondicionales es necesario poder poner en el registro PC *dirección de la instrucción a la cual se quiere ir: hay que conectar la salida de la* Instruction Memory *a la entrada del* PC*, y controlar con una señal de la* Control Unit *esta actualización del* PC *cuando la instrucción sea un salto.*

c) Explica qué es necesario agregar a tu respuesta a b) para permitir instrucciones del tipo saltos condicionales; y explica cómo funcionaría en este caso un salto condicional.

Es necesario poder evaluar una condición (aritmética) antes de decidir si se salta o no. El resultado de la evaluación de la condición (hecho en la ALU*) se codifica en varios bits, p.ej.,* N (negative) *y* Z (zero)*, que se cargan en un nuevo registro* Status. *La* Control Unit *lee el valor de estos bits al momento de ejecutar un salto condicional.*



3. Considera el código en *assembly* que se muestra a la derecha. **a**) Explica el propósito de, y cómo funciona, la secuencia de instrucciones PUSH B, CALL *func* y POP B; y cuál es el efecto de que no haya una instrucción PUSH A (justo antes) ni una instrucción POP A (justo después). **b**) Explica el propósito de, y cómo funciona, la secuencia de instrucciones CMP A,0 y JEQ *end*, y en particular qué ocurre cuando el contenido del registro A es 0. Tanto en a) como en b), identifica las componentes del computador básico que están involucradas y explica qué rol juegan.

```
DATA:
   r 3

CODE :
   MOV  A, ( r )
   MOV  B, 2
   PUSH  B
   CALL  func
   POP  B
   JMP  f i n i s h

finish:
```

```
func :
      s h i f t :
            MOV  A,B
            CMP  A, 0
            JEQ end
            DEC  A
            MOV  B,A
            MOV  A, ( r )
            SHL  A,A
            MOV  ( r ) ,A
            JMP  s h i f t
      end :
            RET
```

a) El propósito es ejecutar la subrutina *func*, pero de modo que el valor del registro *B* al terminar la ejecución de *func* y volver al programa principal (CODE) sea el mismo que *B* tenía justo antes de hacer la llamada; es decir, las actualizaciones del registro *B* al interior de *func* no se ven reflejadas en CODE. Esto ocurre porque el valor de *B* justo antes de la llamada es guardado en el stack, mediante PUSH B, y es restaurado justo después de la llamada mediante POP B.

Por esto mismo, la ausencia de las instrucciones PUSH A y POP A tiene el efecto contrario en el caso del registro *A*: los cambios que el registro *A* sufre dentro de *func* se ven reflejados en CODE cuando *func* termina su ejecución.

Las instrucciones PUSH y POP acualizan el valor del *stack pointer* —justo después y justo antes, respectivamente— de tener acceso a la dirección de memoria apuntada por el *stack pointer*.

b) El propósito es terminar la ejecución de *func* cuando el valor del registro *A* sea igual a 0. La instrucción COMP A, 0 ejecuta *A*–0 (resta 0 al contenido del registro *A*); el resultado no es almacenado sino procesado, de modo que solo si es 0 (todos los bits del resultado son 0), entonces el bit *Z* del registro *Status* queda en 1. Luego, JEQ *end* salta a la instrucción identificada con la etiqueta *end* (coloca en el registro *PC* la dirección de esta instrucción) si el bit *Z* es 1.

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.)**

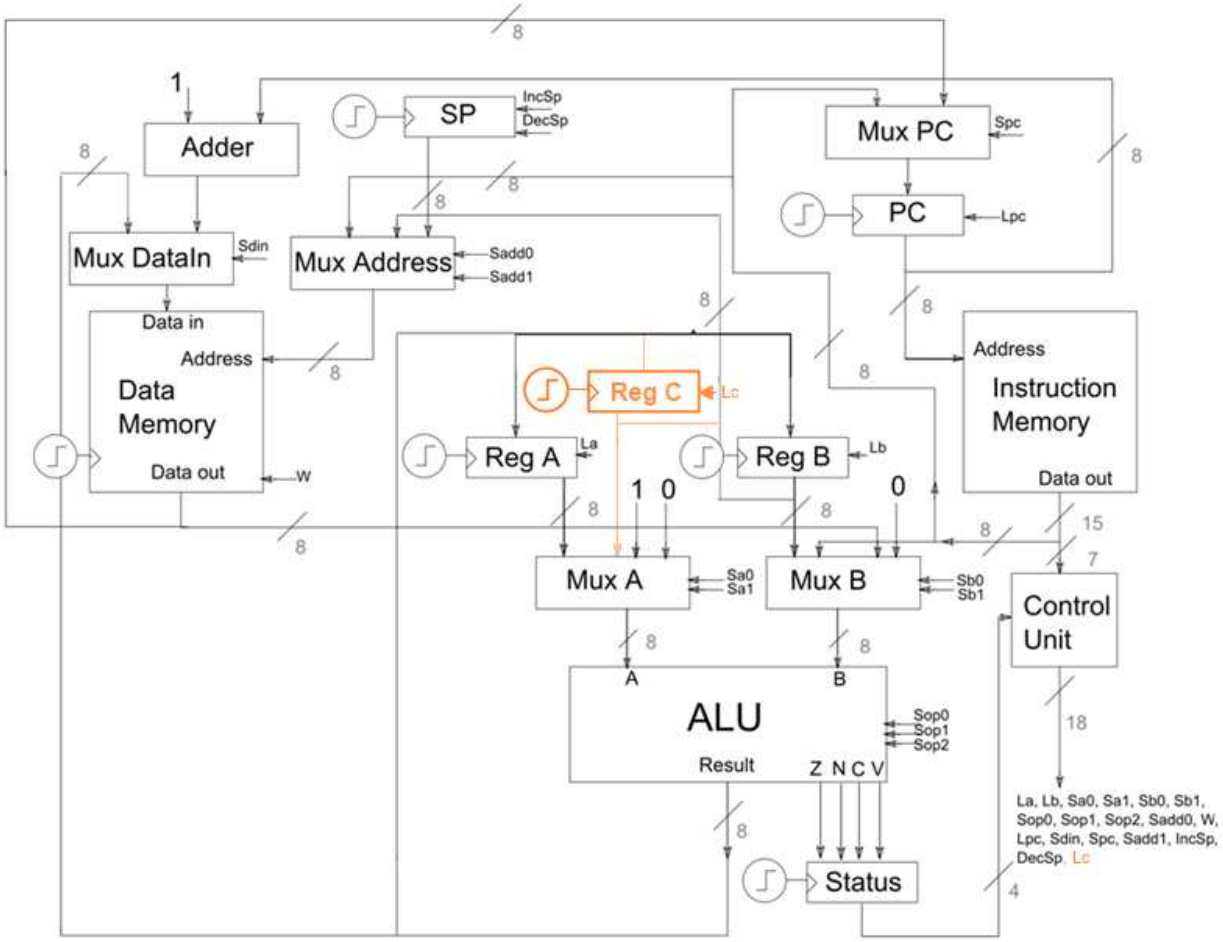
Solución: Estos ataques pueden realizarse debido a la incapacidad de las arquitecturas Von Neumann de diferenciar instrucciones de datos. Una posible solución es modificar la memoria principal, de manera que cada posición de memoria, además de contener una palabra, contenga además un bit que indique 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.

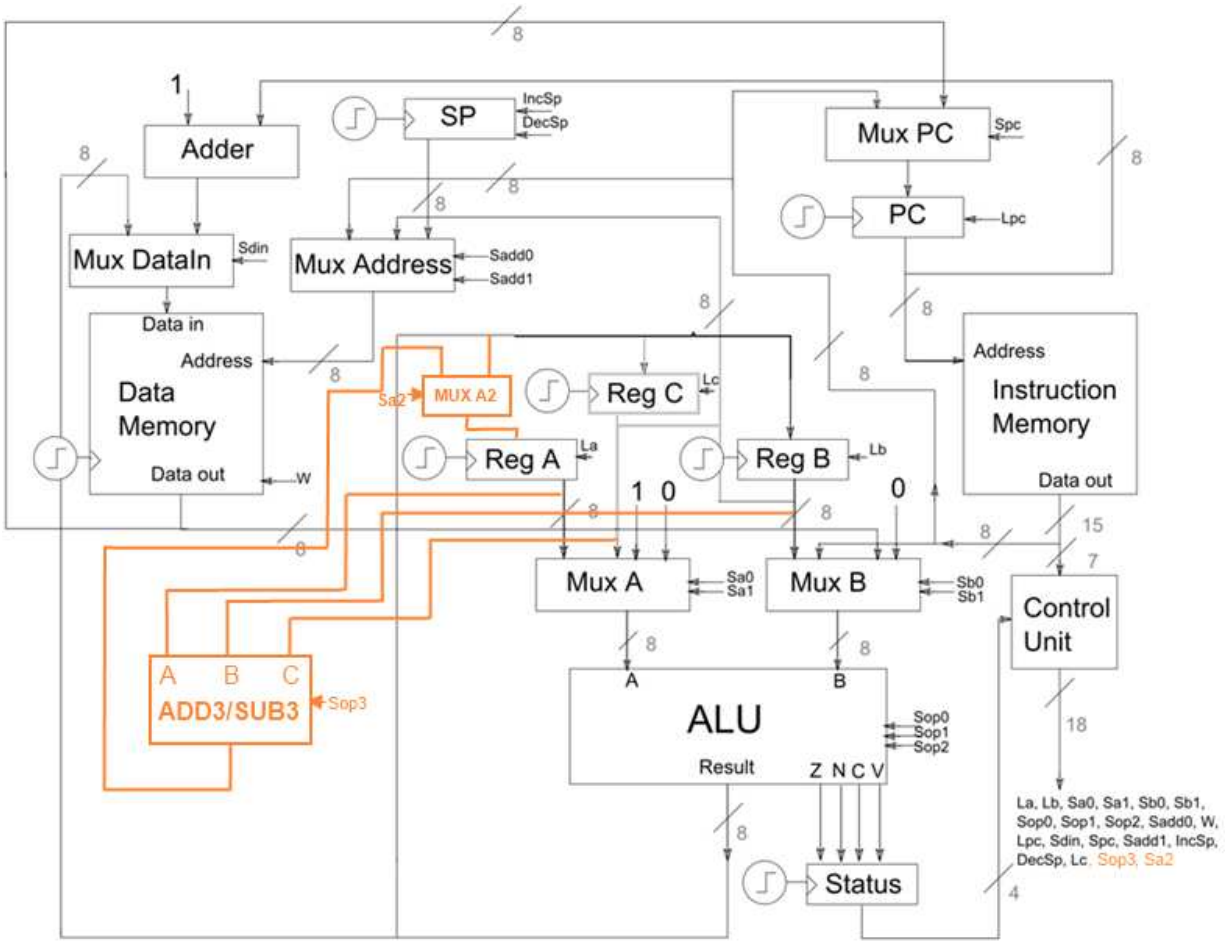
a) Agregar un tercer registro, que cumpla la función de acumulador de resultados y que pueda usarse para direccionamiento indirecto. **(1,5ptos.)**

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



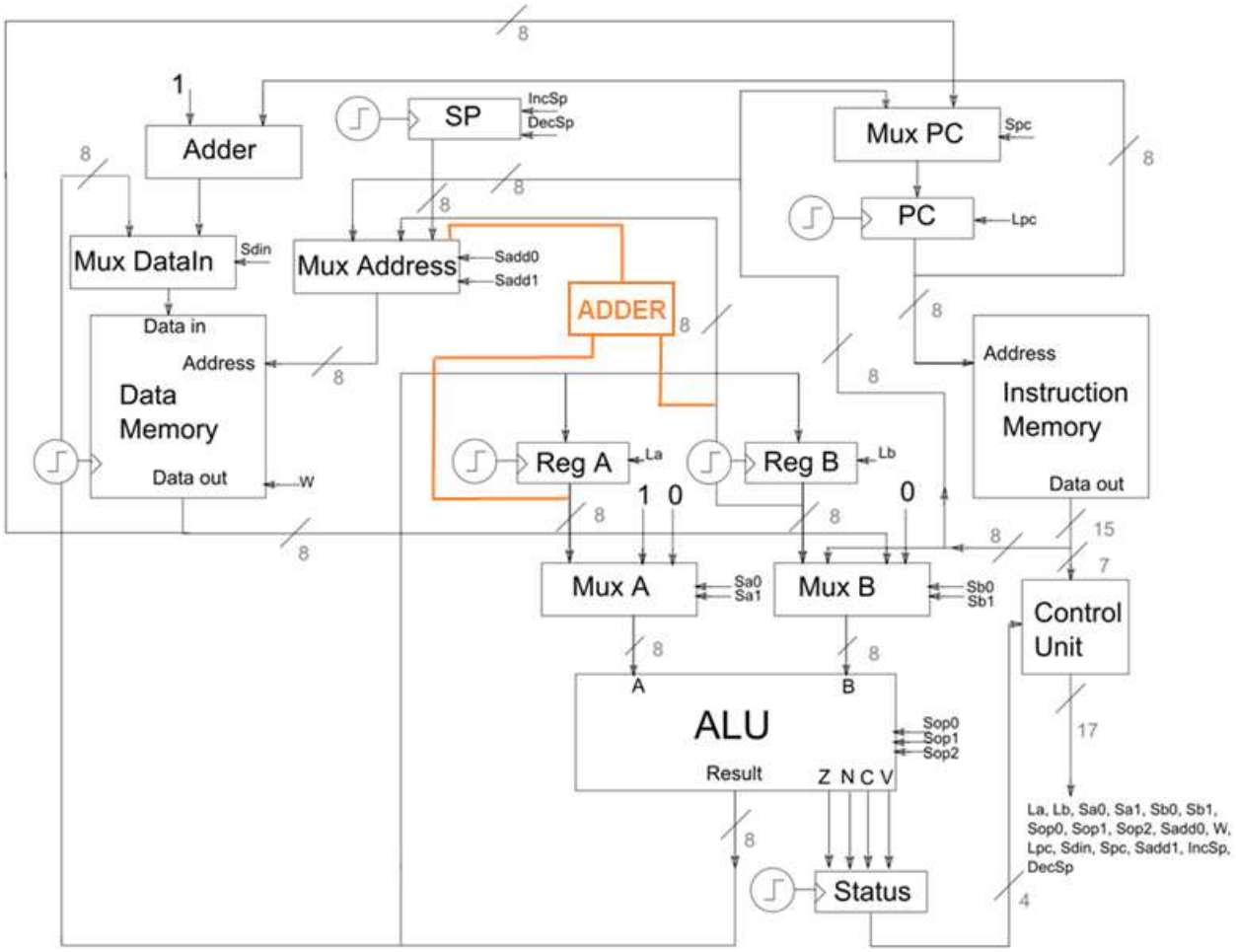
b) Agregar las instrucciones ADD3 reg1, reg2, reg3 y SUB3 reg1, reg2, reg3, que toman los valores en los registros reg1, reg2 y reg3, los suma/resta y los almacena en reg1. **(1,5 ptos.)**

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.



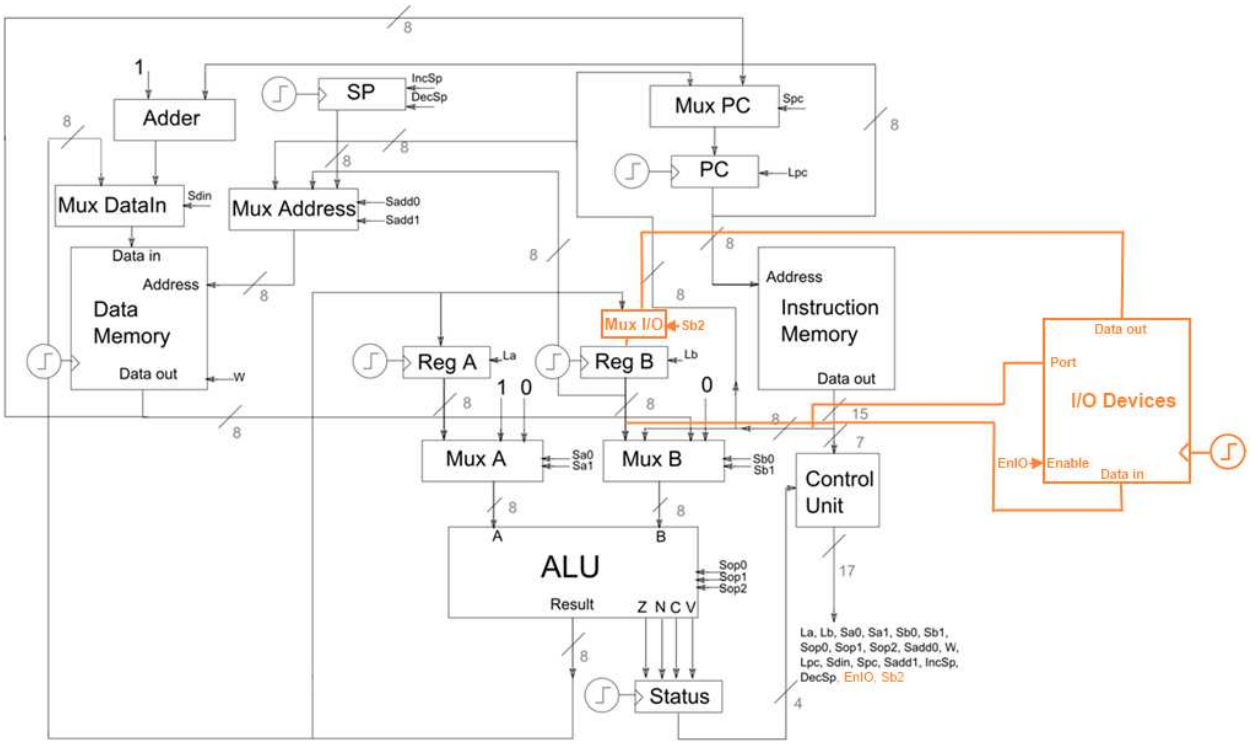
c) Soportar direccionamiento indirecto con registro base y registro índice. **(1,5 ptos.)**

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.



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



IIC2343 – Arquitectura de Computadores (I/2012)

Interrogación 2

Pregunta 1

a) Si se elimina el condition code Z del registro Status, ¿que otro posible condition code podría crearse para ser usado en conjunción con N, de manera de emular el comportamiento de Z? **(0,75 ptos.)**

Solución: Dado que se tiene el condition code N, un condition code que podría ser usado en conjunción con este sería el condition code M, que es uno cuando el resultado de la última operación de la ALU es mayor que cero. De esta manera, si N=0 y M=0, entonces el número en la ALU no es ni positivo ni negativo, por lo que es cero, lo que implicaría Z = 1. Para implementar M basta con hacer un AND de los 8 bits de la ALU, donde a diferencia del condition code Z, el bit más significativo llega negado al AND.

b) Indique la diferencia en el funcionamiento del registro SP entre el computador básico y la arquitectura x86. **(0,75 ptos.)**

Solución: En el computador básico, el SP apunta siempre a la posición anterior en la memoria memoria del último dato almacenado en el stack, mientras que en x86, el SP apunta a la posición en la memoria donde está el último dato almacenado en el stack.

c) Algunas instrucciones de la ISA del computador básico utilizan 2 ciclos para ejecutarse. Indique cuáles son y explique por qué ocurre esto. **(0,75 ptos.)**

Solución: Las instrucciones que toman 2 ciclos son POP y RET. Esto ocurre debido a que es necesario hacer en ciclos distintos la actualización del SP y la lectura de la memoria de datos.

d) ¿Por qué no es necesario dividir en segmentos DATA y CODE un programa escrito en assembly x86? **(0,75 ptos.)**

Solución: Porque en x86, datos e instrucciones van en la misma memoria, por lo que todo lo relacionado con un programa puede guardarse de forma contigua en un bloque de memoria.

e) Explique como genera un mouse mecánico una señal digital a partir del movimiento de la esfera. **(0,75 ptos.)**

Solución: El movimiento de la esfera genera la rotación de 2 ruedas perpendiculares, correspondientes a los ejes x e y. Esta rotación interrumpe la trayectoria de 2 haces de luz por cada rueda, con lo que se puede estimar el sentido y velocidad del movimiento en cada eje, dado un tiempo fijo de muestreo temporal, contando la cantidad de veces que se interrumpieron los rayos y cual de estos fue interrumpido primero.

f) ¿Qué consideración se debería tener en el computador básico, si todos los buses se extienden a 32 bits con la excepción del bus de direcciones? **(0,75 ptos.)**

Solución: Se debe tener la consideración de truncar los números que entren al bus de direcciones, tomando los 8 bits menos significativos.

g) ¿En que consite la ley de Moore? ¿Se ha cumplido esta ley a lo largo del tiempo? **(0,75 ptos.)**

Solución: La ley de Moore afirma que la cantidad de transistores en un microprocesador se duplicará cada 2 años. Esta ley se ha cumplido casi a la perfección a lo durante los último 40 años.

h) Reescriba el siguiente código Java usando el assembly del computador básico. **(0,75 ptos.)**

```
public static void func()
{
    byte a = 64;
    if (a > 0)
    {
        a >> 1
    }
    else
    {
        a = 1;
    }
}
```

Solución:

```
DATA:
a    0x40
CODE:
MOV A, (a)
CMP A, 0
JLE else
SHR A
MOV (a), A
JMP end
else:
MOV A, 1
MOV (a), A
end:
```

Pregunta 2

a) Escriba en el assembly del computador básico un programa que calcule la división entre 2 números enteros, a y b, mediante la llamada a la subrutina $q = div(a, b)$, que también debe ser implementada. Cualquier detalle de implementación o aspecto que se asuma debe quedar claramente explicado. **(2 ptos.)**
Solución: Una implementación en Java de la división entera es la siguiente:

```
public byte div(byte a, byte b){
    byte res = 0;
    a = a-b;
    while (a >= 0)
    {
        res = res + 1;
        a = a-b;
    }
    return res;}
```

Basandose en esta implementación, el programa en assembly del computador básico queda así:

```
DATA:
a       7
b       5
q
var1
var2
res
CODE:
MOV A, a
MOV (var1), A
MOV A, b
MOV (var2), A
CALL div
MOV A, (res)
MOV (q), A
JMP end
div: MOV A, 0
MOV (res), A
MOV A, (var1)
MOV B, (var2)
SUB A,B
CMP A, 0
JL end_while
while: MOV B, (res)
INC B
MOV (res), B
MOV B, var2
SUB A,B
JMP while
end_while: RET
end:
```

b) Escriba en assembly x86, usando **stdcall**, un programa que calcule un histograma de un conjunto de números enteros no negativos, almacenados en memoria. Para este fin, implemente la función $h = hist(x, nx, nc)$, que calcula un histograma, de nc casilleros, de la muestra de datos x, de tamaño nx, y lo retorna en la dirección de memoria h. Asuma que los números se encuentran el intervalo $[0, 255]$ y que el número de casilleros del histograma siempre será un divisor de 256. Cualquier detalle de implementación o aspecto que se asuma debe quedar claramente explicado. **(4 ptos.)**

Solución: Una implementación en Java del cálculo de un histograma es la siguiente:

```
public byte[] hist(byte[] x, byte nx, byte nc)
{
    byte[] h = new byte[nc];
    byte i, ancho;
    ancho = 256/nc;
    for (i = 0; i < nx; i++)
    {
        h[x[i]/ancho]++;
    }
    return h;
}
```

Basandose en esta implementación y asumiendo que el arreglo h ha sido declarado previamente, el programa en assembly x86 queda así:

```
MOV AX, 0
MOV AL, [nc]
PUSH AX
MOV AL, [nx]
PUSH AX
LEA AX, x
PUSH AX
CALL hist
RET

hist:
PUSH BP
MOV BP, SP
SUB SP, 2

MOV  AX, 256
DIV [BP+8]
MOV CL, AL

MOV [BP-2], 0
for:
CMP [BP-2], [BP+6]
JE end_for

MOV BX, [BP+4]
MOV SI, [BP-2]
MOV AX, [BX+SI]
DIV CL
```

```
        LEA BX, h
        MOV SI, AL
        ADD [BX+SI], 1
        ADD [BP-2], 1
        JMP for
        end_for:

        LEA AX, h
        ADD SP, 2
        RET 6

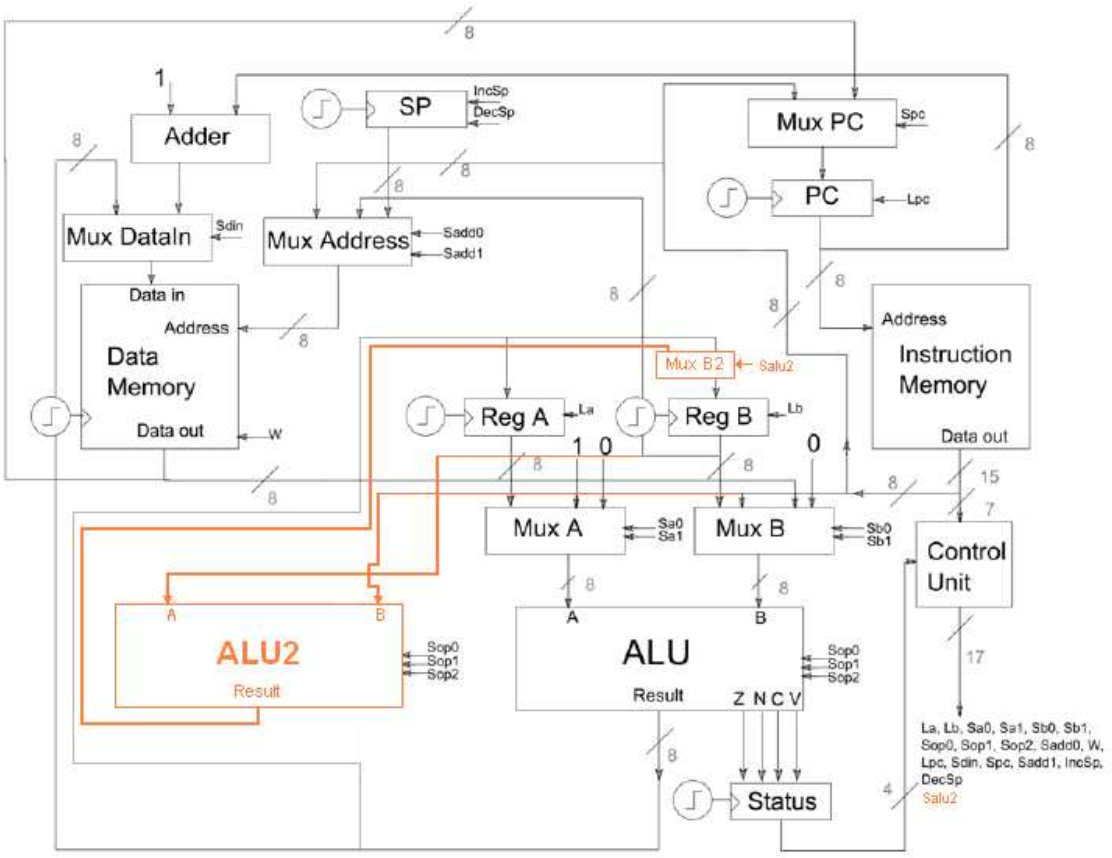
        x       db      11,100,150,13,200
        nx      db      5
        nc      db      4
        h       db      0,0,0,0
```

Pregunta 3

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.

a) Permitir al computador la ejecución de dos operaciones aritméticas-lógicas iguales, pero con distintos argumentos, de manera simultánea, i.e., el proceso debe tomar sólo un ciclo del clock. **(1,5 ptos.)**

Solución: La modificación presentada a continuación agrega 8 nuevas instrucciones al computador, donde cada una de estas corresponde a la versión paralela de las operaciones aritmético-lógicas. Estas instrucciones toman los valores de los registros A y B, los modifican de acuerdo a la operación y al literal ingresado como parámetro, y luego los almacenan nuevamente en los registros A y B. Por ejemplo, la nueva operación ADDP 5, es analogo a ejecutar ADD A, 5 y ADD B, 5. Desde el punto de la microarquitectura, se agrega una ALU adicional, cuyo output se conecta al registro B, por medio del nuevo MUX B2. Además, se agrega la nueva señal $Sb2$, que controla al MUX B2.
La siguiente figura muestra las modificaciones hechas a la microarquitectura:

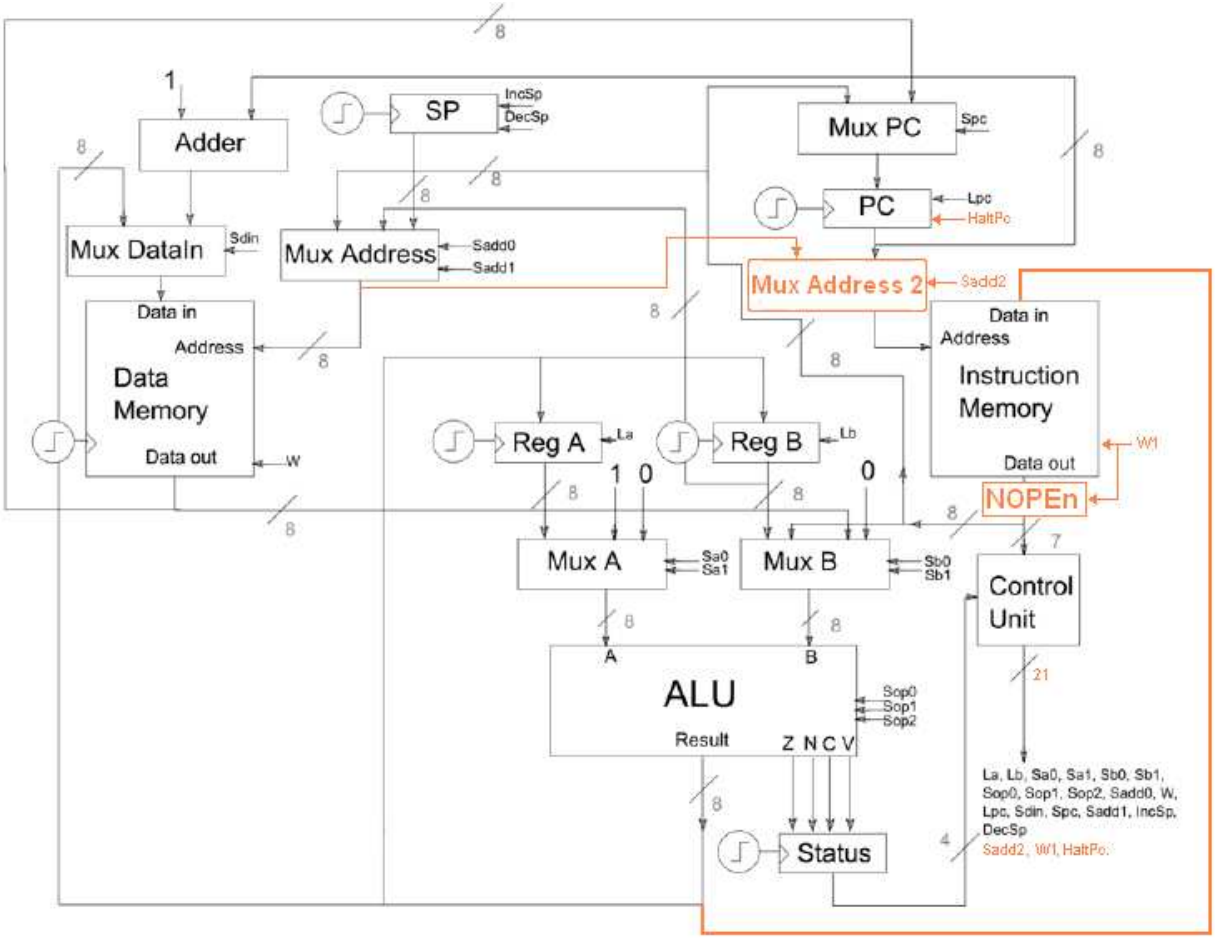


La siguiente tabla presenta los opcodes y señales de control de las nuevas operaciones, excluyendo las sieguiente señales, cuyo valor indicado a continuación se repita de la misma manera en todas las operaciones: Sadd=-, Sdin=-, Spc=-, W=0, IncSp=0 y DecSp=0.

Instr.	Op.	Opcode	Lpc	La	Lb	Salu2	Sa	Sb	Sop
ADDP	Lit	1001110	0	1	1	ALU2	A	LIT	ADD
SUBP	Lit	1001111	0	1	1	ALU2	A	LIT	SUB
ANDP	Lit	1010000	0	1	1	ALU2	A	LIT	AND
ORP	Lit	1010001	0	1	1	ALU2	A	LIT	OR
NOTP	Lit	1010010	0	1	1	ALU2	A	-	NOT
XORP	Lit	1010011	0	1	1	ALU2	A	LIT	XOR
SHLP	Lit	1010100	0	1	1	ALU2	A	-	SHL
SHRP	Lit	1010101	0	1	1	ALU2	A	-	SHR

b) Permitir la autoprogramabilidad manteniendo memorias de datos e instrucciones separadas. **(1,5 ptos.)**

Solución: La modificación realizada cambia la memoria ROM de instrucciones por una memoria RAM, para que pueda ser escrita. Además se añade un mux (Mux Address 2) para elegir el origen de la dirección que ingresa a la memoria, que puede ser el PC o dirección la dirección obtenida de Mux Address. Para asegurar que la instrucción siguiente a la de escritura se ejecute correctamente, se agrega una nueva señal al PC, HaltPC, que evita que este se incremente por una iteración. También se agrega la instrucción NOP, que provoca que el computador no haga nada por un ciclo y se agrega un enabler (NOPEn) a la salida de la memoria de instrucciones, que al ser activado (W1=1) evita que pase la instrucción actual y genera el opcode de la nueva instrucción NOP, mientras que si no es activado (W=0), deja pasar el opcode ingresado. Finalmente, los datos de entrada para esta memoria se obtienen desde la ALU. La siguiente figura muestra las modificaciones hechas a la microarquitectura:



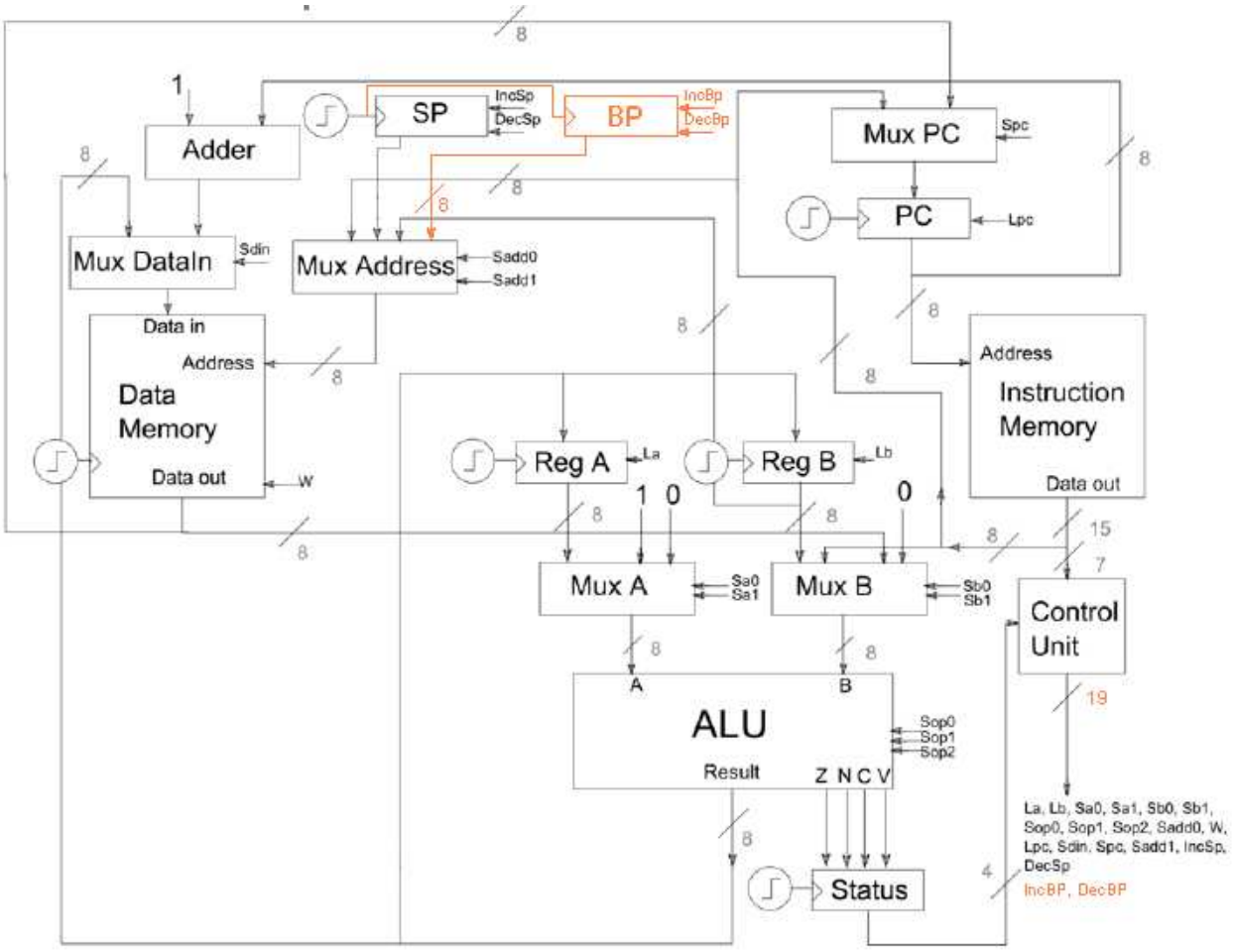
Desde el punto de vista de la ISA, además de NOP, se agregan las instrucciones MOV_DM (Dir), A y MOV_DM (B), A, que escriben el valor almacenado en el registro A en la memoria de datos. Cabe destacar que estas instrucciones toman 2 ciclos, una para escribir el dato en memoria y la siguiente ejecutando NOP La siguiente tabla presenta los opcodes y señales de control de las nuevas instrucciones, incluyendo las nuevas señales Sadd2, HaltPc y W1:

Instr.	Op.	Opcode	Lpc	La	Lb	Sa	Sb	Sop	Sadd	Sdin	Spc	W
NOP	-	1001110	0	0	0	-	-	-	-	-	-	0
MOV_DM	(B), A	1001111	0	0	0	A	0	ADD	B	-	-	0
MOV_DM	(Dir), A	1010000	0	0	0	A	0	ADD	Lit	-	-	0

Instr.	Op.	IncSp	DecSp	Sadd2	HaltPc	W1
NOP	-	0	0	-	1	0
MOV_DM	(B), A	0	0	MuxAddr	1	1
MOV_DM	(Dir), A	0	0	MuxAddr	1	1

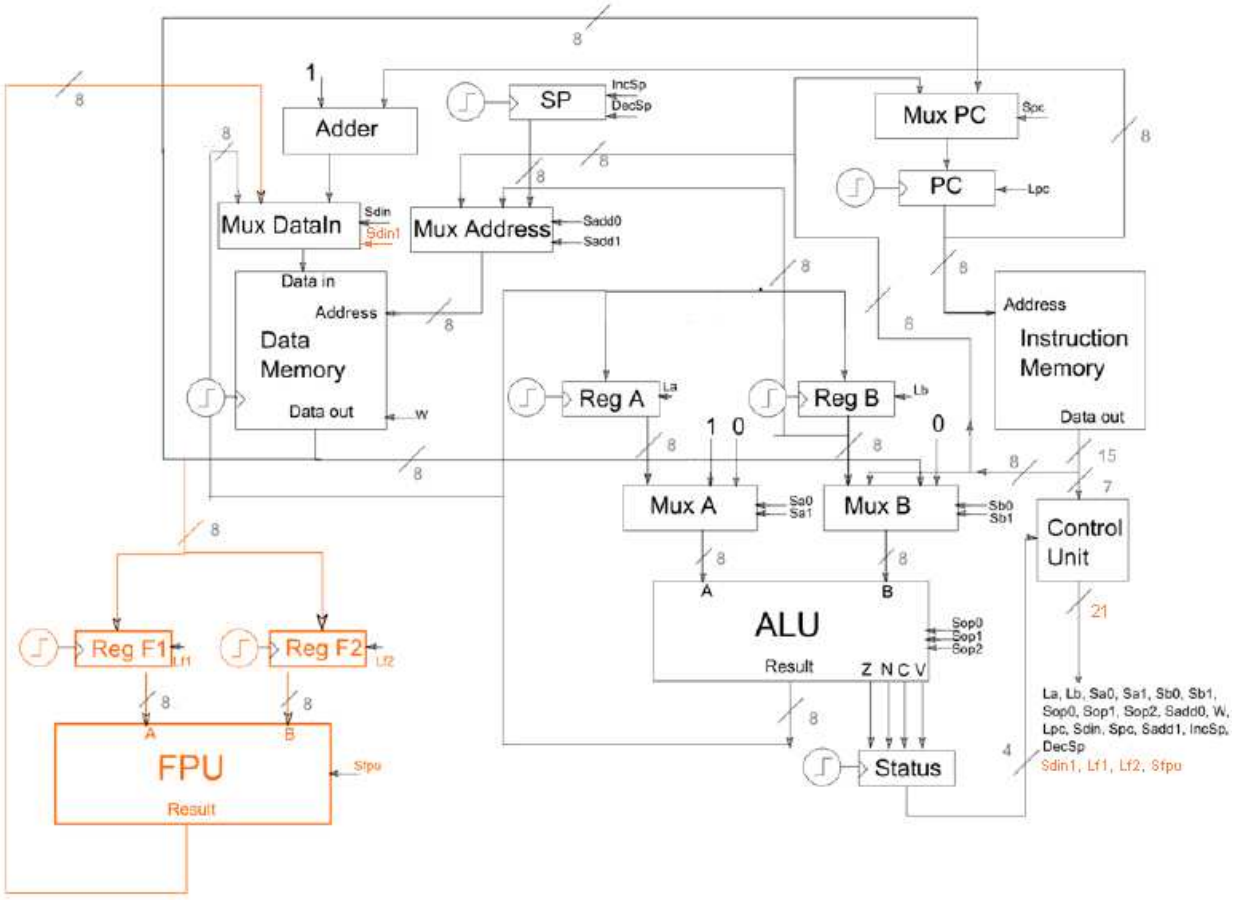
c) (Sólo microarquitectura) Agregar un registro que cumpla las mismas funciones que el BP de la arquitectura x86. **(1,5 ptos.)**

Solución:



d) (Sólo microarquitectura) Agregar una FPU y dos registros dedicados sólo a operaciones de punto flotante. Los resultados de la FPU deben poder almacenarse tanto en estos registros como en la memoria de datos. **(1,5 ptos.)**

Solución:



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



IIC2343 – Arquitectura de Computadores (II/2012)

Interrogación 2

Pregunta 1

a) ¿Por qué motivo no está soportada la instrucción ADD B, LIT en el computador básico? **(1 pto.)**

Solución: Al estar conectado el bus de literales y el registro B al MUX B, no es posible utilizarlos simultáneamente en una operación aritmética o lógica.

b) ¿Aproximadamente cuántos transistores/relés utiliza el computador básico? Justifique sus cálculos y respuesta. **(1 pto.)**

Solución: Basándose en la Ley de Moore, es posible estimar la cantidad de transistores/relés del computador básico, si se asume que este es similar en complejidad al primer procesador de 8 bits de Intel, el 8008. En el gráfico de la Ley de Moore, el primer procesador es el 4004, aparecido en 1971 y con 2300 transistores, por lo que en 18 meses, el mejor procesador debería aumentar sus cantidad de transitores en 2300. Del mismo gráfico se puede estimar que alrededor de 1972 se lanzó el procesador 8008, por lo que el tiempo transcurrido fue de 12 meses, lo que implica un aumento de aproximadamente 1500 transistores. Luego, es posible estimar la cantidad de transitores del procesador 8008, y del computador básico en 3800.

c) ¿Cómo se puede modificar la microarquitectura del computador básico para que este soporte direccionamiento indirecto con registro base + offset? **(1 pto.)**

Solución: Para soportar este modo de direccionamiento, es necesario tomar el bus de salida del registro B y conectarlo con un ADDER, que en su otra entrada recibe el valor del bus de literales. El resultado de este ADDER se agrega como otra entrada al MUX ADDRESS.

d) Describa los tipos de datos soportados nativamente por el computador básico y por la arquitectura x86. **(1 pto.)**

Solución: El computador básico soporta como tipo de dato nativo números enteros y naturales binarios de 8 bits, mientras que la arquitectura x86 soporta número enteros y naturales binarios de 8 y 16 bits. Ninguna de las dos soporta números de punto flotante de forma nativa.

e) Describa y ejemplifique los distintos tipos de **formato de instrucción** utilizados en el computador básico. **(1 pto.)**

Solución: El computador básico posee tres formatos de instrucción distintos:

- Opcode + 2 operandos: Por ejemplo MOV A,B
- Opcode + 1 operando: Por ejemplo PUSH B
- Opcode sin operandos: Por ejemplo RET

f) Describa en detalle la unidad de salto que se encuentra en el interior de la unidad de control del computador básico. **(1 pto.)**

Solución: La unidad de salto se encarga de cargar el registro PC con el valor del bus de literales en caso que se cumpla la condición buscada. La implementación se puede realizar con una serie de caminos lógicos alternativos, uno por cada condición, que entran a un MUX de nueve entradas, donde ocho de estas son para las condiciones y una fijada en 0 lógico. La salida de este MUX va conectada a la señal Lpc del PC, mientras que los bits de control del mux se setean de acuerdo a la instrucción de salto a ejecutar. Las condiciones se pueden agrupar y describir de la siguiente manera:

- JEQ y JNE: Para JEQ, se conecta el bit del registro Status correspondiente al flag Z al MUX. En el caso de JNE se conecta al MUX la negación de este valor.
- JGT y JLE: Para JGT, los bits correspondientes a N y Z son negados y luego conectados a un AND, cuya salida se conecta al MUX. En el caso de JLE, se conecta al MUX la negación de este valor.
- JGE y JLT: Para JGE, el bit correspondiente a N es negado y luego conectado al MUX. En el caso de JLT, se conecta al MUX la negación de este valor.
- JCR: El bit correspondiente a C es conectado al MUX.
- JOV: El bit correspondiente a V es conectado al MUX.

Pregunta 2

En las siguientes preguntas, cualquier detalle de implementación o aspecto que se asuma debe quedar claramente explicado.

a) Escriba en el assembly del computador básico un programa que calcule el n-ésimo número de Fibonacci. **(3 ptos.)**

Solución:

```
DATA:
n ?
index 0
prev1 0
prev2 1
res 0

CODE:
MOV A,(n)
CMP A,0
JEQ end0

while:
MOV A,(n)
MOV B,(index)
CMP A, B
JEQ end
INC B
MOV (index), B
MOV A, (prev1)
MOV B, (prev2)
ADD A, B
MOV (prev1), B
MOV (prev2), A
MOV (res), A
JMP while

end0:
MOV A, 1
MOV (res), A
end:
```

b) Escriba en assembly x86, usando al menos una subrutina y la convención *stdcall*, un programa que calcule la raíz cuadrada natural de un número natural utilizando el método de Newton. Este método aproxima iterativamente el valor de x, tal que f(x) = 0, mediante la siguiente recurrencia: $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$. **(3 ptos.)**
Hint: defina $f(x) = x^2 - n$.

Solución:

```
        MOV AX, 0
        MOV AL, n
        PUSH AX
        CALL newton
        MOV res, AL
        RET

        newton:
        PUSH BP
        MOV BP, SP

        MOV BX, [BP+4] ;BX = n
        MOV CX, BX     ;CX = x_i = n

        do_while:
        MOV AX, BX
        DIV CL         ;AL = n/x_i
        ADD AX, CX     ;AL += x_i + n/x_i
        SHR AL, 1      ;AL = (x_i+n/xi)/2
        CMP AL, CL     ;revisamos si hay cambio
        JE end
        MOV CL, AL     ;x_{i+1} = x_i
        JMP do_while

        end:
        MOV AX, CX
        RET 2

        n      db     ?
        res    db     0
```

Pregunta 3

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

a) Tener stack de uso general e independiente de la memoria de datos. **(3 ptos.)**

b) Dar soporte para la instrucción SUBLEQ A, B, Dir, definida de la siguiente manera:

- Mem[A] = Mem[A] - Mem[B]
- if (Mem[a] $\leq$ 0) $\Rightarrow$ JMP Dir

Su implementación no debe sobreescribir datos en cálculos intermedios. **(2 ptos.)**

c) Aumentar el tamaño del bus de direcciones a 16 bits, manteniendo el resto de los tamaños del computador en 8 bits. Debe ponerse especial énfasis en las instrucciones para direccionar la memoria.**(2 ptos.)**

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



IIC2343 – Arquitectura de Computadores (I/2013)

Interrogación 2

Pregunta 1

a) 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? **(1 pto.)**
Solución: La primera instrucción respalda el valor previamente almacenado en BP, de manera de poder recuparlo si existen llamados a subrutinas anidadas. La segunda instrucción crea un nuevo valor para BP, basado en el actual valor de SP, que indica el tope del stack. Esto permite usar al registro BP como referencia para el direccionamiento de parámetros y variables locales. En caso de no ejecutarse estas instrucciones, no es posible usar al registo BP como referencia, además de no tener seguridad de poder recuperar el valor previo de BP.

b) ¿Qué instrucciones podrían agregarse al computador básico si se agrega al MUX A una entrada para el bus de literales? **(1 pto.)**
Solución: Se podrían agregar instrucciones de transferencia y direccionamiento que involucren al registro B y a un literal de manera simultánea. Esto no era posible antes, ya que el bus de literales y el registro B estaban conectados al mismo MUX.

c) ¿Cómo se puede evitar la sobreescritura accidental de la memoria de datos por el uso del stack en el computador básico? **(1 pto.)**
Solución: Una de las posibles soluciones, es agregar un límite arbitrário al tamaño del stack y al tamaño total usado por las variables. De estas manera, si los tamaño están bien puestos, el tamaño del stack no podrá crecer lo suficiente como para sobreescriba variables.

d) ¿Es posible realizar cálculos con números de punto flotante en el computador básico? **(1 pto.)**
Solución: De manera nativa no es posible hacer cálculos de punto flotante en el computador, ya que no tiene FPU. Para hacer entonces, es necesario emular números de punto flotante mediante software.

e) Enumere y describa las reglas de la convención de llamada *stdcall*. **(1 pto.)**
Solución:

   a) Parámetros son introducidos en el stack de derecha a izquierda.
   b) El registro SP debe quedar apuntando a la misma posición que estaba antes de introducir los parámetros.
   c) El valor de retorno se almacena en el registro AX.

f) Describa situaciones en que se prefieran ISAs RISC sobre CISC y viceversa. **(1 pto.)**
Solución: Una ISA RISC es preferida en situaciones en que el consumo de energía es un aspecto crítico, por lo que se necesita hardware muy simple y de poco consumo. De manera análoga, en situaciones en que lo principal es el rendimiento y el aspecto energético no es el central, se prefiere una ISA CISC.

Pregunta 2

En las siguientes preguntas, cualquier detalle de implementación o aspecto que se asuma debe quedar claramente explicado.

a) Escriba en el assembly del computador básico un programa que calcule el factorial de un número natural n. **(3 ptos.)**

Solución:

```
DATA
n         ?
index     2
i         0
fact      1
mul_res   0

CODE
MOV A, (n)
CMP A, 1
JLE end

while:
MOV A, (n)
MOV B, (index)
CMP A, B
JLT end

MOV A, 0
MOV (i), A
MOV (mul_res), A
mul:
MOV A, mul_res
ADD A, (fact)
MOV (mul_res), A
MOV A, (i)
ADD A, 1
MOV (i), A
MOV B, (index)
CMP A, B
JLT  mul

ADD B, 1
MOV (index), B
MOV A, (mul_res)
MOV (fact), A
JMP while

end:
```

b) Escriba en assembly x86, usando al menos una subrutina, la convención *stdcall* y soporte para variables locales (reg. BP), 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 la parte del arreglo que va desde el elemento central hasta el final. **(3 ptos.)**

Solución: Se asume que el arreglo tendrá un tamaño par y mayor que cero y que el número buscado siempre estará en el arreglo.

```
        MOV AX, 0
        MOV AL, target
        PUSH AX
        MOV AL, n
        PUSH AX
        LEA AX, array
        PUSH AX
        CALL binary_search
        MOV res, AX
        RET

        binary_search:
        PUSH BP
        MOV BP, SP
        MOV BX, [BP + 4]
        MOV SI, [BP + 6]
        MOV CX, [BP + 8]

        start:
        SHR SI, 1
        CMP [BX + SI], CL
        JE end
        JG start
        ADD BX, SI
        JMP start

        end:
        MOV AX, BX
        ADD AX, SI
        POP BP
        RET 6

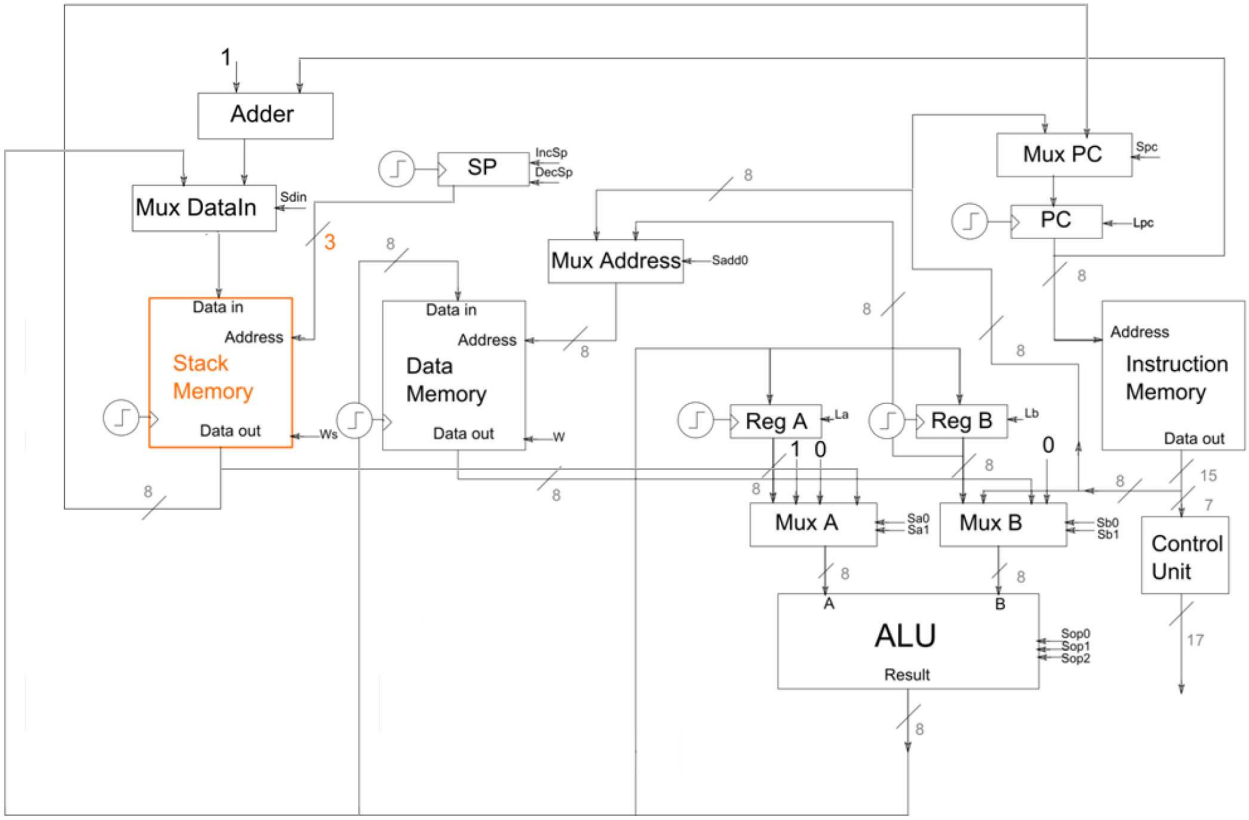
        res    dw
        target db    ?
        n      db    ?
        array  db    ?
```

Pregunta 3

Se desea modificar el computador básico a nivel de microarquitectura e ISA. Para los siguientes puntos detalle las modificaciones que haría. Utilice diagramas de componentes y conexiones y tablas de opcodes e instrucciones cuando corresponda. En caso que sus soluciones utilicen más de un ciclo de clock por instrucción, los valores previamente almacenados en los registros no deben perderse.

a) Tener stack de uso general e independiente de la memoria de datos. **(2 ptos.)**

Solución: A continuación se muestran las modificaciones realizadas a la microarquitectura del computador básico.

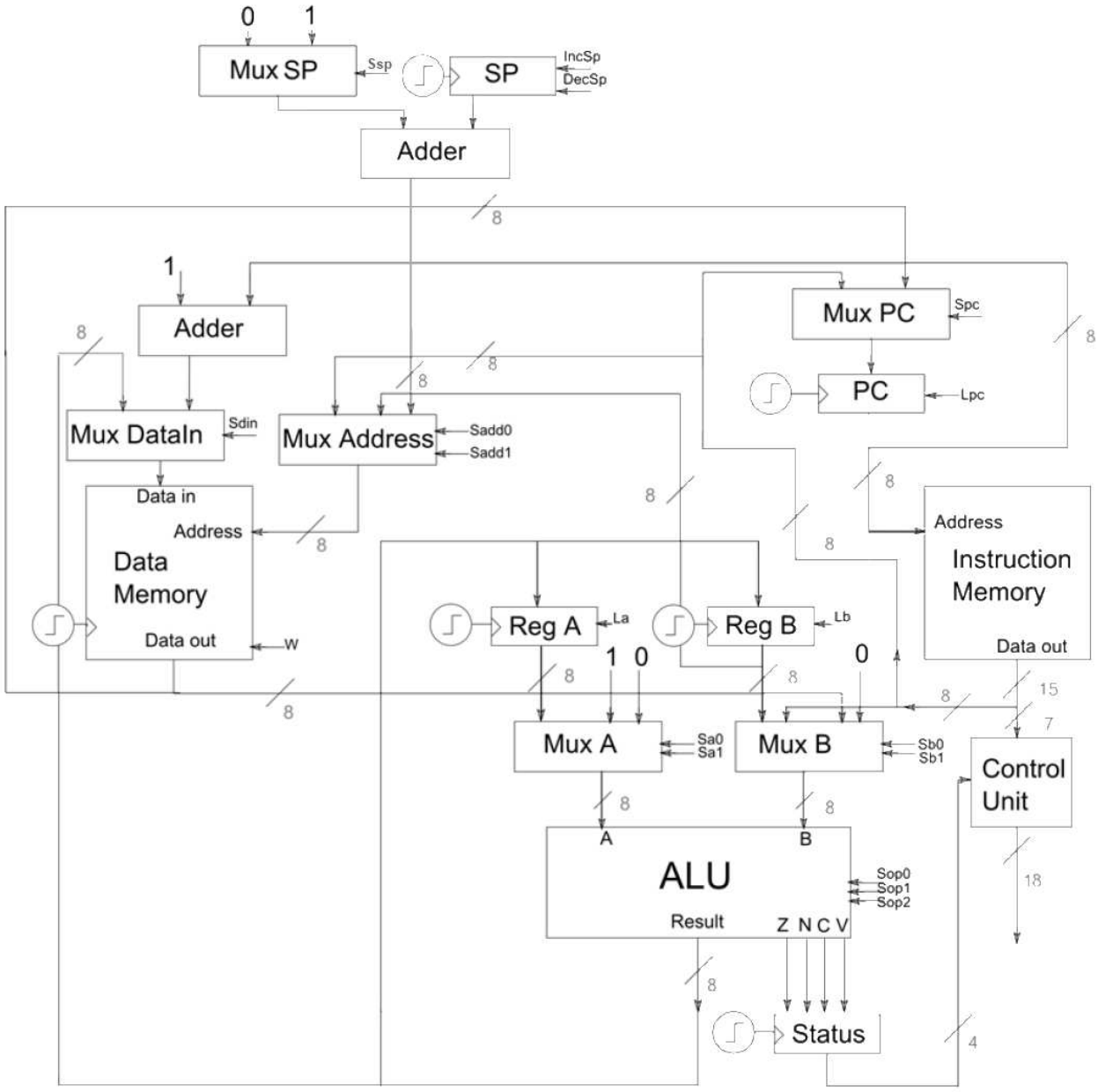


En el diagrama puede apreciarse la eliminación de las señal de control Sdin, que ya no es necesaria, ya que la dirección en el stack vendrá siempre desde el registro SP. Además, se agrega la señal Ws, que controla la escritura en el stack. Para poder soportar la extracción de datos del stack para almacenarlos en los registros, se extiende la linea de salida del stack hasta el MUX A. A nivel de ISA, los cambios necesarios son unicamente la modificación de la señales de control correspodientes a PUSH, POP, RET y CALL. Estas instrucciones deben ahora tomar en cuenta el uso de la señal Ws y la nueva entrada del MUX A, como se muestra a continuación:

Inst.	Op.	Opcode	Lpc	La	Lb	Sa0,1	Sb0,1	Sop0,1,2	Sadd0	Spc	W	Ws	Sdin	IncSp	DecSp
CALL	Dir	1000101	1	0	0	-	-	-	-	LIT	0	1	PC	0	1
RET		1000110	0	0	0	-	-	-	-	-	0	0	-	1	0
		1000111	1	0	0	-	-	-	-	DOUT	0	0	-	0	0
PUSH	A	1001000	0	0	0	A	0	ADD	-	-	0	1	ALU	0	1
PUSH	B	1001001	0	0	0	0	B	ADD	-	-	0	1	ALU	0	1
POP	A	1001010	0	1	0	-	-	-	-	-	0	0	-	1	0
		1001011	0	1	0	DOUT	0	ADD	-	-	0	0	ALU	0	0
POP	B	1001100	0	0	0	-	-	-	-	-	0	0	-	1	0
		1001101	0	0	1	DOUT	0	ADD	-	-	0	0	ALU	0	0

b) Permitir al computador básico la ejecución de todas sus instrucciones en un sólo ciclo del clock. **(2 ptos.)**

Solución: Para permitir la ejecución en un solo ciclo del clock, es necesario asegurar que POP y RET se ejecuten en un ciclo. Para lograrlo, se agrega un Adder, el cual recibe un 1 o un 0 desde el nuevo MuxSP, que tiene una entrada, puesta en 1 y otra en 0, dependiendo si la instrucción ejecutada es alguna de estas últimas dos. De esta manera, se obtiene de manera inmediata el valor correcto desde la memoria y no hay que esperar un ciclo más para hacerlo. A continuación se muestran las modificaciones realizadas a la microarquitectura:

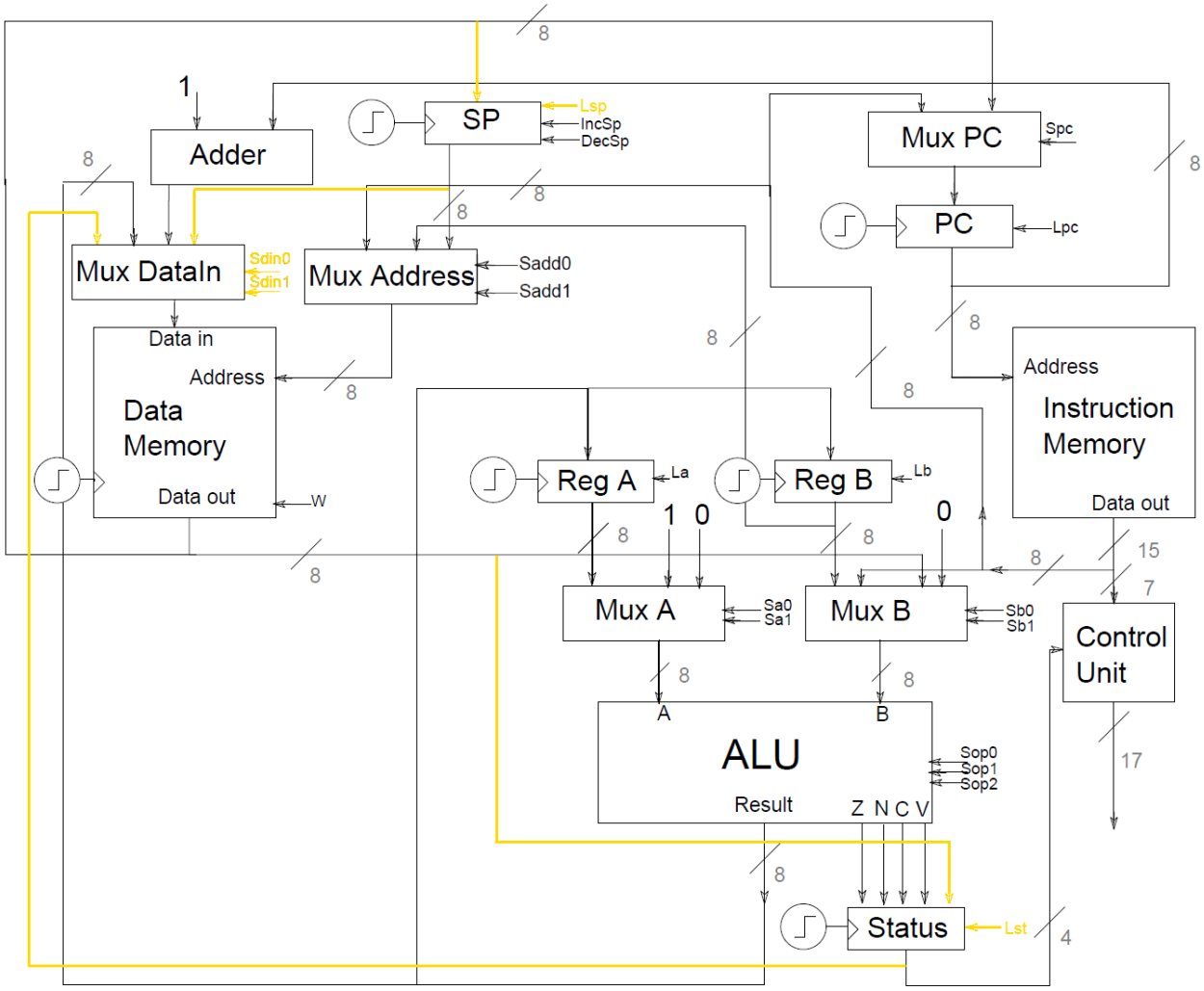


Desde el punto de vista de la ISA, es necesario modificar las señales de control de esas instrucciones, para que activen ahora la señal Ssp cuando se utilicen alguna de estas dos instrucciones. Esto significa que ahora las instrucciones POP y RET setearán la señal Ssp en 1, mientras que el resto de las instrucciones lo seteará en 0.

Inst.	Op.	Opcode	Lpc	La	Lb	Sa0,1	Sb0,1	Sop0,1,2	Sadd0,1	Sdin0	Spc0	W	IncSp	DecSp	Ssp
RET		1000110	1	0	0	-	-	-	SP	-	DOUT	0	1	0	1
POP	A	1001010	0	1	0	0	DOUT	ADD	SP	-	-	0	1	0	1
POP	B	1001100	0	0	1	0	DOUT	ADD	SP	-	-	0	1	0	1

c) Soportar las instrucciones PUSHALL y POPALL. PUSHALL almacena en el stack el contenido de todos los registros (A,B,SP,PC,Status), mientras que POPALL rescata desde el stack estos valores y los vuelve a poner en los registros correspondientes. **(2 ptos.)**

Solución: Estas instrucciones pueden soportarse facilmente utilizando 5 opcodes para cada una, donde cada opcode se relaciona con el respaldo/carga de uno de los registros. Para soportar esto desde el punto de vista del hardware, es necesario crear conexiones desde los registros SP y Status, para que sus valores sean respaldados/cargados.



Para la ISA, debemos utilizar los opcodes creados anteriormente para respaldo/carga de los registros A, B y PC, y además se necesitan 4 opcodes nuevos para respaldo/carga de los registros SP y Status.

Inst.	Op.	Opcode	Lpc	La	Lb	Sa0,1	Sb0,1	Sop0,1,2	Sadd0,1	Sdin0,1	Spc0	W	IncSp	DecSp	Lsp	Lst
PUSHALL		1001110	0	0	0	-	-	-	SP	PC	-	1	0	1	0	0
		1001000	0	0	0	A	0	ADD	SP	ALU	-	1	0	1	0	0
		1001001	0	0	0	0	B	ADD	SP	ALU	-	1	0	1	0	0
		1001111	0	0	0	-	-	-	SP	ST	-	1	0	1	0	0
		1010000	0	0	0	-	-	-	SP	SP	-	1	0	1	0	0
POPALL		1001100	0	0	0	-	-	-	-	-	-	0	1	0	0	0
		1010001	0	0	0	0	-	-	SP	-	-	0	0	0	1	0
		1001100	0	0	0	-	-	-	-	-	-	0	1	0	0	0
		1010010	0	0	0	0	-	-	SP	-	-	0	0	0	0	1
		1001100	0	0	0	-	-	-	-	-	-	0	1	0	0	0
		1001101	0	0	1	0	DOUT	ADD	SP	-	-	0	0	0	0	0
		1001010	0	0	0	-	-	-	-	-	-	0	1	0	0	0
		1001011	0	1	0	0	DOUT	ADD	SP	-	-	0	0	0	0	0
		1000110	0	0	0	-	-	-	-	-	-	0	1	0	0	0
		1000111	1	0	0	-	-	-	SP	-	DOUT	0	0	0	0	0



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

IIC2343 – Arquitectura de Computadores (I/2014)

Examen

- Duración 2:15 horas (08:45 - 11:00)
- Conteste una pregunta por hoja. Escriba su nombre en cada hoja de respuestas, hágalo ahora!
- **No hay preguntas**. Si tiene alguna duda escriba los supuestos necesarios junto con su respuesta.

Pregunta 1 (1.5 puntos)

A continuación se muestran dos algoritmos escritos en pseudo-código, cada uno implementado de dos maneras diferentes. Para medir el desempeño de éstos se tiene una función `tic()` que marca el inicio del "cronómetro" y una función `toc()` que marca el fin. Para cada uno de los ejemplos indique qué código corre más rápido (o bien si no hay diferencias), considerando solamente el código que está entre `tic()` y `toc()`. Justifique.

Caso 1 (0.5 puntos)

```
int matrix() {
  byte mat[N][M] =
    random_matrix(N,M);
  int sum = 0;
  tic();
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
      sum += mat[i][j];
    }
  }
  toc();
  return sum;
}
```

```
int matrix() {
  byte mat[N][M] =
     random_matrix(N,M);
  int sum = 0;
  tic();
  for (int j = 0; j < M; j++) {
    for (int i = 0; i < N; i++) {
      sum += mat[i][j];
    }
  }
  toc();
  return sum;
}
```

Caso 2 (1 punto)

```
int sum() {
  byte arr[N] = random_array(N);
  int sum = 0;
  tic();
  for (int i = 0; i < N; i++) {
    if (arr[i] < 128) {
      sum++;
    }
  }
  toc();
  return sum;
}
```

```
int sum() {
  byte arr[N] = random_array(N);
  arr = sort(arr);
  int sum = 0;
  tic();
  for (int i = 0; i < N; i++) {
    if (arr[i] < 128) {
      sum++;
    }
  }
  toc();
  return sum;
}
```

Respuesta

Caso 1 En este caso corre más rápido el caso de la izquierda, ya que el código va accediendo a lugares contiguos de memoria, optimizando el uso de la caché. La implementación de la derecha debe ir "saltando" por diferentes espacios de memoria, logrando un hit rate mucho más bajo.

Caso 2 Corre más rápido el de la derecha. Como el arreglo está ordenado, la CPU predice de menor manera el branch a seguir debido al `if`. En la realidad, la CPU va ajustando dinámicamente el criterio de branch prediction, lo cual hace que el código de la derecha corra de manera más eficiente.

Pregunta 2 (1.5 puntos)

Se tiene una caché de 16 entradas 2-way associative con 4 palabras por bloque, utilizada para un espacio de memoria de 32 palabras. Indique lo siguiente:

1. Cómo se dividen los bits de la dirección de memoria de una palabra para ser utilizados en la información de la tabla de caché? **(0.5 puntos)**

   Respuesta Los 5 bits de la dirección ($2^5 = 32$), distribuidos como $b_4b_3b_2b_1b_0$ se pueden dividir de la siguiente manera:

   - b_4: Conjunto
   - b_3, b_2: Tag
   - b_1, b_0: Ubicación de la palabra en el bloque

   El orden de los bits utilizados no importa (y por ende los accesos en la parte 2 podrían cambiar), pero los dos bits de tag al igual que los dos bits de ubicación de la palabra en el bloque deben ser contiguos.

2. Utilizando la tabla diseñada en el punto anterior, indique el estado de la memoria caché después de acceder a las siguientes direcciones: **0, 1, 3, 4, 9**

   Use LFU para reemplazar y LRU para desempatar. **(1 punto)**

Respuesta Los primeros tres accesos son al mismo bloque de memoria. Pondremos este bloque en el primer bloque disponible del conjunto 0:

Conjunto	Índice de bloque	Ubic. Palabra	Bit validez	Tag	# Accesos	Tiempo	Dato
0	0	00	1	00	3	1	mem[0]
		01	1	00	3	1	mem[1]
		10	1	00	3	1	mem[2]
		11	1	00	3	1	mem[3]
	1	00	0				
		01	0				
		10	0				
		11	0				
1	0	00	0				
		01	0				
		10	0				
		11	0				
	1	00	0				
		01	0				
		10	0				
		11	0				

Para el acceso a la memoria 4 debemos utilizar el segundo bloque disponible:

Conjunto	Índice de bloque	Ubic. Palabra	Bit validez	Tag	# Accesos	Tiempo	Dato
0	0	00	1	00	3	2	mem[0]
		01	1	00	3	2	mem[1]
		10	1	00	3	2	mem[2]
		11	1	00	3	2	mem[3]
	1	00	1	01	1	1	mem[4]
		01	1	01	1	1	mem[5]
		10	1	01	1	1	mem[6]
		11	1	01	1	1	mem[7]
1	0	00	0				
		01	0				
		10	0				
		11	0				
	1	00	0				
		01	0				
		10	0				
		11	0				

Para el último acceso reemplazamos el bloque 1 del conjunto 0, debido a que ha sido accedido menos veces:

Conjunto	Índice de bloque	Ubic. Palabra	Bit validez	Tag	# Accesos	Tiempo	Dato
0	0	00	1	00	3	3	mem[0]
		01	1	00	3	3	mem[1]
		10	1	00	3	3	mem[2]
		11	1	00	3	3	mem[3]
	1	00	1	10	1	1	mem[8]
		01	1	10	1	1	mem[9]
		10	1	10	1	1	mem[10]
		11	1	10	1	1	mem[11]
1	0	00	0				
		01	0				
		10	0				
		11	0				
	1	00	0				
		01	0				
		10	0				
		11	0				

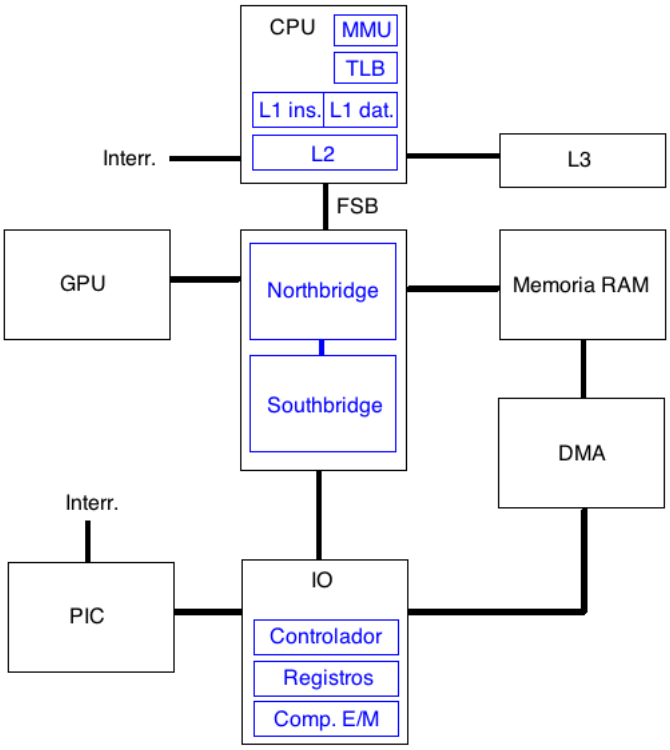
No fue necesario usar LRU para desempatar.

Pregunta 3 (1.5 puntos)

Haga un diagrama de la arquitectura de buses de un computador Intel. Debe incluir los siguientes componentes:

- CPU
- Componentes de la arquitectura de buses intel
- Dispositivos IO (Detalle su estructura interna)
- Interrupciones
- DMA
- Esquema de memoria tipo Von Neumann
- 3 niveles de Caché
- GPU
- Memoria Virtual (MMU + TLB)

Respuesta Los componentes se muestran en el siguiente diagrama:



Pregunta 4 (1.5 puntos)

Responda las siguientes preguntas:

1. De qué sirve tener timers e interrupciones de timers en la CPU? **(0.5 puntos)**

   Respuesta Es crucial ya que sin estos timers el sistema operativo no puede quitar el control a un programa de manera forzada. Dado que el sistema operativo puede configurar estos timers para interrumpir a la CPU en un tiempo posterior, es que éste tiene la seguridad que adelante la CPU, mediante la interrupción del timer, interrumpirá al proceso volviendo el control al sistema operativo.

2. Por qué pueden haber problemas de coherencia de caché en una CPU de 1 solo core? **(0.5 puntos)**

   Respuesta Pueden haber problemas si un dispositivo de IO escribe en memoria usando DMA. Al escribir con DMA la información no pasa por la CPU, por lo cual se podría escribir en memoria un dato que existe en caché sin que la CPU sepa sobre el cambio, produciéndose problemas de coherencia de caché.

3. Explique qué es out-of-order excecution. **(0.5 puntos)**

Respuesta Es una técnica en computación que permite ejecutar instrucciones en assembly de manera desordenada sin afectar el propósito original del programa a ejecutar. Por ejemplo, teniendo las instrucciones:

```
add A, 1
sub B, 2
add A, B
```

puedo intercambiar las instrucciones 1 y 2 sin afectar el estado final del programa. Esto sirve para hacer más eficiente el pipeline, considerando que pueden haber accesos a memoria principal, dependencia de datos u otras demoras que pueden ser compensadas haciendo out-of-order execution.

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



IIC2343 – Arquitectura de Computadores (II/2014)

Solución Interrogación 2

Pregunta 1

a) 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 **(1 pto.)**
Solución: Una posible convención podría utilizar los registros de la arquitectura x86 para almacenar los parámetros de entrada de las subrutinas. Si la cantidad de parámetros excede la cantidad de registros, se utiliza el stack para los restantes parámetros. Comparada con *stdcall*, esta convención es más rapida para leer los argumentos, pero también es más compleja, ya que no todos se ubican en el mismo lugar.

b) ¿Cuál es la función del clock en un computador? ¿Que pasaría si este es removido? **(1 pto.)**
Solución: El clock permite sincronizar todos los elementos de memoria de un computador. Sin él, no existe garantía de que los resultados almacenados en los elementos de memoria, luego de cualquier operación, sean correctos.

c) Dada la microarquitectura del computador básico, ¿es posible crear una **ISA** distinta la actual? Argumente su respuesta. **(1 pto.)**
Solución: Es posible crear nuevas ISAs usando un subconjunto de las instrucciones soportadas por la ISA actual. Además, es posible utilizar también un superconjunto, utilizando instrucciones que no tienen un *opcode* asignado, pero que tienen una combinación de señales de control.

d) ¿Por qué muchos de los dispositivos móviles, como smartphones y tablets, utilizan arquitecturas basadas en **RISC**? Argumente su respuesta. **(1 pto.)**
Solución: La arquitectura RISC, al estar basada en pocas instrucciones muy básicas, utiliza hardware simple que requiere poca energía. Esto la hace una arquitectura idea para dispositivos donde el consumo de energía es un parámetro crítico.

e) Escriba un programa en assembly del computador básico, que calcule el inverso aditivo de un número entero de 8 bits. **(1 pto.)**
Solución:

```
DATA
n         ?
n_inv

CODE
MOV A, (n)
NOT B,A
INC B
MOV (n_inv), B
```

f) ¿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. **(1 pto.)**
Solución: Si es posible emular el registro BP, utilizando una dirección fija de memoria. De esta manera, en vez de hacer referencia **BP**, se hará referecia a esta dirección. El único cuidado necesario es asegurarse que el valor almacenado en esa dirección no sea sobreescrito.

Pregunta 2

En lo siguientes ejercicios, deberá modificar la microarquitectura y/o la ISA del computador básico para que cumpla ciertos requerimientos. Todos estos cambios deben ser claramente especificados.

a) Modifique el computador básico, para que este utilice un esquema Von Neumann, *i.e.*, memoria de datos e instrucciones unificadas en una sola. **(3 ptos.)**
Solución: Desde el punto de vista de la microarquitectura, la idea principal para este ejercicio es dividir cada ciclo del clock en dos etapas. En la primera etapa, se incrementa el PC, que ahora está conectado a la memoria RAM compartida. Esta palabra (instrucción) es enviada a la unidad de control. Es importante que durante toda esta etapa, el resto del computador no sea alimentado con un flanco de subida. Una vez que la unidad de control el alimentada con la instrucción, comienza la segunda etapa, generando un flanco de subida para todos los elementos restantes, mientras que el PC no debe ser alimentado con un flanco de subida. De esta manera, la instrucción se ejecutará correctamente, terminando en el inicio de un nuevo ciclo, correspondiente al incremento del PC.
Desde el punto de vista de la ISA, se requiere que las instrucciones no permitan que los datos leidos de memoria lleguen a la unidad de control. El traspaso entre memoria y unidad de control solo debe ocurrir en la primera etapa del ciclo.

b) Modifique el computador básico para que este pueda detectar la posibilidad de que el stack sobreescriba datos almancenados en memoria. Agregue además la(s) instrucción(es) necesaria(s) para que un programa pueda manejar esta situación. **(3 ptos.)**
Solución: Para la microarquitectura, la idea es agregar **N** registros de 1 bit, donde **N** es la cantidad de palabras que puede almacenar la memoria. Estos registros indicarán si la palabra en esa posición ha sido escrita por una instrucción del tipo **MOV**, almacenando un 1. De esta manera, se asume que los registro parten con un 0 almacenado. Luego, se agrega un nuevo bit al registro status, **O**, cuyo valor se genera en base a la palabra apuntada actualmente por el registro **SP**, *i.e.* la salida del registro SP se conecta al bus selector de un MUX de **N** entradas, donde cada una de estas está conectada a uno de los registros de 1 bit. La salida de este MUX es el bit **O**.
Para que un programa pueda manejar esto, se agrega la instrucción JSO (**J**ump **S**tack **O**verwrite), que actua de manera similar al resto de las instrucciones de salto. En este caso, verifica si el bit **O** del registro status está en 1 para efectuar el salto.

Pregunta 3

a) 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: **(3 ptos.)**

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

Solución:

```
        JMP    main
        a db    ?
        b db    ?
        res db 0
    main:
        MOV    AX, 0
        MOV    AL, b
        PUSH   AX
        MOV    AL, a
        PUSH   AX
        CALL   euclides
        MOV    res, AL
    euclides:
        PUSH   BP
        MOV    BP, SP
        MOV    BX, [BP-6]
        CMP    BL, 0
        JMP    setA
        MOV    AX, [BP-4]
        DIV    BL
        MOV    CX, 0
        MOV    CL, AH
        PUSH   CX
        PUSH   BX
        CALL   euclides
        JMP    return
    setA:
        MOV    AX, [BP-4]
    return:
        POP BP
        RET 4
```

b) Un computador utiliza una arquitectura Harvard + RISC, con las siguientes características principales:

- Sólo 3 registros, **A**, **X** e **Y**, todos de 8 bits, donde sólo el registro **A** tiene acceso a operaciones aritméticas o lógicas. Las únicas operaciónes aritméticas válidas para **X** e **Y** son INC X o INC Y y DEC X o DEC Y.
- Las operaciones aritméticas y lógicas de esta arquitectura, reciben sólo un parámetro, ya que afectan unicamente al registro **A**. Por ejemplo, ADD 0x10 sería igual a ADD A, 0x10. Las operaciones disponibles son las mismas que en x86, con excepción de MUL y DIV, que no existen en esta arquitectura.
- Múltiples instrucciones de carga, definidas de la siguiente manera:

Instrucción	**Descripción**	**Uso**
LD? #Lit	Carga en el reg. **?** el valor del literal **Lit**	LDA #0x01
LD? Dir	Carga en el reg. **?** el contenido de la dir. **Dir**	LDX 0x34
LDA Dir,?	Carga en el reg. **A** el contenido de la dir. dada por **Dir+?**	LDA 0x34,X
ST? Dir	Carga en la dir. **Dir** el valor almacenado en el reg. **?**	STA 0x7C
STA Dir,?	Carga en la dir. **Dir** el valor almacenado en la dir. dada por **A+?**	STA 0x7C,Y
TRANS Reg1,Reg2	Carga en el reg. **Reg1** el valor almacenado en el reg **Reg2**	TRANS A,X

Utilizando la arquitectura recién descrita, escriba un programa que calcule el producto punto entre dos vectores de números enteros no negativos. **(3 ptos.)**

3b)

Asumo que ambos arreglos/vectores son del mismo largo y que la operación no causa Overflow.

```
Length: db ? ;largo de los arreglos

Res: db ? ;resultado

Arr1: db ? ;puntero al inicio del arreglo 1

Arr2: db ? ;puntero al inicio del arreglo 2

Main:

      LDA #0x00 ;cargamos un 0 en A

      STA Res ;ponemos un 0 en el resultado

      LDX Length;cargamos un 0 en X

.loop

      TRANS A,X ;transferimos el valor de X a A

      CMP #0x00 ;comparamos si es igual al largo de los arreglos

      JGE .exit ;si es igual o mayor al largo salimos

      LDA Res

      PUSH A ;hacemos push del resultado

      CALL Mul ;multiplicamos [Arr1 + x] y [Arr2 + x] dejamos el
resultado de la multiplicación en Res

      POP A ;Recuperamos el valor del resultado en el registro A

      ADD Res ;sumamos el resultado de la multiplicacion con el resultado
actual

      STA Res ;cargamos el resultado a la variable Res

      DEX ;decrementamos X en 1 (DEC X)

      JMP .loop ;volvemos al loop

.exit

      ;resultado queda en Res
```

```
      RET
MUL:
      LDA Arr1,x ;pone el valor de Arr1 + x en A
      TRANS Y , A ;carga Arr1 + x en el registro Y
      LDA #0x00 ;carga un 0 en Res
      STA Res
.loop
      TRANS A , Y ;transfieree a Y al registro A
      CMP #0x00
      JEQ .exit ;si Y es 0 entonces retorna
      LDA Res ;carga el valor de Res en A
      ADD Arr2,x ;le suma Arr2,x a Res
      STA Res ;carga el valor en res
      DEY ;decrementa Y (DEC Y)
      JMP .loop ;vuelve al loop
.exit
      RET
```

1.5 por uso adecuado del ASM.

1.5 por el algoritmo

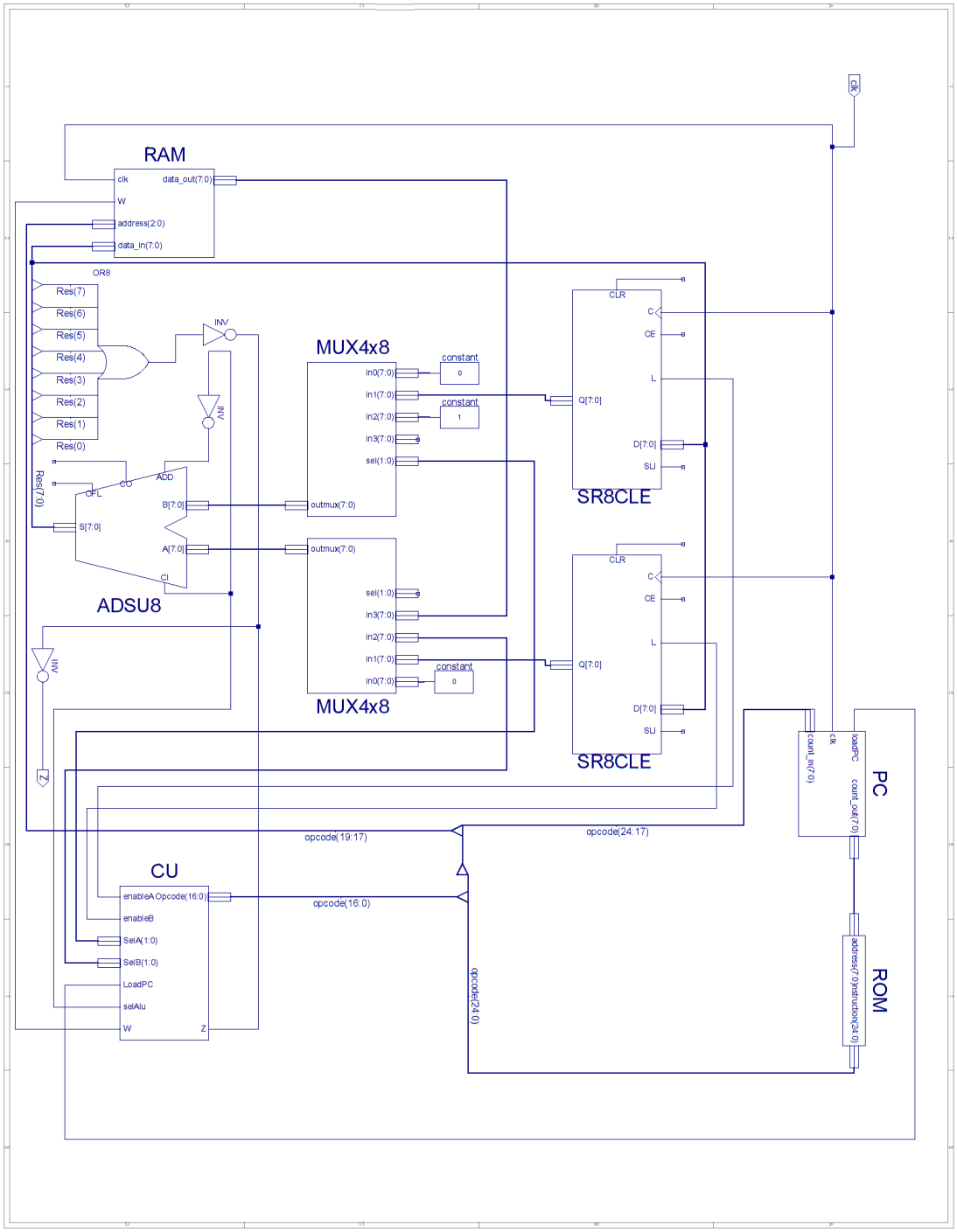
Si el uso del ASM tiene puntaje 0 la pregunta completa tiene 0.

Si la pregunta está incompleta recibe penalización en ambos puntajes.

Pregunta 4

a) La arquitectura es Harvard porque separa la memoria de instrucciones y la de datos.

b) Posible solución:



c) Posible solución, de acuerdo a la respuesta de b)

ADD A, B
“00000000 000000000 010 01 01 0”

SUB B, 10
“00001010 000000000 001 01 01 1”

INC B
“00000000 000000000 001 10 01 0”

MOV (3), A
“00000011 000000000 100 01 00 0”

MOV B, (7)
“00000111 000000000 001 00 01 0”

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:
```

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



IIC2343 – Arquitectura de Computadores (II/2015)

Interrogación 2

Pregunta 1

a) Compare las arquitecturas Harvard y Von Neumann desde el punto de vista del tiempo de ejecución de las instrucciones. Fundamente y explique claramente las diferencias. **(1 pto.)**
Solución: Al necesitar cómo mínimo dos accesos a memoria no simultáneos para ejecutar una instrucción arbitraria, la arquitectura Von Neumann es en teoria más lenta que la Harvard, que necesita sólo un ciclo para hacer lo mismo.

b) ¿Cuántas pasadas por el código fuente debe hacer como mínimo un assembler para x86? ¿Cuántas debe hacer el assembler del computador básico? **(1 pto.)**
Solución: Al no conocer a priori el lugar donde están las declaraciones de variables, el assembler para x86 necesitará como mínimo 2 pasadas: la primera para almacenar los nombres y ubicación de las variables y la segunda para generar el código. Por otro lado, el assembly del computador básico sólo necesita una, ya que las variables sólo se declaran en el segmento DATA.

c) ¿Cómo favorece a los dispositivos móviles la ley de Moore? **(1 pto.)**
Solución: Al mejorar la miniaturización, es posible construir procesadores más potentes, que utilizan menos espacio y consumen menos energía. Esto es exactamente lo que necesita un procesador movil.

d) Describa el procedimiento que realiza un disco duro para acceder a un dato. **(1 pto.)**
Solución: Para acceder a un dato, el disco duro gira el disco magnético y posiciona el cabezal en el radio adecuado. Luego, al pasar sobre la ubicación del dato, el cabezal lee la información.

e) ¿Qué beneficios podría traer el uso de procesador RISC para servidores? **(1 pto.)**
Solución: Al ser procesadores más simples, su uso de energía es menor. Esto resulta muy beneficioso para servidores que no realicen tareas muy complejas, ya que se podría tener muchos de estos, por una fracción del consumo energético de un grupo de servidores tradicionales.

f) Explique cómo funciona la transferencia de direcciones y datos desde/hacia los dispositivos mapeados a memoria. **(1 pto.)**
Solución: Las direcciones son redireccionadas mediante un *Address Decoder*, que decide si corresponde a un dispositivo de I/O o a la memoria. Luego, esta dirección llega al dispositivo correcto, que recibe/envía el dato por el bus de datos.

Pregunta 2

a) Modifique el computador básico para que tenga soporte para las instrucciones IN y OUT. Describa los cambios a la microarquitectura y a la ISA. **(2 ptos.)**

Solución: Los cambios a realizar son los siguiente, asumiendo que los dispositivos pueden conectarse entre los puertos 1-255:

- Agregar una señal de control a la salida de la unidad de control. Esta señal será 1 sólo cuando la instrucción sea IN o OUT, el resto del tiempo será 0.
- Para OUT: Crear un nuevo bus de 8 bits, que sale del registro A, con dirección a los dispositivos de I/O. Además, sacar una copia del bus de literales, en dirección a los dispositivos de I/O. El bus que viene de A será el bus de datos de entrada a los dispositivos, mientras que el que sale del bus de literales será el de direcciones. Finalmente, este último bus debe estar controlado por un enabler, que se activa mediante la nueva señal de control agregada.
- Para IN: Crear un nuevo bus de 8 bits, que viene desde los dispositivos hasta el Mux A, aprovechando la entrada que no está ocupada. Además, para indicar el puerto se utiliza el bus de direcciones definido en el item anterior.

b) Conecte al computador modificado en el ítem anterior, un dispositivo de I/O que utilice los puertos 13 (comandos), 14 (estado) y 15 (datos). Diseñe las conexiones entre los buses de datos y direcciones y los registros de comandos, estado y datos del dispositivo. **(2 ptos.)**
Solución: Se asume que los registros de estado y datos son de sólo lectura, mientras que el de comandos es de sólo escritura.

- Para el registro de comandos, conectar la señal Load de este a la salida de un circuito lógico que compara el valor en el bus de direcciones con el número 13, entregando 1 si es igual y 0 en otro caso. El bus de datos de salida a los dispositivos de I/O se conecta a la entrada del registro de comandos.
- Para el registro de estado y datos, conectar la salida de estos a un mux de 4 entradas, donde sólo 3 se usan: registro de estado, registro de datos, literal 0. El control del mux se conecta con un circuito lógico que retorna 0 si el bus de direcciones contiene un 14 (estado), 1 si contiene un 15 (datos) y 2 en cualquier caso (literal 0). La salida del mux se conecta con el bus de salida de los dispositivos de I/O (definido anteriormente), que va al Mux A.

c) Asuma que el dispositivo de I/O conectado en el ítem anterior es una tarjeta de red. 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. **(2 ptos.)**
Solución: Se define el comanda que inicia el proceso de recepción como 0, el estado de dato recibido como 0.

```
DATA
CODE
    MOV   B, 0x40
    MOV   A, 0
    OUT   13, A
  for:
    IN    14, A
    CMP   A, 0
    JNE   for
    IN    15, A
    MOV   (B), A
    INC B
    JMP for
```

Pregunta 3

Implemente en assembly x86 el algoritmo para ordenar arreglos de enteros *min-max sort*. Este algoritmo opera buscando el máximo y el mínimo de un arreglo y colocándolos en los extremos correspondientes, para luego repetir el proceso con el subarreglo que no contiene a los extremos. En caso de usar subrutinas, utilice la convención *stdcall*. Cualquier cosa que asuma debe quedar claramente explicada. **(6 ptos.)**

Solución: Se asume que el arreglo a ordenar es de números enteros de 8 bits.

```
  JMP   main
  N     dw   ?
  A     db   ?,?,?,...
main:
  PUSH  N
  LEA   BX, A
  PUSH  BX
  CALL  min_max_sort
  RET
min_max_sort
  PUSH  BP
  MOV   BP, SP
  MOV   BX, [BP+4]
  MOV   CX, [BP+6]
  MOV   DI, 0 ; indice del maximo
  MOV   AX, 0 ; indice del minimo
  MOV   DH, -128 ; valor maximo
  MOV   DL, 127 ; valor minimo
  MOV   SI, 0 ; contador
for:
  CMP   SI, CX
  JEQ   end_for
  CMP   [BX+SI], DL
  JLT   new_min
  CMP   [BX+SI], DH
  JGT   new_max
  JMP   next
new_min:
  MOV   DL, [BX+SI]
  MOV   AX, SI
  JMP   next
new_max:
  MOV   DH, [BX+SI]
  MOV   DI, SI
next
  ADD   SI, 1
  JMP   for
end_for:
  MOV   SI, AX
  MOV   AL, [BX]
  MOV   [BX], DL
  MOV   [BX+SI], AL
  MOV   SI, CX
  SUB   SI, 1
  MOV   AL, [BX+SI]
  MOV   [BX+SI], DH
  MOV   [BX+DI], AL
```

```
  CMP   CX, 2
  JEQ   return
  CMP   CX, 3
  JEQ   return
  SUB   CX, 2
  PUSH  CX
  ADD   BX, 1
  PUSH  BX
  CALL min_max_sort
return:
  POP BP
  RET   4
```

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



IIC2343 – Arquitectura de Computadores (I/2016)

Interrogación 2

Pregunta 1

a) Si en una microarquitectura Von Neumann la razón velocidad/precio de la CPU es órdenes de magnitud mayor que la de la memoria RAM, ¿que tipo de ISA es más conveniente? **(1 pto.)**
Solución: Conviene una ISA CISC, con el fin de minimizar los accesos a memoria, que serán mucho más lentos que el procesamiento de la instrucción.

b) ¿Por qué es necesaria la existencia de la instrucción RET Lit en un computador x86? **(1 pto.)**
Solución: Es necesario para dar soporte a las convenciones de llamada que requieren que la subrutina llamada deje el stack en su estado original. RET Lit elimina del stack Lit palabras luego de retorna de la subrutina.

c) Describa paso a paso el proceso que se lleva a cabo para ejecutar un programa almacenado en un disco duro. **(1 pto.)**
Solución:

   a) Transferir el programa, total o parcialmente, desde el disco a la memoria principal.

   b) Cargar el registro PC (IP) con la dirección de memoria en la cual se ubica la primera palabra del programa.

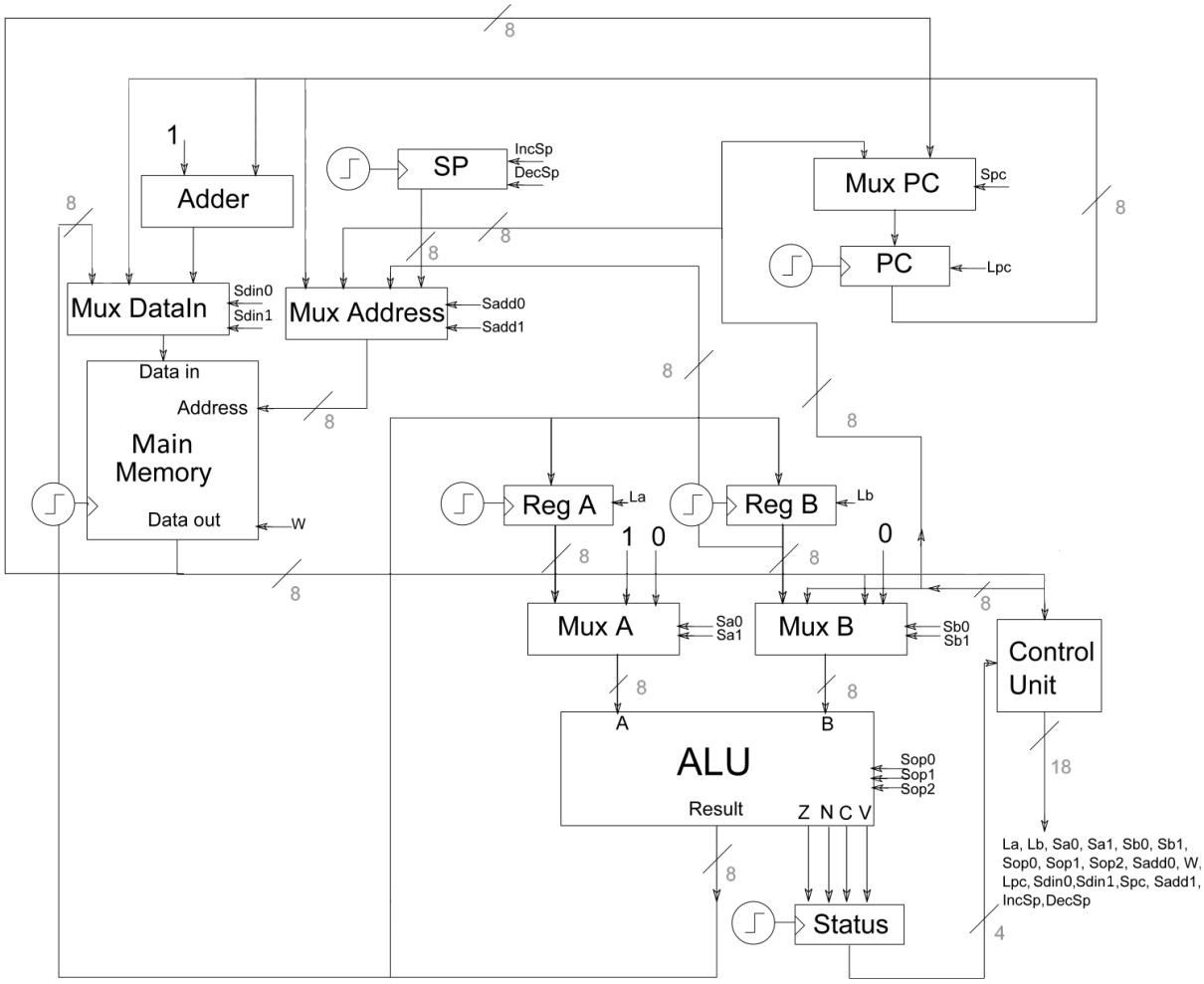
d) Al procesar una IRQ, ¿por qué el respaldo de los condition codes debe ser anterior a la verificación del flag IF? **(1 pto.)**
Solución: Al verificar el valor de IF, los flags son modificados, por lo que se pierde sus valores anteriores. Por este motivo, el respaldo de los flags debe realizarse antes.

e) En un programa escrito en assembly x86, una variable de tipo **db** es declarada en el cuerpo de una subrutina. ¿Es posible referenciar esta variable fuera de la subrutina? **(1 pto.)**
Solución: Sí, es posible, debido a que al ser declarada, su dirección de memoria se encuentra disponible para ser referenciada desde cualquier sector del programa.

f) ¿Cuál es la función del vector de interrupciones? ¿Cuál es su contenido? **(1 pto.)**
Solución: El vector de interrupciones almacena las dirección de inicio de las distintas ISR registradas en el sistemas. Al momento de procesar una IRQ, se debe buscar en el vector de interrupciones la dirección de la ISR asociada con el dispositivo que generó la IRQ.

Pregunta 2

Considere la siguiente figura que presenta la versión Von Neumann del computador básico:



En base a este computador, conteste las siguientes preguntas:

a) ¿Cuál es la menor cantidad de ciclos que puede tomar la ejecución la ejecución de una instrucción? Justifique su respuesta. **(1 pto.)**
Solución: Se necesitan tres ciclos. Uno para transferir el OpCode, otro para el literal y finalmente uno para ejecutar la instrucción.

b) Indique que modificaciones deben realizarse al assembly y la ISA del computador básico para que sea compatible con este nuevo computador. **(1 pto.)**
Solución: A pesar de que no es necesario, el assembly puede simplificarse al eliminar la definición de los segmentos CODE y DATA, que ya no son necesarios. Desde el punto de vista de la ISA, los únicos cambios obligatorio son aumentar a 8 bits los opcodes, indicar la cantidad de ciclos que toman ahora las instrucciones.

c) ¿Siguen siendo la unidad de control y el PC iguales a los del computador básico? Justifique detalladamente su respuesta. **(2 ptos.)**

Solución: No, ambos sufren cambios importantes. El PC debe suspender su incremento cada 2 pasos, para permitir la ejecución de la instrucción. Por otro lado, la unidad de controle debe bloquear su entrada cuando corresponda leer datos de memoria y no instrucciones.

d) ¿Cuál es el valor del registro A al terminar la ejecución del siguiente programa en el computador básico Von Neumann? Justifique detalladamente su respuesta. **(2 ptos.)**

```
            JMP main
func:       MOV A, 17
            POP B
            MOV (B), A
            PUSH B
            MOV A, 7
            MOV B, 3
            RET
main:       MOV A, 7
            MOV B, 3
            CALL func
            ADD A,B
```

Solución: Sin conocer el valor de los nuevos opcodes, no es imposible inferir el valor de A, ya que al sobrescribir el valor almacenado en la dirección de retorno de la subrutina (líneas 3, 4 y 5), con el número 17, el opcode que estaba almacenado en esa dirección se perdió. Si utilizamos los valores de los opcodes del computador básico, el número 17 corresponde a la instrucción SUB A,B, por lo tanto el valor final de A sería 4.

Pregunta 3

Un computador x86 de 16 bits ejecuta una proceso de cálculo científico de alta prioridad, que utiliza intensivamente la CPU. El proceso lee y escribe datos constantemente desde un disco interno de almacenamiento no volátil. Este dispositivo se encuentra conectado al South Bridge y es capaz de transferir datos a una velocidad máxima de 200 MB/s. Para las transferencias de salida, los controladores de disco exponen un buffer temporal de 40 bytes, cuyo contenido completo es copiado al dispositivo, en base a la dirección de memoria indicada en el registro de direcciones. Las transferencias de entrada funcionan de manera análoga.

La siguiente tabla indica algunos de los puertos utilizados por el sistema para interacción con dispositivos de I/O:

Puerto	Función
0-99	Entradas address decoder. 5 bytes cada una. Formato: |Dir. mem. inicio (2 bytes)|# palabras intervalo (1 byte)|Puerto inicio (2 bytes)|
133	Reg. comandos controlador DMA
134-135	Reg. origen controlador DMA
136-137	Reg. destino controlador DMA
138-139	Reg. cont. palabras controlador DMA
155	Reg. comandos controlador disco 1
156-157	Reg. estado controlador disco 1
158-161	Reg. direcciones controlador disco 1
162-201	Buffer temporal controlador disco 1

a) ¿A qué interfaz debe conectarse el dispositivo de almacenamiento para maximizar su velocidad de transferencia de datos? **(1 pto.)**
Solución: Debe conectarse a la interfaz SATA, que permite transferencias de hasta 300 MB/s.

b) ¿Que problema tiene potencialmente el esquema de comunicación de los controladores de disco? **(1 pto.)**
Solución: Al ser accedido sólo mediante puerto, el controlador no permite el uso de DMA para liberar a la CPU de las transferencias de datos desde el disco.

c) En base a la tabla, indique en detalle como mejoraría el esquema de comunicación de los controladores, con el fin de maximizar el uso de la CPU por parte del proceso de cálculo científico. **(2 ptos.)**
Solución: Para maximizar el uso de la CPU es obligatorio utilizar el controlador de DMA para realizar las transferencias. Para hacerlo, es necesario cargar en el Address Decoder la transformación desde un rango de direcciones de memoria a los puertos del controlador de disco que correspondan al su buffer temporal. Por ejemplo, al cargar la siguiente entrada en los puerto 0 al 4 (Address Decoder), el buffer del controlador de disco puede ser accedido usando las direcciones de memoria en el rango 500-539: 500 |40 |162.

d) Describa detalladamente el procedimiento que se debe llevar a cabo, cada vez que el proceso de cálculo requiere leer o escribir en el dispositivo de almacenamiento, considerando todas las modificaciones realizadas en el ítem anterior. **(2 ptos.)**
Solución:

   a) Cargar en el controlador de DMA la direcciones de inicio y destino de los datos y la cantidad de palabras a copiar. Es necesario considerar que los datos se leen y escriben en el buffer del controlador de disco, por lo que la dirección de origen o destino, dependiendo del caso, debe ser la asignada en el ítem anterior al puerto 162. Basándose en la solucion entregada acá, la dirección de inicio o destino es la 500. Otro aspecto importante es que el tope a transferir hacia y desde el disco es 40 bytes por vez, dado que el buffer el controlador de disco tiene ese tamaño. Si la transferencia es mayor a eso, es necesario dividirla en múltiples transferencias pequeñas.

b) Cargar en el registro de de direcciones del disco, puertos 158 a 161, la dirección desde o hacia donde se leerá o escribirá en el disco. Esta dirección corresponde al espacio direccionable del disco y no tiene relación con el espacio direccionable del computador.

c) Finalmente, dependiendo si se lee o escribe, enviar los comandos correspondientes al controlador de DMA y al controlador de disco. Si es lectura, primero se debe enviar el comando de lectura al controlador de disco y luego el de transferencia al controlador de DMA. Si es escritura, primero se envía el comando de transferencia al controlador de DMA y luego el de escritura al controlador de disco.

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



IIC2343 – Arquitectura de Computadores (II/2016)

Solución Interrogación 2

Pregunta 1

a) Luego de recibir la señal `INTA`, ¿qué tarea(s) debe realizar un controlador de interrupciones? **(1 pto.)**
Solución: Una vez recibida la señal, el controlador de interrupciones debe revisar en sus registros internos cuál fue el dispositivo que generó la IRQ. Una vez obtenido este ID, es enviado a la CPU a través del bus de datos.

b) Dos computadores poseen la misma ISA, pero uno tiene microarquitectura Von Neumann, mientras que el segundo es Harvard. ¿Es posible ejecutar el mismo programa en ambos computadores, sin realizar modificaciones a este? **(1 pto.)**
Solución: Sí, si ambos computadores usan la misma ISA, la ejecución del programa será independiente de la microarquitectura.

c) Implemente, utilizando sólo la instrucción `SUBLEQ a,b,c`, la instrucción `SUB a,b`. **(1 pto.)**
Solución: `SUB a,b`

- dir0: `SUBLEQ a,b,dir1`
- dir1: ...

d) ¿Por qué motivo podría dejar de cumplirse la ley de Moore en el futuro cercano? **(1 pto.)**
Solución: Si el proceso de miniaturización sigue desarrollandose al ritmo actual, el tamaño de los transistores llegará dentro de poco al límite físico para el tamaño de una compuerta lógica.

e) ¿Que ventaja(s) tiene el mapear un dispositivo de I/O a memoria? **(1 pto.)**
Solución: Un dispositivo de I/O mapeado a memoria puede ser utilizado en conjunto con el DMA para permitir transferencias desde/hacia memoria que no requieran la participación de la CPU.

f) Al finalizar la siguiente sección de código, ¿es el contenido de los registros AX y BX el mismo? **(1 pto.)**

```
MOV [0],AX
MOV BH, [0]
MOV BL, [1]
```

Solución: No, ya que la arquitectura x86 utiliza endianness little-endian, por lo que al guardar el valor en BX, el orden de los bytes estará invertido, si se compara con AX.

Pregunta 2

a) ¿Es posible soportar subrutinas en el computador básico, si se eliminan de la ISA las instrucciones `CALL` y de salto? Justifique su respuesta. **(1 pto.)**
Solución: Si es posible. Basta con utilizar la instrucción RET para emular la instrucción CALL de la siguiente forma:

I. Se pone un label (label_ret) después de la llamada a la subrutina para tener el valor (PC + 1).
II. Se guarda el valor del label en algún registro (por ejemplo MOV A, label_ret) y se hace push al stack (PUSH A).
III. Se pone un label (label_call) antes del código de la subrutina para así poder entrar a esta.
IV. Se guarda el valor del label en algún registro (por ejemplo MOV A, label_call) y se hace push al stack (PUSH A).
V. Se ejecuta la instrucción RET para saltar al ultimo valor guardado en el stack (la linea donde comienza la subrutina).

b) Modifique (sólo) la ISA del computador básico para soportar la instrucción `CALL reg`, que permite llamar a la subrutina ubicada en la dirección de memoria almacenada en el registro `reg`. **(2 ptos.)**
Solución: Se puede reservar una dirección de memoria para poder guardar el valor del registro en esta dirección (llamémosla dir, esta dirección es fija y se utilizará siempre que se ejecute la instrucción CALL reg). Luego, se debe agregar la instrucción CALL (a), que es equivalente al CALL normal, solo que el valor al cual se saltará no viene dado por un literal desde la memoria de instrucciones, sino que viene de la memoria principal, en la posición (a), por lo que la instrucción se debe realizar en 2 ciclos: Uno para guardar en la memoria principal el valor del PC + 1 antes de entrar a la subrutina, y otro para obtener el valor al cual se quiere saltar desde la dirección dir en la memoria principal (Esto es soportado por el computador básico ya que el MUX PC permite cargar al PC por un literal proveniente de la memoria de instrucciones o por un valor de la memoria principal).

Las señales de instrucción para soportar CALL (a) son:

Lpc	La	Lb	Sa0,1	Sb0,1	Sop0,1,2	Sadd0,1	Sdin0	Spc0	W	IncSp	DecSp
0	0	0	-	-	-	SP	PC	-	1	0	1
1	0	0	-	-	-	LIT	-	MEM	0	0	0

Finalmente, para implementar CALL reg basta con definirla como una instrucción de 3 ciclos como sigue:

I. MOV (dir), reg
II. CALL (dir)

c) 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. **(3 ptos.)**
Solución: La arquitectura x86 es autoprogramable, ya que está basado en la arquitectura Von Neumannm, en donde la memoria principal se utiliza tanto para guardar instrucciones como para almacenar datos. Así, si se conoce toda la especificación de la ISA, es posible escribir programas que escriban programas, en particular subrutinas, de la siguiente forma:

I. Se guarda úna dirección de memoría específica en donde comenzará a escribirse la subrutina.
II. Desde esa dirección, se comienzan a escribir números de manera contigua en memoria que corresponden a los opcodes de la subrutina.

III. Al terminar de escribir la subrutina en memoria, se hace CALL a la dirección en donde comienza la subrutina.

Pregunta 3

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

a) ¿Cuantos dispositivos que generen solicitudes de interrupción pueden conectarse? **(1 pto.)**
Solución: Debido a que se puede conectar un número arbitrario de dispositivos en la controladora USB, es infinito la respuesta.

b) 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. **(1 pto.)**
Solución: Un mecanismo que opera es que al generar una interrupción, se llama a la ISR de la controladora USB, ésta lee algún registro de estado de la controladora para determinar la rutina que debe seguir atendiendo y la invoca, ya sea como Trap, subrutina, etc.

c) ¿Que ocurriría en este computador si se ejecuta la instrucción `MOV [0],AX`? **(1 pto.)**
Solución: Se cambiaría el puntero a la ISR del timer del sistema, no confundir con cambiar el timer. Se puede inferir que esto sería fatal para el funcionamiento del computador.

d) 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 sólo 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. **(3 ptos.)**
Solución: Una posible solución es transportar el vector de interrupciones a un registro especializado, que sólo puede ser escrito en modo supervisor y es accesado por nuevas instrucciones para escritura, por ejemplo: SINT <ID Interupción>, <Label ISR = lit>. La ejecución seguiría usando INT en la ISA, pero su camino de ejecución debe ser alterado.

Arquitectura de Computadores – IIC2343

I2

13 octubre 2017

1. Considera la siguiente función **`factorial`**, que para un número entero *n* dado calcula *n*!, definida recursivamente en un (pseudo)lenguaje de alto nivel:

```
int factorial(int n):
   if n == 0:
      return 1
   else:
      return n*factorial(n-1)
```

a) [**2**] Escribe el código correspondiente en el lenguaje *assembly* x86 (286, 386 o Core i7).

b) [**1**] Escribe en el mismo (pseudo)lenguaje de alto nivel una versión iterativa (en vez de recursiva) de la función **`factorial`**.

c) [**1**] Escribe el código en lenguaje *assembly* x86 correspondiente a tu función iterativa de b).

d) [**2**] Se suele decir que las versión iterativa de un algoritmo es más eficiente que la correspondiente versión recursiva. Explica claramente por qué esto es cierto en el caso de la función **`factorial`**; o bien explica claramente por qué no lo es. Haz referencia a los códigos de tus respuestas a a) y c).

Respuesta.

a) Aparece en la p. 12 de los apuntes "7-Arquitectura x86".

b)

```
int factorial(int n):
   f = 1
   for i = 2 ... n:     —si n es 0 o 1, este loop simplemente no se ejecuta
      f = f*i:
   return f
```

c) [falta]

d) Es cierto, ya que tanto el código para a) como el código para c) incluyen las *n* multiplicaciones, pero el código para c) no incluye las instrucciones para la llamada recursiva (que se ejecuta *n* veces), incluyendo el manejo del stack: PUSH, CALL, POP.

2. Considera el siguiente programa escrito en un (pseudo)lenguaje de alto nivel, consistente en una parte principal, **`main`**, y dos funciones (o subrutinas), **`f`** y **`g`**:

```
10 main:
14     int a = 9
18     int b = 6
22     int c = f(a, b)
26     print(c)
```

```
30 int f(int p, int q):
34     if q == 0:
38         return p
42     int r = p%q
46     return g(q,r)
```

```
50 int g(int x, int y):
54     if x == y:
58         return x
62     if x > y:
66         int z = x-y
70         return f(z, y)
74     int v = y-x
78     return f(x, v)
```

Muestra el contenido del stack de *frames* (o registros de activación) a medida que se ejecuta el programa; en particular, el *frame pointer* (o *base pointer*), el *stack pointer*, la dirección de retorno, los parámetros y las variables locales, en cada uno de los siguientes momentos:

a) Cuando **`f`** es llamada por primera vez desde **`main`** y está a punto de ejecutar **`if q == 0`**.

b) Cuando **`g`** es llamada por primera vez desde **`f`** y está a punto de ejecutar **`if x == y`**.

c) Cuando **`f`** es llamada por primera vez desde **`g`** y está a punto de ejecutar **`if q == 0`**.

Respuesta.

Esta es la versión "exhaustiva". Sin embargo, la respuesta correcta es más simple. Interesa principalmente saber si tienen claro el concepto del uso de un stack para implementar llamadas de funciones. Está basada en el ejemplo de las torres de Hanoi incluido en el archivo ISA.pptx (diaps. 48-50). Por lo tanto, no es necesario pasarse a código x86.

a) Antes de llamar a f, el se usaron 2 PUSHs de 2 bytes cada uno y además se llamó a una subrutina por lo tanto esto equivale a que el SP haya quedado en -6, luego se hace otro PUSH (-8 queda el SP con esto) al BP y luego se le pone el valor de SP a BP y luego al SP se le resta -2 para ponerle una variable local. En resumen, el Stack queda así:

SP->	246	Variable Local r	Basura
	247	Variable Local r	
BP->	248	Valor de BP	Basura
	249	Valor de BP	
	250	Dirección de Retorno	26
	251	Dirección de retorno	
	252	Parámetro q	6
	253	Parámetro q	
	254	Parámetro p	9
	255	Parámetro p	

La dirección de retorno quedaría justo después del llamado a f, esta corresponde a la ubicación de MOV varC,AX , que en nuestro código sería justo cuando llamamos a print(c), si nos guiamos por los números del código original, la dirección de retorno debiera ser 26.
Los parámetros de la función f serian los números a y b, por lo tanto serian 9 y 6 y estarían posicionados en el stack entre la posición 252 y 255.

La única variable local utilizada es r y variable estaría almacenada en el stack en las ubicaciones 246 y 247. No se conoce su valor ya que aún no es inicializada por lo que tiene basura.

En resumen, sería lo siguiente:

Dato	Valor	Ubicación en el stack
BP	248	Almacena su valor anterior en 248 y 249
SP	246	---
Dirección de retorno	26	250 y 251
Parámetro p	9	254 y 255
Parámetro q	6	252 y 253
Variable local r	Basura	246 y 247

b) Cuando g es llamado por primera vez, como vimos en a), el SP está posicionado en 246 y luego se le hacen 2 PUSH de 2 bytes cada uno para introducir los parámetros por lo que baja hasta 242, luego se llama a g por lo que el SP baja hasta 240 para darle espacio a la dirección de retorno. Luego cuando se llama g lo primero que se hace es un PUSH de 2 bytes (por lo que queda en 238) y luego BP pasa a tener el valor de SP y luego se le resta 4 a SP para darle espacio a las variables locales z y v por lo que SP queda en la posición 234.

En resumen, quedaría el Stack de esta manera:

Pertenecen a la función g	SP->	234	Variable Local v	Basura
		235	Variable Local v	
		236	Variable Local z	Basura
		237	Variable Local z	
	BP->	238	Valor de BP	248
		239	Valor de BP	
		240	Dirección de Retorno	46~50
		241	Dirección de Retorno	
		242	Parámetro y	3
		243	Parámetro y	
		244	Parámetro x	6
		245	Parámetro x	
Pertenecen a la función f		246	Variable Local r	3
		247	Variable Local r	
		248	Valor de BP	Basura
		249	Valor de BP	
		250	Dirección de Retorno	26
		251	Dirección de retorno	
		252	Parámetro q	6
		253	Parámetro q	
		254	Parámetro p	9
		255	Parámetro p	

La dirección de retorno ahora sería justo después del llamado a g y si nos guiamos por el código inicial seria la posición 46~50. Los parámetros corresponderían al valor de q y r en el llamado de f por lo que serían 6 y 9%6 = 3, el valor de BP seria 238 y las variables locales z y v aun no tendrían un valor por lo que tendrían basura.

En resumen, sería lo siguiente:

Dato	Valor	Ubicación en el stack
BP	238	Almacena su valor anterior en 238 y 239 (valor anterior 248)
SP	234	---
Dirección de retorno	46~50	240 y 241
Parámetro y	3	244 y 245
Parámetro x	6	242 y 243
Variable local z	Basura	236 y 237
Variable local v	Basura	234 y 235

c) Ya que en la función g, el parámetro x era 6 y el parámetro y era 3 y 6 es mayor a 3 entonces entra la funcion al segundo if. Por lo que f será llamado con los parámetros z = 6-3 = 3 y parámetro y que es 3.

Al momento de llamar a f, el SP esta en 234 se hacen primero 2 PUSHs por lo que queda en 230, luego se llama a f con el call por lo que SP queda en 228 y luego se hace un PUSH para el BP por lo que SP queda en 226, el BP queda con el mismo valor que SP y luego al SP se le resta 2 para darle un espacio a la variable local r quedando así en 224. Por lo que quedaría así:

Pertenecen a la función f	SP->	224	Variable Local r	Basura
		225	Variable Local r	
	BP->	226	Valor de BP	238
		227	Valor de BP	
		228	Dirección de Retorno	66~70
		229	Dirección de retorno	
		230	Parámetro q	3
		231	Parámetro q	
		232	Parámetro p	3
		233	Parámetro p	
Pertenecen a la función g		234	Variable Local v	Basura
		235	Variable Local v	
		236	Variable Local z	3
		237	Variable Local z	
		238	Valor de BP	248
		239	Valor de BP	
		240	Dirección de Retorno	46~50
		241	Dirección de Retorno	
		242	Parámetro y	3
		243	Parámetro y	
		244	Parámetro x	6
		245	Parámetro x	
Pe		246	Variable Local r	3

		247	Variable Local r	
		248	Valor de BP	Basura
		249	Valor de BP	
		250	Dirección de Retorno	26
		251	Dirección de retorno	
		252	Parámetro q	6
		253	Parámetro q	
		254	Parámetro p	9
		255	Parámetro p	

Como se dijo antes el parámetro x e y tendrían el valor 3.
La dirección de retorno no quedaría clara ya que en ASM no puedes hacer retornar una función, sino que debes hacerlo por separado, pero sería un numero entre 66 y 70. La variable local r no han sido inicializada aun por lo que contienen basura.

3. Con respecto a cada uno de los siguientes modos de direccionamiento: ***inmediato***, ***directo***, ***por registro***, ***indirecto por registro***, e ***indexado***, responde lo siguiente:

Suponiendo que hay un arreglo de n números enteros en memoria, ¿qué dificultades y/o qué facilidades te entrega cada modo de direccionamiento para poder sumar los n números? Sé preciso en tus respuestas. Considera que cuentas con las instrucciones del lenguaje assembly x86 similares a la del computador básico.

Respuesta.

D. inmediato: El argumento de la instrucción no es la dirección del dato sino el dato propiamente tal. No hay cómo hacer un código general que sume n números: es necesario escribir n instrucciones SUM, cada una acompañada por uno de los n números. Dos ejecuciones del programa suman los mismos n números.

D. directo: El argumento de la instrucción es la dirección que el dato ocupa en memoria; pero esta dirección es fija, por lo que hay que conocerla de antemano y especificarla explícitamente para cada dato. También son necesarias n instrucciones SUM, cada una acompañada por la dirección en memoria de uno de los números. Dos ejecuciones del programa pueden sumar dos conjuntos distintos de n números.

D. por registro: El argumento de la instrucción es el registro en que está el dato. Es necesario ir cargando en el registro cada uno de los n números … pero ¿cómo? Se necesita otro tipo de direccionamiento para instrucciones LOAD, en cuyo caso (tal vez) mejor usamos este otro direccionamiento para la suma.

D. indirecto por registro: El argumento de la instrucción es el registro en el que está la dirección de memoria del dato; puede hacer referencia a diferentes direcciones de memoria en cada ejecución de la instrucción (guardando una dirección diferente en el registro cada vez), lo que permite recorrer fácilmente el arreglo de números.

D. indexado: El argumento de la instrucción está compuesto de dos partes: una dirección de memoria constante (p.ej., la del primer dato del arreglo) y un registro; la dirección especificada por el argumento es la suma de la dirección constante más el valor almacenado en el registro, que puede variar en cada ejecución de la instrucción, permitiendo recorrer fácilmente el arreglo de números.



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2343 — Arquitectura de Computadores — 1/2018
I2

Martes 8 de Mayo de 2018

La interrogación se compone de 3 preguntas. Debe responder cada una en una hoja separada, aunque puede utilizar más de una hoja para responder a una pregunta. Si no responde a una pregunta, debe entregar una hoja en blanco para ella. Recuerde poner su nombre en todas las hojas de respuesta.

Al momento de entregar, evite doblar bordes o corchetear las hojas, si no que sencillamente indique cuántas hojas son para esa pregunta en caso de utilizar más de una y procure que su hoja esté lo más lisa posible.

1. Responda brevemente las siguientes preguntas

   1.1) ¿Qué consideraciones se debe tener cuando un computador soporta más de una ISA? ¿Para qué se puede hacer esto? ¿Qué partes de la microarquitectura requieren atención? **(1 pt)**
   Solución:
   En primer lugar se debe tener en mente en qué ISA comenzará el computador, ya que usualmente el computador se encuentra con una ISA activa a la vez. Por otro lado, se debe considerar si el soporte es mediante alguna emulación o la microarquitectura soporta nativamente los elementos de esta ISA.
   Sobre el para qué se hace, es posible pensar en el caso de x86 donde la ISA fue progresando junto con los avances para permitir nuevas aplicaciones que utilizaran las nuevas características del hardware, como números más grandes, sin embargo se requería que los programas hechos para los procesadores anteriores pudieran ejecutar sin problemas.
   Finalmente, la unidad de interés en este escenario es la *Unidad de Control* del computador.

   1.2) Suponga que tiene una *caché* con mapeo directo. ¿Se demora más esta en sustituir una línea que una caché *fully associative*? ¿Por qué? **(1 pt)**
   Solución:
   No, se demora menos porque no tiene que aplicar sustitución. El mapeo directo indica de inmediato cuál línea sustituir, mientras que *fully associative* requiere de un protocolo de sustitución, lo cual exige revisar cada entrada para buscar el primero en entrar (FIFO), el menos usado (LFU) o el usado hace más tiempo (LRU).

   1.3) ¿Bajo qué condiciones puede ser conveniente un esquema de *polling* frente a uno de interrupciones, en un contexto de *Input/Output*? **(1 pt)**
   Solución:
   Frente a situaciones en que se sabe que la velocidad de entrada o salida es baja, por ejemplo, en el caso de un teclado o mouse.

   1.4) Al implementar *memory-mapped* I/O en el computador básico, ¿qué buses se deben intervenir y por qué? **(1 pt)**
   Solución:
   En principio se intervendrán los buses Memory Address, W, Data In y Data Out de la memoria, esto ya que entre todos ellos se instalará el *Address-decoder* que dependiendo de la dirección entrante por Address y el estado del bit W, eligirá si traspasar la señal y datos a la RAM o enviarlo a algún dispositivo I/O. Esto se realiza en este nivel ya que permite que el resto de la microarquitectura no tenga necesidad de ser intervenida para funcionar con este esquema de I/O.

   1.5) ¿Cómo se clasifica la ISA x86? Justifique y comente si esto tiene alguna implicancia para la microarquitectura. **(1 pt)**

Solución:
Se clasifica como una ISA *CISC*, es decir de instrucciones complejas. Sobre las implicancias para la microarquitectura, intuitivamente uno podría pensar en que requiere para una correcta implementación a lo menos registros para tomar las funciones de AX, BX, CX, DX, SI, DI, SP y BP.

1.6) ¿Por qué es deseable incluir una jerarquía de *cachés* en un computador? **(1 pt)**
Solución:
Debido a que la memoria es más lenta que los registros (y la caché). También, los principios de localidad espacial y temporal sugieren que es buena idea tener una jerarquía, ya que las mayores capacidades en los niveles superiores pueden compensar la penalidad de *miss* en los niveles inferiores.

2. Suponga que dispone de un computador compatible con la ISA clásica de x86 de 16 bits. En este computador se desea conectar una mesa de control que posee 25 botones y 10 controles que giran y tiene la particularidad de que permite que otros dispositivos compatibles, como un *joystick*, se conecten a esta. Para este dispositivo I/O:

2.1) Entregue un esquema de sus conexiones internas y conexión con el computador, usando *ports*. **(2 pts)**

2.2) Diseñe una tabla de interrupciones y mapeos adecuada. **(2 pts)**

2.3) Escriba una ISR que permita controlar este dispositivo. **(2 pts)**

3. Considere el siguiente registro de accesos de dos procesos en un computador con direcciones virtuales y con uso de **TLB** (*Translation Lookaside Buffer*):

Acceso	Proceso	Dirección Virtual	Página	TLB	Tabla de página
1	2	000101011	1	*Miss*	*Page Fault*
2	1	000111101	1	*Miss*	*Page Fault*
3	1	001101011	3	*Miss*	*Page Fault*
4	1	001011111	2	*Miss*	*Page Fault*
5	1	010011110	4	*Miss*	*Page Fault*
6	1	000110101	1	*Hit*	Página encontrada
7	1	000111011	1	*Hit*	Página encontrada
8	1	001010010	2	*Hit*	Página encontrada
9	1	001111011	3	*Hit*	Página encontrada
10	1	010010101	4	*Hit*	Página encontrada
11	1	010110100	5	*Miss*	*Page Fault*
12	1	001100101	3	*Miss*	Página encontrada
13	1	000111011	1	*Hit*	Página encontrada
14	2	000100101	1	*Miss*	*Page Fault*
15	1	001011101	2	*Miss*	Página encontrada
16	1	001111111	3	*Miss*	Página encontrada

Cuadro 1: Tabla de accesos a memoria.

Basándose en los siguientes datos:

- La memoria física posee 4 bloques.
- El *offset* utiliza 5 bits.
- La TLB y memoria física parten vacías.

Describa la TLB y la memoria virtual del computador. En particular, rellene con la página asociada a cada dirección virtual **(0.5 pts)** y luego responda las siguientes preguntas:

3.1) ¿Cuántas líneas posee la TLB? Justifique **(1 pt)**.
Respuesta:
Tiene 4 líneas la TLB, porque las primeras 4 páginas del programa 1 son *miss*, pero luego, esas 4 páginas son *HIT*. Esto significa que esas 4 líneas están en la TLB, pero cuando se solicita la página 5, es *miss* y después pido la 3 que estaba dentro y fue miss, es decir, fue sustituida esta entrada. Si fue sustituida significa que estaba llena con 4 páginas.

3.2) ¿Cúal es la política de sustitución de la TLB? Justifique **(1.5 pts)**
Respuesta:
Si vemos los hit y miss, en el acceso N°5, la TLB tiene las páginas 1, 2, 3 y 4. Luego llega la página 5, miss y cuando volvemos a preguntar por la página 3, ya no está. Esto implica que hubo un reemplazo de la página 5 por la 3. Si analizamos cada sustitución:

- LRU elimina el 1, porque las últimas páginas vistas son la 2 y 3
- FIFO elimina el 1, porque fue la primera página en entrar en la TLB
- LFU elimina 2, 3 o 4 porque tienen la misma cantidad de accesos

Lo importante es que tanto LRU y FIFO sustituyen la página 1, pero esa página fue Hit en el acceso 13, por lo tanto, se descartan esas 2 sustituciones y queda LFU como política de sustitución.

3.3) ¿Cuántas páginas soporta cada programa? **(0.5pts)**
Respuesta:

Con el offset se puede determinar el tamaño de cada página. Con la dirección más grande virtual se puede determinar el tamaño de la memoria virtual. La cantidad de páginas es la división entre ambos valores. Cada página tiene tamaño 2^5 y la dirección virtual tiene 2^9. Hay $2^9/2^5$ páginas = 16 páginas

3.4) ¿Cúal es el tamaño de la memoria física? Indique en bits o bytes. **(0.5 pts)**
Respuesta:
Con el offset, se puede determinar el tamaño de cada página y la cantidad de bloques de la memoria física ya está dicha. El tamaño de la memoria física es el tamaño de cada página por la cantidad de bloques. Es $2^5 * 4$ = 128 bytes

3.5) ¿Cúal es la política de sustitución en la memoria física? Justifique **(2 pts)**
Respuesta:
LFU.
Si vemos los page fault que provocan sustitución:

- Llega la página 4 del programa 2. El estado de la RAM en ese momento es:
  * El número aumenta el LRU (por cada acceso) para decir que el mínimo es el usado hace más tiempo.

Cuadro 2: RAM cuando llega pág.4 del prog. 2

Programa	Página	FIFO	LRU*	LFU
2	1	0	0	1
1	1	1	1	1
1	3	2	2	1
1	2	3	3	1

  Por los accesos 7,8,9 dicen que los bloque 1, 2 y 3 del programa 1 están en la memoria física. Por lo tanto se fue el bloque 1 del programa 2. Esto no aporta información porque es el primer bloque en llegar, el usado hace más tiempo y presenta misma cantidad de usos que los demás.
- Llega la página 5 del programa 1. El estado de la RAM en ese momento (luego de los accesos 6, 7, 8, 9 y 10) es:

Cuadro 3: RAM cuando llega pág. 5 del prog. 1

Programa	Página	FIFO	LRU*	LFU
1	4	4	9	2
1	1	1	6	3
1	3	2	7	2
1	2	3	8	2

  * El número aumenta el LRU para decir que el mínimo es el usado hace más tiempo.
  Por los accesos 12, 13 y 15, se dice que los bloque 1, 2 y 3 del programa 1 están en la memoria física. Por lo tanto se fue el bloque 4 del programa 1. Es decir, se fue el último bloque que entró y el recién usado. En otras palabras no puede ser FIFO o LRU. Además presenta la mínima cantidad de usos, por lo tanto LFU todavía sigue funcionando, solo que su desempate no es FIFO.
- Llega la página 1 del programa 1. El estado de la RAM en ese momento (después de los accesos 12 y 13) es:
  * El número aumenta el LRU para decir que el mínimo es el usado hace más tiempo.
  Por los accesos 15 y 16 dicen que los bloque 2 y 3 del programa 1 están en la memoria física. Por lo tanto se fue el bloque 1 o 5 del programa 1. Justamente el bloque 5 es el menos usado, por lo tanto LFU es el único algoritmo de sustitución posible en este problema.

Cuadro 4: RAM cuando llega pág. 1 del prog. 1

Programa	bloque	FIFO	LRU*	LFU
1	5	5	10	1
1	1	1	11	4
1	3	2	11	3
1	2	3	8	2

***Hint*:** Se recomienda entender qué pasa en cada acceso.

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



IIC2343 – Arquitectura de Computadores (II/2011)

Interrogación 3

Pregunta 1

a) Describa la función del northbridge y southbridge en la arquitectura x86. **(0,75 ptos.)**
Solución: El North Bridge es el controlador encargado de comunicar a la CPU, la memoria RAM, la tarjeta gráfica y al resto de los dispositivos de I/O, conectados con el South Bridge. El South Bridge es el encargado de comunicar a todo el resto de los I/O con la memoria y la CPU, a través del North Bridge, por lo que se le conoce como controlador de I/O.

b) Describa brevemente los pasos que se realizan cuando un programa realiza dinámicamente una petición de una nueva página de memoria. **(0,75 ptos.)**
Solución: Al ocurrir una petición, el programa le cede el control de la CPU al sistema operativo, el cual se encargará de mapear un marco físico disponible a una nueva página del programa, actualizando la tabla de páginas del programa. Si no hay marcos disponibles, se deberá hacer antes un proceso de swap-out para liberar un marco.

c) ¿Que datos nunca serán almacenados en la memoria caché de un computador con un sistema operativo que implementa multiprogramación? **(0,75 ptos.)**
Solución: Las tablas de páginas de los programas nunca serán almacenadas en la cache de la memoria principal, ya que los datos más usados de estás son almacenadas en la TLB.

d) Describa el peor caso posible, desde la óptica de la cantidad de accesos, que puede ocurrir cuando un programa intenta obtener un dato almacenado en memoria, en un sistema con memoria virtual y caché. **(0,75 ptos.)**
Solución: El programa quiere acceder a una dirección virtual, el primero paso siempre será ir a buscar la entrada de la tabla de páginas correspondiente a la TLB. En caso de que la TLB no tenga la entrada, ocurre una excepción denominada ***TLB miss exception***, la cual entrega el control al sistema operativo para que este vaya a buscar a la tabla de páginas del proceso actual la entrada y la copie a la TLB. Cuando el sistema operativo vaya a buscar la entrada, si esta está marcada que está en disco, ocurre un page fault y el SO debe hacer un swap in del marco correspondiente desde el swap file, reemplazar algún marco de memoria física, actualizar la tabla de páginas y copiar la entrada a la TLB. Una vez completado esto se repite el acceso a la TLB y ahora habrá un hit. Este es el **worst path** de acceso a memoria.

e) ¿Qué almacena el vector de interrupciones en la arquitectura x86? **(0,75 ptos.)**
Solución: El vector de interrupciones almacena las direcciones de memoria de las ISR (Interrupt Service Routine), que se ejecutan al gatillarse la interrupción correspondiente.

f) En un computador de 32 bits con 1 GB de RAM, se ejecuta un proceso que en cierto momento utiliza 1.5 GB de RAM. ¿Cómo es posible que ocurra esto sin que el proceso se caiga? Comente acerca del tiempo de ejecución de este proceso. **(0,75 ptos.)**
Solución: Esto se debe a que el espacio direccionable con 32 bits es de 4GB. Al utilizar memoria virtual, un proceso tiene acceso a todo este espacio, sin importar la cantidad de memoria física existente. Luego, si la memoria física se acaba, se utilizará espacio en disco. Por este motivo, el tiempo de ejecución del proceso será altísimo, ya que el acceso a disco es órdenes de magnitud más lento que el de memoria.

g) Se tiene una memoria caché, con función de correspondencia de mapeo directo, de 64KB con bloques de 16 palabras de 1 byte y una memoria principal de 32MB. La CPU quiere leer el dato de la memoria principal cuya dirección es 0x0104. Asumiendo que los bloques parten desde la dirección 0000h ¿En cuál bloque de la caché debería estar este dato? **(0,75 ptos.)**
Solución: La dirección 0x0104 corresponde a la 260 en decimal y se encuentra en el bloque 16 ($\lfloor \frac{260}{16} \rfloor$) de la memoria física. Luego, como la caché es de 64KB, para las direcciones desde la 0 hasta la 65535, su bloque de la memoria física y de la caché tendrán el mismo número. De esta manera, el bloque en la caché de la dirección 0x0104 es el 16.

h) Escriba 2 motivos por los cuales el acceso a los dispositvos de I/O se realiza comúnmente a través de interrupciones de software. **(0,75 ptos.)**
Solución:

I. El acceso a los dispositivos de I/O es posible realizarlo generalmente sólo cuando la CPU está en modo supervisor, por lo que los programas no pueden acceder directamente a ellos. Así, el mecanismo de interrupciones de software permite este acceso, protegiendo la comunicación.

II. Permite ahorrarse tener que implementar esa comunicación cada vez que se quiera acceder a un dispositivo de I/O.

Pregunta 2

Suponga que se tiene un computador con arquitectura x86, que tiene una memoria principal de 256 bytes, con el siguiente mapa de memoria:

Dirección	Función asociada
0-7	Exception handlers
8-15	Vectores de interrupciones de hardware
16 17-31	Vector de interrupción de escritura en disco Vectores de interrupciones de software
32-234	Memoria de uso libre
235 236-250	Registro de estado del disco duro (memory mapped) Buffer del disco duro (memory mapped)
251 252 253 254 255	Registro de controlador DMA: estado del controlador (memory mapped) Registro de controlador DMA: dirección inicial de memoria a copiar (memory mapped) Registro de controlador DMA: dirección inicial de buffer del disco (memory mapped) Registro de controlador DMA: cantidad de palabras a transferir (memory mapped) Registro de controlador DMA: comandos (memory mapped)

Se desea escribir un programa que permita escribir una secuencia de datos en un archivo del disco duro, ocupando interrupciones de software:

a) Escriba la ISR asociada al vector de interrupción 16, que permita escribir una secuencia de palabras en el disco, **sin ocupar DMA**. Para esto, la subrutina asume que cuando fue llamada se cumple lo siguiente: **(2 ptos.)**

- La dirección de la primera palabra de memoria a ser copiado se encuentra almacenada en el registro BX.
- La cantidad de palabras a copiar se encuentra almacenada en el registro CX.
- La secuencia se escribirá a partir de la primera dirección del buffer y hasta que se complete la cantidad de palabras o este se llene.

b) Escriba un programa que escriba las palabras desde la dirección 50 a la 100 de memoria al disco duro, ocupando interrupción de software para acceder a la ISR antes implementada. Considere que cuando el buffer se llena, el controlador del disco copiará estos datos al medio magnético y avisará el término de la operación en su registro de estado, escribiendo el valor 1: mientras no se complete esta operación, no se pueden transferir más datos. **(2 ptos.)**

Solución a) y b):

```
org 100h

ISR16:
 MOV SI, 0          ; SI se ocupara como indice para recorrer memoria y buffer
 MOV DX,BX          ; DX almacena como backup la direccion inicial de memoria

start:              ; Loop que recorre arreglo a ser copiado
 CMP SI, CX         ; Revisar condicion de termino arreglo if(SI > CX) goto end
 JGE end

 MOV BX, DX         ; Recuperamos la direccion inicial de memoria
 MOV AL, [BX+SI]    ; AL = mem[BX+SI]
 MOV BX, 236        ; Se almacena en BX direccion de primera palabra del buffer
 MOV [BX + SI], AL  ; buffer[BX+SI] = AL

 INC SI              ; Incrementar iterador arreglo
 JMP start

end:
 IRET

program:
 MOV BX, 50         ; Indicamos que queremos transferir a partir de la palabra 50
 MOV CX, 15         ; Indicamos que queremos transferir 15 palabras (tamano buffer)

 INT 16             ; Llamamos a la ISR por primera vez

 CALL waitBuffer

 MOV BX, 65         ; Indicamos que queremos transferir a partir de la palabra 65
 MOV CX, 15         ; Indicamos que queremos transferir 15 palabras (tamano buffer)

 INT 16             ; Llamamos a la ISR por segunda vez

 CALL waitBuffer

 MOV BX, 80         ; Indicamos que queremos transferir a partir de la palabra 75
 MOV CX, 15         ; Indicamos que queremos transferir 15 palabras (tamano buffer)

 INT 16             ; Llamamos a la ISR por segunda vez

 CALL waitBuffer

 MOV BX, 95         ; Indicamos que queremos transferir a partir de la palabra 75
 MOV CX, 5         ; Indicamos que queremos transferir 15 palabras (tamano buffer)

 INT 16             ; Llamamos a la ISR por segunda vez

 RET

waitBuffer:
 MOV AL, [251]      ; Leemos registro de estado del controlador, mapeado a Mem[251]
 CMP AL,1           ; if(AL != 1) wait
 JNE waitbuffer
 RET
```

Puntaje a):

- (0,6 ptos) Recorrer arreglo de palabras de memoria
- (0,6 ptos) Obtener dato de memoria
- (0,8 ptos) Copiar dato de memoria a buffer del disco

Puntaje b):

- (0,8 ptos) Configurar registros para llamar a interrupcion de software (4 veces, 0,2 ptos cada vez)
- (0,8 ptos) Llamar a interrupcion de software (4 veces, 0,2 ptos cada vez)
- (0,4 ptos) Esperar que este disponible el buffer para volver a escribir. Si esta parte la hicieron dentro de la ISR también es válido.

c) Modifique lo que sea necesario en la ISR y el programa anterior de manera que ahora la transferencia de realice mediante DMA. Para hacerlo, debe configurar el controlador DMA y luego iniciar el proceso de transferencia escribiendo el valor 255 en el registro de comando del controlador DMA. Puede asumir que en caso de llenarse el buffer, el controlador DMA se encargará de reconfigurarse para continuar con el proceso de transferencia, y no necesita intervención de la CPU para eso. **(2 ptos.)**
Solución c):

```
org 100h

ISR16:
 MOV [252], BX      ; Configuramos la direccion inicial de memoria
 MOV [253], 236     ; Configuramos la direccion inicial del buffer
 MOV [254], CX      ; Configuramos la cantidad de palabras
 MOV [255], 255     ; Enviamos el comando para iniciar la transferencia
 IRET

program:
 MOV BX, 50         ; Indicamos que queremos transferir a partir de la palabra 50
 MOV CX, 50         ; Indicamos que queremos transferir 50 palabras (todo)
 INT 16             ; Llamamos a la ISR una vez
 RET
```

Puntaje 2.3:

- (0,3 ptos) Configurar la direccion inicial de memoria
- (0,3 ptos) Configurar la direccion inicial del buffer
- (0,3 ptos) Configurar la cantidad de palabras
- (0,4 ptos) Enviar el comando para iniciar la transferencia
- (0,4 ptos) Configurar registros para llamar a interrupcion
- (0,3 ptos) Llamar a ISR

Pregunta 3

El siguiente programa se ejecutó en un computador con arquitectura x86 que tiene una caché de 4 bloques de 2 palabras cada uno:

Dirección	Label		
0		MOV	[var2],1
1	loop1:	MOV	AL,[var2]
2		MUL	[var2]
3		CMP	[var1],AL
4		JL	end
5		INC	[var2]
6		JMP	loop1
7	end:	DEC	[var2]
8		RET	
9	var1	db	**?**
10	var2	db	0

Al ejecutar el programa completo se obtuvo la siguiente secuencia de accesos a memoria:

0-10-1-10-2-10-3-9-4-5-10-6-1-10-2-10-3-9-4-5-10-6-1-10-2-10-3-9-4-7-10-8

Esta secuencia de accesos a memoria generó los siguiente estados en la caché:

Dir	B0	B1	B2	B3
0	0-1			
10	0-1	10-11		
1	0-1	10-11		
10	0-1	10-11		
2	0-1	10-11	2-3	
10	0-1	10-11	2-3	
3	0-1	10-11	2-3	
9	0-1	10-11	2-3	8-9
4	0-1	10-11	2-3	4-5
5	0-1	10-11	2-3	4-5
10	0-1	10-11	2-3	4-5
6	6-7	10-11	2-3	4-5
1	6-7	10-11	0-1	4-5
10	6-7	10-11	0-1	4-5
2	6-7	10-11	0-1	2-3
10	6-7	10-11	0-1	2-3

Dir	B0	B1	B2	B3
3	6-7	10-11	0-1	2-3
9	8-9	10-11	0-1	2-3
4	8-9	10-11	4-5	2-3
5	8-9	10-11	4-5	2-3
10	8-9	10-11	4-5	2-3
6	6-7	10-11	4-5	2-3
1	0-1	10-11	4-5	2-3
10	0-1	10-11	4-5	2-3
2	0-1	10-11	4-5	2-3
10	0-1	10-11	4-5	2-3
3	0-1	10-11	4-5	2-3
9	8-9	10-11	4-5	2-3
4	8-9	10-11	4-5	2-3
7	6-7	10-11	4-5	2-3
10	6-7	10-11	4-5	2-3
8	8-9	10-11	4-5	2-3

En base a esta información, responda lo siguiente:

a) ¿Qué valores puede tener la variable `var1` para que efectivamente se genere la secuencia de accesos detallada previamente? **(1,5 ptos)**
Solución: El rango de valores es entre 4 y 8 (ambos inclusive).

Puntaje:

- Si el rango es correcto 1,5 ptos
- Si el rango inferior está correcto y el superior tiene un error de ± 1 (i.e se respondió 7 o 9) 0,5 ptos.
- Si el rango superior está correcto y el inferior tiene un error de ± 1 (i.e se respondió 3 o 5) 0,5 ptos.

b) ¿Cuál es el hit rate? **(0,5 ptos)**
Solución: El hit rate es $\frac{17}{32}$. 0,5 ptos si está correcto, 0 en otro caso.

c) ¿Qué tipo de caché es: unified o split? **(0,5 ptos)**
Solución: La cachés es unified. 0,5 ptos si está correcto, 0 en otro caso.

d) ¿Qué función de correspondencia y algoritmo de reemplazo (si corresponde) utiliza esta caché? **(1,5 ptos)**
Solución: La función de correspondencia es fully associative (0,5 ptos) y el algoritmo de reemplazo es random(1,0 ptos). También es válido indicar en que partes el algoritmo de reemplazo obedece a que tipo, por ejemplo, LFU desde el acceso N hastal em M, LRU desde X a Y.

e) ¿Es posible mejorar el desempeño de esta caché durante la ejecución de este programa, sin modificar la cantidad y tamaño de los bloques? Si es posible, explique una posible mejora que se podría realizar para lograr un mejor hit rate que el actual y demuestre que efecticamente su modificación lográ mejorar el hit rate. Si no es posible, justifique porqué. **(2,0 ptos)**
Solución: Hay dos formas de responder esta respuesta. La primera es modificar la función de corrrespondencia o algoritmo de reemplazo con alguno de los que se vio en clases. En este caso se obtiene 0,5 ptos por elegir una solución que efectivamente mejore el hit rate, y 1,5 por implementarla y demostrar que mejoró el hit rate. La segunda forma es inventar un algoritmo de reemplazo propio que sea específico al programa propuesto y mejore el hit rate. En este caso se obtiene 0,5 ptos por explicar el algoritmo propuesto y 1,5 ptos por implementarlo y demostrar la mejora

Pregunta 4

Asuma que una CPU tiene un espacio de direccionamiento virtual de 13 bits cuyas páginas son de 1KB. Esta máquina, sin embargo, cuenta tan sólo con 4KB de memoria RAM disponibles para marcos. Asuma que los marcos están inicialmente vacíos. En un momento comienza a ejecutarse un proceso (P1) el cual, durante su ejecución utiliza las direcciones de memoria desde la 0 hasta la 1500. Luego de esto el sistema operativo hace un cambio de contexto con lo que empieza a ejecutarse un segundo proceso (P2) el cual, durante su ejecución, utiliza las direcciones de memoria desde la 0 hasta la 500, y desde la 4500 a la 5000. Los datos que el proceso P2 almacena en éstas últimas direcciones (de la 4500 hasta la 5000) son compartidos por el proceso P3 (tanto para lectura como para escritura), el cual los accesa a través de las direcciones virtuales 2452 a la 2952. Se genera otro cambio de contexto y empieza a ejecutarse este tercer proceso (P3), el cual hace uso de las direcciones de memoria desde la 0 hasta la 1000, utilizando además datos desde la dirección 2500 a la 2600. Suponga que la política de reemplazo de páginas en los marcos es FIFO.

a) Determine, para cada marco, de qué proceso o procesos es la información y/o datos que contiene. También indique qué paginas, de haber, se encuentran en disco. **(1 pto.)**
Solución:

Antes de P3

P1
P1
P2
P2

Después de P3

P3	P1 en disco
P1	
P2	
P2/P3	

b) Escriba las tablas de página asociadas a estos tres procesos. **(1 pto.)**
Solución:

Proceso 1			
Pág.	Marco	Val.	Disco
0	0	0	1
1	1	1	0
2	x	0	0
3	x	0	0
4	x	0	0
5	x	0	0
6	x	0	0
7	x	0	0

Proceso 2			
Pág.	Marco	Val.	Disco
0	2	1	0
1	x	0	0
2	x	0	0
3	x	0	0
4	3	1	0
5	x	0	0
6	x	0	0
7	x	0	0

Proceso 3			
Pág.	Marco	Val.	Disco
0	0	1	0
1	x	0	0
2	3	1	0
3	x	0	0
4	x	0	0
5	x	0	0
6	x	0	0
7	x	0	0

Luego de un cambio de contexto el proceso P1 lee la dirección de memoria 500. Posterior a esto, el mismo proceso requiere escribir en la dirección de memoria 600.

c) Determine en qué dirección real se escribe en la memoria principal al escribir este proceso en la dirección 600. **(1 pto.)**
Solución: La dirección virtual 600 corresponde a la página 0 del proceso P1. Luego del último

cambio de contexto, esta página está mapeada al marco 1 de la memoria física. Luego, la dirección física correspondiente a la dirección virtual 600 será **1024 + 600 = 1624**.

d) Determine, para cada marco, de qué proceso o procesos es la información y/o datos que contiene. También indique qué paginas, de haber, se encuentran en disco. **(1 pto.)**
Solución:

P3
P1
P2
P2/P3

P1 en disco

e) Escriba las tablas de página asociadas a estos tres procesos. **(1 pto.)**
Solución:

Proceso 1			
Pág.	Marco	Val.	Disco
0	1	1	0
1	1	0	1
2	x	0	0
3	x	0	0
4	x	0	0
5	x	0	0
6	x	0	0
7	x	0	0

Proceso 2			
Pág.	Marco	Val.	Disco
0	2	1	0
1	x	0	0
2	x	0	0
3	x	0	0
4	3	1	0
5	x	0	0
6	x	0	0
7	x	0	0

Proceso 3			
Pág.	Marco	Val.	Disco
0	0	1	0
1	x	0	0
2	3	1	0
3	x	0	0
4	x	0	0
5	x	0	0
6	x	0	0
7	x	0	0

f) Indique en qué direcciones físicas, de estar, se encuentran las direcciones virtuales: **(1 pto.)**

- 2500 del proceso P3
  Solución: 3524
- 2000 del proceso P1
  Solución: No hay dirección física, ya que se encuentra en disco.
- 4548 del proceso P2.
  Solución: 3524

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



IIC2343 – Arquitectura de Computadores (I/2012)

Interrogación 3

Pregunta 1

a) ¿Que problema de consistencia en la escritura en memoria podría ocurrir si la memoria caché se ubicara antes de la MMU, i.e. la memoria caché trabaja con direcciones virtuales? **(1 pto.)**

Solución: Si la caché se ubica antes de la MMU, entonces este trabaja con direcciones virtuales. De esta manera, en un instante dado, en la caché pueden existir dos bloques asociados a paginas virtuales diferentes, pero que apuntan al mismo bloque. Así, si se escribe en uno de estos bloques, las páginas tendrán distinto contenido, a pesar de estar mapeadas al mismo bloque.

b) Si dos procesos comparten una porción de memoria menor al tamaño de un marco, ¿existe algún problema si uno de los procesos escribe en un porción no compartida del mismo marco? **(1 pto.)**

Solución: Dado que un marco no puede ser subdividido, cualquier escritura en el será visible para los dos procesos, sin importar si esta fue dentro del área compartido o no.

c) ¿Es posible reutilizar el contenido de la TLB cuando ocurre un cambio de contexto? **(1 pto.)**

Solución: Dado que las asignaciones en la TLB, al hacer un cambio de contexto, no necesariamente corresponden a la tabla de páginas del proceso que se encuentra actualmente ejecutándose, la TLB debe ser borrada para que se llene con mapeos válidos.

d) Asumiendo que la caché en el primer nivel de una jerarquía de memoria es muy rápida y muy pequeña, ¿que función de correspondencia recomendaría en este caso? **(1 pto.)**

Solución: Se recomendaría usar una caché fully associative.

e) Un programa necesita copiar a otra posición de memoria (duplicar) una matriz que pesa 1 GB. ¿Como debería realizar este proceso el programa? **(1 pto.)**

Solución: El proceso debería utilizar DMA, ya que la memoria, por defecto, está mapeada a memoria. De esta manera, la CPU no debe preocuparse de la transferencia.

f) Un computador con microarquitectura avanzada realiza la búsqueda en la TLB al mismo tiempo que la búsqueda en el caché, en vez de hacerlo de manera secuencial, i.e., primero en la TLB y

luego en la caché. ¿Que relación debe cumplirse, con respecto al tamaño de página y la cantidad de bits necesaria para encontrar un bloque en la caché, para que este esquema presente una mejora en el rendimiento? **(1 pto.)**

Solución: Debe cumplirse que la cantidad de bits necesarias para buscar una palabra dentro de una página sea la misma que la cantidad de bits necesaria para realizar una búsqueda en la caché (sin considerar el tag). De esta manera, al hacer la transformación virtual-física, el resultado de esta que no corresponde al offset, se utiliza para verificar si el tag del bloque obtenido en el caché coincide.

Pregunta 2

Considere un computador de 32 bits, con páginas de memoria de 4K palabras y memoria virtual de dos niveles. El primer nivel consiste en un diccionario de tablas de páginas, mientras que el segundo contiene las tablas. Este esquema permite disminuir el espacio usado por las tablas de páginas, ya que la única que debe estar siempre en memoria es la correspondiente al diccionario.

a) Describa la subdivisión de una dirección de memoria virtual con este esquema. **(2 ptos.)**

Solución: Dado que el diccionario siempre debe estar en memoria, este deberá estar almacenado en una página de memoria. Luego, este puede tener un tamaño máximo de 4K entradas por lo que se necesitan 12 bits para especificar una entrada en el diccionario. Cómo los márcos son de 4K palabras, el offset también debe ser de 12 bits. Finalmente, dado que el computador es de 32 bits, la tabla de páginas sólo puede tener 2^8 entradas. Gráficamente, la división queda de la siguiente forma:

Índice en Diccionario	Índice en Tabla de Páginas (Página)	Offset dentro del Marco
12 bits	8 bits	12 bits

b) Explique los pasos que deben realizarse para traducir una dirección de memoria virtual a una física (no tome en cuenta los posibles accesos a disco). **(2 ptos.)**

Solución: El primer paso consiste en buscar en el diccionario, utilizando los 12 bits más significativos de la dirección virtual, la tabla de páginas asignada a ese índice. Una vez obtenida identificada la página, se obtiene esta desde la memoria. Cabe destacar que en cada marco de memoria caben 2^4 tablas de páginas. Luego, se busca dentro de la página el marco correspondiente a la página indicada por el índice ubicado en los bits [12-19] de la dirección virtual. Finalmente, a la dirección del marco obtenido en el paso anterior se le concatena el offset que se ubica en los 12 bits menos significativos de la dirección virtual, con lo que se obtiene la dirección física.

c) ¿Cuantas operaciones de memoria son necesarias para leer o escribir una palabra? **(1 pto.)**

Solución: Se necesitan 3 operaciones. Una para leer el diccionario, otra para la tabla de paginas y una final para el marco.

d) ¿Cuanta memoria física necesita un proceso que utiliza una página de memoria virtual? **(1 pto.)**

Solución: Si se necesita una sola página de memoria, se requiere tener un diccionario, una tabla y un marco. Luego, para el diccionario se necesitan 4KB (1 marco-página), mientras que para una tabla se necesitan 2^8B= 256B y para un marco 4KB. Así, para un proceso de una sóla página, se necesitan 4096B+4096B+256B= 8448B.

Pregunta 3

a) Un computador tiene 16 MB de memoria direccionable y una caché de 128 KB, con función de correspondencia 2-way associative y política de sustitución LRU, y con bloques de 16 palabras de 1 byte. Asuma que la caché está vacia y que se ejecuta un programa con la siguiente secuencia de accesos a direcciones de memoria: 3C2B20h, 3C2324h, 172B24h, 3C2B28h, 182B27h, 258032h, B22328h, 172328h. Escriba para cada uno de los accesos su correspondiente tag, conjunto y bloque en la caché. **(3 ptos.)**

Solución:
Tamaño de la caché = 128 KB = $16 \times 2 \times$N, luego son N = 4096 sets conjuntos.
Bits por dirección = $\log_2(16\text{MB}) = \log_2(2^{24}) = 24$
Bits de direccionamiento por bloque = $\log_2(16) = 4$, luego se usan los 4 bits menos significativos para direccionar una palabra dentro de un bloque.
Bits de direccionamiento para conjuntos = $\log_2(4096) = 12$, por lo tanto se utilizan los 12 bits a continuación de los 4 bits menos significativos para direccionar un conjunto.
Bits de direccionamiento para tag = 24–4–12 = 8
Accesos:

1. **3C2B20h**: Se ubica en el bloque 0 del conjunto 2B2h, con tag 3Ch.
2. **3C2324h**: Se ubica en el bloque 0 del conjunto 232h, con tag 3Ch.
3. **172B24h**: Se ubica en el bloque 1 del conjunto 2B2h, con tag 17h (bloque 0 del conjunto ya estaba ocupado).
4. **3C2B28h**: Hit en el bloque 0 del conjunto 2B2h con tag 3Ch.
5. **182B27h**: Se ubica en el bloque 1 del conjunto 2B2h, con tag 18h (se saca el bloque con tag 17h, por política LRU).
6. **258032h**: Se ubica en el bloque 0 del conjunto 803h, con tag 25h.
7. **B22328h**: Se ubica en el bloque 1 del conjunto 232h, con tag B2h.
8. **172328h**: Se ubica en el bloque 0 del conjunto 232h, con tag 17h (se saca el bloque con tag 2Ch, por política LRU).

b) Un computador con soporte para memoria virtual tiene una jerarquía de memoria consistente en una memoria caché, memoria RAM y disco duro. La obtención de una palabra de la caché toma 20ns, mientras que si la palabra no está en la caché pero si en la memoria RAM, se necesitan 60ns para cargarla en la caché desde la memoria. Finalmente, si la palabra no se encuentra en la memoria RAM, se necesitan 60ms para obtenerla desde el disco y cargarla en la memoria RAM. Si el hit-rate promedio de la caché es 0.9 y el de la memoria RAM es de 0.99, ¿cuál es el tiempo promedio que toma el obtener una palabra en este sistema? **(3 ptos.)**

Solución: Luego:

- Tiempo acceso primedio = Hit-Time(caché) + Miss-Rate(caché) × Miss-Penalty(caché)
- Hit-Time(caché) = 20ns, Miss-Rate(caché) = 1 - 0.9 = 0.1
- Miss-Penalty(caché) = Hit-Time(memoria) + Miss-Rate(memoria) × Miss-Penalty(memoria)
- Hit-Time(memoria) = 60ns, Miss-Rate(memoria) = 1 - 0.99 = 0.01
- Miss-Penalty(memoria) = Hit-Time(disco) + Miss-Rate(disco) × Miss-Penalty(disco)
- Hit-Time(disco) = 12ms = 12×10^6ns, Miss-Rate(disco) = 0

Luego, el tiempo de acceso promedio es: $20ns + 0{,}1 \times (60ns + 0{,}01 \times (12 \times 10^6 ns + 0)) = 12026ns$

Pregunta 4

Suponga que se tiene un computador con arquitectura x86, que tiene una memoria principal de 1024 bytes, con el siguiente mapa de memoria:

Dirección	Función asociada
0-7	Exception handlers
8-15	Vectores de interrupciones de hardware
16	Vector de interrupción de dibujo en pantalla
17-31	Vectores de interrupciones de software
32-123	Memoria de uso libre
124-1023	frame buffer (lo que se dibuja en pantalla) de la tarjeta de video (memory mapped)

Se desea escribir una subrutina que permita dibujar líneas horizontales o verticales en una pantalla de 30x30 pixeles, mediante el uso de interrupciones de software. Cada pixel de la pantalla puede tomar 256 tonalidades de gris, donde 0 representa el negro y 255 el blanco. Se asume que el pixel (0,0) se encuentra en la esquina superior izquierda de la pantalla.

a) Escriba la ISR asociada al vector de interrupción 10h, que permita dibujar una serie de pixeles en pantalla. Para esto, la subrutina asume que cuando fue llamada se cumple lo siguiente: **(4 ptos.)**

- La columna y fila de inicio de la línea se encuentra almacenada en el registro BX, donde la fila se ubica en BL y la columna en BH.
- El largo de la línea, en pixeles, se encuentra almacenado en los 7 bits menos significativos del registro CL, mientras que la dirección de la línea se encuentra almacenada en el bit más significativo de CL. Si es 1 la línea es horizontal, si es 0 es vertical.
- El color de la línea, se encuentra almacenado en el registro CH.

Solución: Se asume que el frame buffer se organiza en orden de filas

```
ISR10h:
  MOV AL, 30        ; Se calcula el inicio de la linea
  MUL BH            ; y se almacena en BX
  MOV DX, AX
  ADD DX, BL
  ADD DX, 124
  MOV BX, DX

  MOV DI, 1         ; Calculamos el salto por pixel en
  MOV AL, CL        ; memoria (1 o 30), dependiendo si
  SHR AL, 7         ; es vertical o horizontal. Se almacena
  CMP AL, 1         ; en DI
  JE end_if
  MOV DI, 30

end_if:
  SHL CL, 1         ; Se ajusta CL para que represente
  SHR CL, 1         ; el largo.
  MOV SI, 0         ; Se inicializan los contadores
  MOV AL, 0
```

```
for:
  CMP AL, CL       ; Se verifica la condicion de fin
  JE end
  MOV [BX+SI], CH ; Se dibuja el pixel
  ADD SI, DI
  INC AL
end:
  IRET
```

b) Escriba un programa que dibuje un cuadrado, cuyos lados miden 10 pixeles, en el centro de la pantalla. Además, el color de cada lado deberá diferenciarse del de su vecino por lo menos en 20 tonalidades de gris. **(2 ptos.)**

Solución:

```
  org 100h

  MOV BH, 10       ; Fila
  MOV BL, 10       ; Columna
  MOV CH, 20       ; Color
  MOV CL, -118     ; Largo diez y horizontal (1000 1010)
  INT 10h          ; Llamamos a la ISR de dibujo

  MOV BH, 19       ; Fila
  MOV BL, 10       ; Columna
  MOV CH, 60       ; color
  MOV CL, -118     ; Largo diez y horizontal (1000 1010)
  INT 10h          ; Llamamos a la ISR de dibujo

  MOV BH, 10       ; File
  MOV BL, 10       ; Columna
  MOV CH, 80       ; Color
  MOV CL, 10       ; Largo diez y vertical (000 1010)
  INT 10h          ; Llamamos a la ISR de dibujo

  MOV BH, 10       ; Fila
  MOV BL, 19       ; Columna
  MOV CH, 40       ; Color
  MOV CL, 10       ; Largo diez y vertical (000 1010)
  INT 10h          ; Llamamos a la ISR de dibujo

  RET
```

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



IIC2343 – Arquitectura de Computadores (II/2012)

Solución Interrogación 3

Pregunta 1

a) Explique y describa el concepto de jerarquía de memoria. Describa un esquema en que se use una jerarquía de tres niveles y detalle cada uno de estos. **(1 pto.)**

Solución: La jerarquía de memoria de un computador se basa en el uso de diversos niveles de memoria, cada uno de los cuales ocupa una tecnología distinta, aprovechando las ventajas de cada tecnología: para almacenar mucha información se utilizan tecnologías que tienen un costo bajo por Byte, como los discos magnéticos; para acceder rápidamente a información se utilizan tecnologías electrónicas de rápido acceso, como SRAM (static RAM); para niveles intermedios se utilizan tecnologías con velocidades de acceso y costos intermedios, como DRAM (dynamic RAM).

b) En un computador con arquitectura x86, un dispositivo defectuoso genera interrupciones aleatorias que degradan fuertemente el rendimiento del sistema. ¿Es posible eliminar este problema sin desconectar el dispositivo? **(1 pto.)**

Solución: Si, es posible hacerlo mediante el enmascaramiento de interrupciones del PIC.

c) ¿Qué ventaja tiene una caché del tipo **split** sobre una **unified**? **(1 pto.)**

Solución: La caché split tiene zonas independientes para datos e instrucciones, por lo que puede aprovechar de mejor manera los patrones de localidad espacial y temporal de datos e instrucciones, que la gran mayoría de las veces son distintos.

d) Un fabricante de hardware sacará prontamente al mercado una tarjeta de video (GPU) con 4 GB de RAM. ¿Qué mapeo de memoria le recomendaría al fabricante para poder interactuar con este dispositivo? **(1 pto.)**

Solución: Se le recomienda utilizar direcciones de memoria independientes para sus registros de comandos y estado, y además mapear sólo un parte de la memoria de video a la memoria principal (para no saturar esta) y dejar que la tarjeta de video se encargue y DMA se encarguen de completar las transferencias.

e) Dada un memoria caché de 1 KB con bloques de 16 palabras de 1 byte y función de correspondencia **fully associative**, que se encuentra llena, ¿cuantas comparaciones secuenciales se deben realizar para decidir que bloque sustituir? Describa además un esquema que permita minimizar el tiempo requerido para este proceso, usando la misma cantidad de comparaciones. **(1 pto.)**

Solución: La caché tiene 64 bloques, por lo que serán necesarias 63 comparaciones secuenciales. Si estas se hacen se manera paralela, de 2 en 2, se realizan las mismas 63, pero demorarían aproximadamente $frac110$ del tiempo.

f) Describa una posible solución al problema de consistencia entre caché y RAM del esquema **write-back**. **(1 pto.)**

Solución: Es posible solucionar esto agregando un bit de consistencia a cada bloque de la memoria RAM. Si el bloque ha sido modificado en la caché y no en la RAM, se setea este bit en 0. Luego, en caso que se requiera utilizar ese bloque por algún elemento del computador (ej. controlador DMA), si el bit de consistencia está en 0, deberá actualizarse el bloque en la memoria RAM.

Pregunta 2

a) Considere una jerarquía de memoria dada por memoria RAM y caché, donde ambas utilizan palabras de 16 bits (16 bits = 2B = W). La memoria RAM tiene 1 GW (G = 1 Giga = 1024*1024*1024) de capacidad, mientras que la caché una capacidad 1 MW (M = 1 Mega = 1024*1024) y bloques de 256W. Responda las siguientes preguntas, asumiendo que la caché utiliza una función de correspondencia **4-way associative**:

- Calcule el número de bloques y conjuntos de la memoria caché. **(0.5 ptos.)**

  Solución: La cachés de 1 MW y tiene bloques de 256W, por lo tanto tiene $\frac{1024 \times 1024}{256} = 4096$ bloques. Luego, si la caché es **4-way associative**, tiene $\frac{4096}{4} = 1024$ conjuntos.

- Describa la división de la direcciones de memoria, incluyendo offset dentro del bloque, conjunto y tag. **(0.5 ptos.)**

  Solución: La memoria principal es de 1 GW, por lo tanto tiene 2^{30} palabras de 16 bits, luego, las direcciones de memoria tienen 30 bits. Como los bloques son de 256W y tenemos además 1024 conjuntos, necesitamos los 8 bits menos significativos de la dirección para el offset y los 10 bits siguientes para el conjunto. Por lo tanto, los 12 bits más significativos se utilizan para el tag.

b) Considere un computador con microarquitectura Von Neumann, donde la tasa de ciclos de clock por instrucción es igual a **1.0** cuando todos los accesos a memoria producen hits en la caché. Además, esta memoria caché tiene un **miss rate** de 2 % y **miss penalty** de 25 ciclos de clock. Si en un programa el 50 % de las instrucciones corresponde a lectura/escritura de memoria, ¿cuánto más rápida sería la ejecución del programa si todas las instrucciones del programa produjeran hits en la caché? **(2 ptos.)**

Solución: Si se asume un programa de 100 instrucciones, sabemos que cuando el hit-rate es 100 %, el programa demorará 100 ciclos. Si tenemos ahora una situación donde el hit rate es 98 %, como en la arquitectura Von Neumann cualquier instrucción requiere al menos un acceso a memoria (fetch), se tiene que 2 instrucciones, de un total de 100, tendrán un caché miss. Además de esto, se sabe que el 50 % de las instrucciones son de escritura/lectura, por lo tanto, como el 98 % de 50 es 1, tendremos 1 instrucción que generará un miss cuando quiera acceder los datos. Sumando, tenemos 97 instrucciones con hit y 3 con miss, por lo que el tiempo total será de 175 ciclos, por lo que la ejecución cuando el hit-rate es 100 % será 1.75 veces más rápida.

c) Un computador de 32 bits tiene una caché, sólo para datos, de 32 KB con bloques de 64 palabras de 1 byte y función de correspondencia **2-way associative**, además de política de sustitución **LFU**. Considere el siguiente programa:

```
int m[512][512];
int sum = 0;
for (int i = 0; i < 512; i ++)
{
    for (int j = 0; j < 512; j++)
    {
        sum += m[i][j];
    }
}
for (int i = 0; i < 512; i ++)
{
    for (int j = 0; j < 512; j++)
    {
        sum += m[j][i];
    }
}
```

Asumiendo que la variable *sum* y los contadores i y j se almacenan en registros y que la matriz m se almacena en memoria en orden de filas, calcule el hit rate que produce el programa anterior en la caché. **(3 ptos.)**

Solución: De los datos del enunciado, se puede extraer que la caché tiene 512 bloques y 256 conjunto. Además, como un dato de tipo *int* utiliza 4 bytes (32 bits), cada bloque podrá contener 16 ints. De la misma manera, para almacenar una fila de la matriz se necesitarán 32 bloques de la caché, mientras que para llenar la caché se necesitan sólo 16 filas de la matriz. Teniendo esto en consideración y sabiendo que la matriz es almacenada por filas, se puede notar que 1 de cada 16 lecturas en el primer **doble-for** producirá un miss, mientras que las otras 15 serán hits, ya que el bloque (16 ints) estará almacenado en la caché debido al miss previo. Por lo tanto, el hit-rate luego del primer **doble-for** es de $\frac{15}{16}$. Para el segundo **doble-for** la situación es distinta, pero más simple. Dado el tamaño de la matriz y la distribución de conjuntos y bloques en esta, el recorrido en orden de columnas no generará nunca un hit en la caché, ya que los bloques escritos en esta serán sustituidos antes de poder ser accedidos nuevamente. Por lo tanto, en este caso el hit-rate es 0. Luego, si sumamos los dos casos, tenemos que el hit-rate total es $\frac{15}{16} + \frac{0}{16} = \frac{15}{32}$.

Pregunta 3

Suponga que se tiene un dispositivo de adquisición de imágenes térmicas conectado a un computador que tiene una microarquitectura especializada para a 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.

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 donde se almacenan. **(4 ptos.)**

Solución: La solución utiliza la IRQ 17 y asume que recibe como parámetro la dirección de inicio de escritura en disco en el registro BX.

```
ISR17:
    INT 16
while:
    MOV AX, [7]
    CMP AX, F0h
    JE while

    MOV AX, 124
    MOV CX, 900
    INT 15
```

b) Escriba un programa que llame a la subrutina del item 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. **(2 pto.)**

Solución: La solución utiliza la IRQ 17 y asume que recibe como parámetro la dirección de inicio de escritura en disco en el registro BX.

```
        MOV DX, 0
        MOV BX, 1024
        MOV SI, 0
    while:
        CMP DX, 3
        JE end
        ADD DX, 1
        INT 17
        MOV AX, [7]
        CMP AX, 00h
        JE while
    end:
        MOV DX, 0
        ADD BX, 900
        ADD SI, 1
        CMP SI, 3
        JLT while
```

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



IIC2343 – Arquitectura de Computadores (I/2013)

Solución Interrogación 3

Pregunta 1

a) El principio de localidad espacial explica en parte el buen funcionamiento de la memoria caché. Sin embargo, es posible no cumplir este principio, disminuyendo el rendimiento de la memoria. Describa un ejemplo específico de esto y explique por qué se produce. **(1 pto.)**
Solución: Un ejemplo específico es escribir en memoria una matriz de gran tamaño (que las filas no quepan completamente en la caché) en orden de filas y luego, leer esta matriz en orden de columnas. De esta manera, datos que están espacialmente cercanos en la memoria RAM, no serán leídos en el orden corresondiente.

b) Describa las tres funciones de correspondencia vistas en clases. **(1 pto.)**
Solución:

- Mapeo Directo: Cada bloque de la memoria principal sólo puede almacenarse en un bloque fijo de la memoria caché, definido de acuerdo al tamaño de la caché y el tamaño de los bloques.
- Fully Associative: Cada bloque de la memoria principal puede ubicarse en cualquier bloque de la memoria caché.
- N-way Associative: La memoria caché es dividida en conjuntos de N bloques. Luego, cada bloque de la memoria principal sólo puede ubicarse en un conjunto de la caché, pero en ese conjunto puede ubicarse en cualquier bloque.

c) Describa que elementos de un sistema con soporte para multiprogramación (SO, CPU y Memoria) deben actualizarse al realizar un cambio de contexto. **(1 pto.)**
Solución: Al hacer un cambio de contexto debe actualizarse la TLB, los registros PTBR y de modo de la CPU. En caso de ser necesario, también debe hacerse swap-in de los marcos que hayan sido enviados a disco y deban ser utilizados por el proceso. Finalmente, el PCB del proceso saliente debe actualizarse con los valores de los registros y flags, para poder recuperar su estado en el futuro.

d) ¿Cómo debería modificarse la arquitectura del computador básico para que tenga soporte para multiprogramación? **(1 pto.)**
Solución: Debería agregarse una MMU a la CPU, y los registros PTBR y de modo.

e) ¿Cuántos dispositivos de I/O pueden conectarse a un computador x86? **(1 pto.)**
Solución: Se pueden conectar máximo 15 dispositivos, debido al esquema esclavo-maestro utilizado por los PICs.

f) En un computador x86, describa por cuáles elementos del hardware pasa una serie de datos transmitidos desde la memoria RAM a la tarjeta de video mediante DMA. **(1 pto.)**
Solución: Los datos deben pasar primero a través del bus de memoria para llegar al North-Bridge. Luego viajan a través del bus gráfico hasta la memoria de la tarjeta de video.

Pregunta 2

a)
- En un computador el tiempo promedio de acceso a los datos es de 5ns. ¿Cuál es el hit-time de la caché, si su hit-rate es 0.99 y el tiempo de acceso a la memoria RAM es 400 veces mayor al hit-time de la caché? **(0.5 ptos.)**
  Solución:

$$T_{prom} = HT_{caché} + MR_{caché} \times MP$$
$$5ns = x + 0{,}01 \times 400x$$
$$x = \frac{5ns}{1 + 0{,}01 \times 400}$$
$$x = 1ns$$

- En un computador con memoria caché de 16KB, bloques de 8 palabras y tiempo de acceso de 10ns, se transfiere desde la memoria principal un bloque completo en 120ns, ¿cuál es el hit-rate de la caché para obtener un tiempo de acceso promedio de 20ns? **(0.5 ptos.)**
  Solución:

$$T_{prom} = HT_{caché} + MR_{caché} \times MP$$
$$20ns = 10ns + (1 - x) \times 120ns$$
$$x = 1 - \frac{10ns}{120ns}$$
$$x \approx 0{,}92$$

b) Un computador posee una memoria principal de 32KB con palabras de 16 bits y una memoria caché de 8KB dividida en conjuntos de 4 bloques y 64 palabras por bloque. Asuma que la caché está inicialmente vacía y el procesador solicita las siguientes direcciones de memoria de manera secuencial: 1, 2, ..., 4072. Este proceso se repita en total 10 veces. Si la caché es 10 veces más rápida que la memoria principal y asumiendo una política de reemplazo LRU, estime la mejora en tiempo de acceso dada por el uso de la caché. **(2 ptos.)**
Solución: En este problema, dado que los accesos a memoria son secuenciales y son menos que el tamaño de la caché, nunca se deberán hacer sustituciones de bloques, al ser un esquema N-way. De esta manera, a partir de la primera lectura de un bloque, todo el resto serán hits. Así, si asumimos que el tiempo de acceso a la caché es t y a la memoria principal es $10t$, tenemos que el tiempo total que demora un computador sin memoria caché en este proceso es: $4072 \times 10 \times 10t = 407200t$. Luego, dado que para las 4072 peticiones de memoria se utilizan en total $\lceil \frac{4072}{64} \rceil = 64$ bloques, el tiempo total de acceso para un sistema con caché será: $64 \times (10+1)t + (4072 - 64) \times t + 9 \times 4072t = 41360t$. Luego, el esquema que utiliza caché tomará $\frac{41360}{407200} \approx 0{,}1016$ del tiempo que tomaría sin caché.

c) La siguiente figura presenta el estado de la memoria principal de un computador con memoria virtual en un instante dado: En base a esto, asuma la siguiente situación

- Tamaño de cada página y de cada tabla de páginas es de 4 palabras.
- Existen dos procesos en ejecución, P1 y P2.
- Las páginas no existentes (no asociadas a marcos) se denotan con -.

Tabla de págs. P1

Dirección	Contenido
0	6
1	15
2	3
3	−2
4	−45
5	8
6	28
7	−2
8	9
9	−3
10	8
11	12

Tabla de págs. P2

12	0
13	−12
14	24
15	–
16	0
17	−12
18	−16
19	–
20	0
21	0
22	0
23	0

24	5
25	−3
26	−3
27	2
28	0
29	1
30	2
31	3
⋮	⋮

- En una tabla de páginas, el bit más significativo de cada palabra indica si la página está en memoria (0) o disco (1).
- Ambos procesos solicitan las siguientes direcciones virtuales: 0, 1, 4, 5, 8, 10, 12, 15.

Para cada proceso, transforme las direcciones virtuales en físicas. Si la transformación fue exitosa, indique el dato obtenido. En caso contrario, indique el tipo de page fault generado. **(3 ptos.)**
Solución:

Proc.	Dir. Virt.	Pág.	Offset	Marco	Valor
1	0	0	0	Disco	Page fault por marco en disco
1	1	0	1	Disco	Page fault por marco en disco
1	4	1	0	8	4
1	5	1	1	8	5
1	8	2	0	28	84
1	10	2	2	28	86
1	12	3	0	Disco	Page fault por marco en disco
1	15	3	3	Disco	Page fault por marco en disco
2	0	0	0	0	6
2	1	0	1	0	15
2	4	1	0	Disco	Page fault por marco en disco
2	5	1	1	Disco	Page fault por marco en disco
2	8	2	0	24	68
2	10	2	2	24	70
2	12	3	0	-	Page fault por marco no asignado
2	15	3	3	-	Page fault por marco no asignado

Pregunta 3

Un robot simple, conectado a un computador, es accesible mediante mapeo a memoria. Este robot se mueve en un espacio cuadrado infinito, en el cual cada grilla puede estar vacía o contener una muralla. El robot tiene comandos para ser prendido, apagado, avanzar 1 espacio hacia adelante, girar a la izquierda en 90^o y examinar lo que hay adelante. Cada vez que el robot encuentra desocupado, i.e., ha sido recién iniciado o ha terminado una acción, genera una interrupción para informar que es posible darle un nuevo comando.

a) Describa el mapa de memoria necesario para manejar el robot. **(1 pto.)**

Dirección	Contenido/Función asociada
0	Dirección ISR de manejo del robot.
1	Registro de comandos del robot.
2	Registro de estado del robot.
3-...	Memoria de uso libre

b) Defina el formato de los datos que recibirá el robot como comandos y que entregará este para informar su estado. **(1 pto.)**

Ubicación	Comando/Estado	Valor
Reg. Comandos	Encender	255
Reg. Comandos	Apagar	0
Reg. Comandos	Avanzar	1
Reg. Comandos	Girar Izq.	2
Reg. Comandos	Examinar	4
Reg. Estado	Recién encendido	0
Reg. Estado	Nada que informar	255
Reg. Estado	Espacio libre adelante	1
Reg. Estado	Muralla adelante	2

c) Escriba en assembly x86 la ISR asociada al control del robot, siguiendo el siguiente comportamiento: el robot avanza hasta encontrar una muralla, en cuyo caso girará a la izquierda hasta encontrar un espacio vacío para avanzar, teniendo la precaución de que el robot no retroceda. Asuma que el espacio ha sido diseñado para que el robot no se quede pegado girando eternamente. **(4 ptos.)**

Solución:

```
ISR_robot:
MOV BX, 0x0002
CMP [BX], 0x00
JE inicializar
CMP [BX], 0xFF
JE examinar
CMP [BX], 0x01
JE check_retroceso
JMP girar_izq

inicializar:
MOV BX, 0x0003  ;inicializamos variable en
MOV [BX], 0x00  ;direccion de memoria 3

examinar:
MOV BX, 0x0001
MOV [BX], 0x04
JMP end_isr

check_retroceso:
MOV BX, 0x0003
CMP [BX], 0x02  ;verificamos direccion de retroceso
JE girar_izq

avanzar:
MOV [BX], 0x00
MOV BX, 0x0001
MOV [BX], 0x01
JMP end_isr

girar:
MOV BX, 0x0003
ADD [BX], 0x01
MOV BX, 0x0001
MOV [BX], 0x02

end_isr:
IRET
```

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



IIC2343 – Arquitectura de Computadores (II/2014)

Solución Interrogación 3

Pregunta 1

a) Describa y compare entre ellos los conceptos de i) interrupción, ii) interrupción de software y iii) excepción. **(1 pto.)**
Solución: Una interrupción es un mecanismo que tienen los dispositivos de I/O para solicitar la atención de la CPU, sin necesidad de que ella inicie la acción. La interrupciones son generadas por hardware. Las interrupciones de software son interrupciones generadas por un programa, que llaman a funcionalidades específicas, generalmente del sistema operativo. Es análogo a llamar a una subrutina que no es parte del programa. A diferencia de las interrupciones tradicionales, no hay hardware involucrado. Finalmente, las expcepciones son un tipo especial de interrupción que puede ser originada por software o hardware. La excepción se gatilla cuando ocurre algún evento especial, que está especificado previamente, por ejemplo, una división por cero. En caso de ocurrir, el control de la CPU se pasa a una ISR especial para su manejo, generalmente perteneciente al sistema operativo.

b) El algoritmo de reemplazo **MRU** (Most Recently Used), a diferencia de LRU, descarta primero los elementos que han sido ocupados más recientemente. ¿En que casos podría ser útil el uso de este esquema? **(1 pto.)**
Solución: Este esquema podría ser útil cuando se lee un arreglo de manera secuencial y repetidamente, *i.e.*, una vez que se termina de leer, se comienza nuevamente a leer desde el principio.

c) ¿Cuáles son las ventajas y desventajas de tener un controlador de DMA en un computador? **(1 pto.)**
Solución: La ventaja es la liberación de la CPU de la responsabilidad de copiar datos entre distintas posiciones de memoria. Una desventaja es los problemas de consistencia que deben solucionarse cuando se utiliza una caché tipo write-back.

d) ¿Que características recomendaría para la memoria caché de primer nivel (L1), en un procesador que tiene dos núcleos? Justifique su respuesta. **(1 pto.)**
Solución: Para maximizar el rendimiento, se debería usar una caché distinta para cada núcleo, además de que cada una de estas cachés debiese ser del tipo split.

e) Describa el proceso booting de un computador y las partes de éste que están involucradas. **(1 pto.)**
Solución: Al momento de encender el computador, se carga en la memoria principal el programa de inicio, BIOS o UEFI, que se encuentra en una memoria ROM y se modifica el PC para que apunte al principio del programa. Este programa se encarga de inicializar los dispositivos y de verificar el estado de todos los componentes del computador. Una vez concluido este proceso, el programa de inicio actualiza el PC y carga en memoria principal el sector de booteo del disco duro, que contiene la información de como iniciar el sistema operativo.

f) Describa al menos dos posibles soluciones para el problema de consistencia de memoria que se genera al tener un esquema de escritura de caché write-back **(1 pto.)**
Solución: La primera solución consiste en agregar un bit extra a cada palabra de la memoria RAM, que indique si esta ha sido modificada en la caché. Si es este el caso, el programa que esté accediendo a esta palabras debe ir a buscarla a la caché o actualizar la memoria principal. La segunda solución es más simple y consiste en que todo acceso a memoria se haga a través de la memoria caché, más especificamente, el DMA realiza las copias de memoria viendo la memoria caché y no la RAM.

Pregunta 2

a) ¿Cómo afecta el tamaño de las líneas de la caché al rendimiento? Compare y analice distintos casos para el tamaño y relaciónelo con los principios de localidad. **(1 pto.)**
Solución: Al aumentar el tamaño de las líneas, se da prioridad al principio de localidad espacial, ya que se tendrá una mayor cantidad de elementos cercanos. El problema con esto es que si el tamaño de la caché se mantiene constante, se reduce la cantidad de líneas de esta, lo que tiene un efecto negativo desde el punto de vista de la localidad temporal. Además, puede ser que un tamaño grande de línea no corresponda al uso real que se le da a la memoria, por lo que finalmente puede traer elementos a la caché que nunca serán utilizado.
Si por el contrario, se disminuye el tamaño de las líneas, se necesitarán muchos más accesos a memoria, ya que se priorizará sólo la localidad temporal.

b) El tiempo de acceso promedio a memoria (TAPM) puede ser modelado usando la siguiente expresión:

$$\text{TAPM} = \textit{hit-time} + \textit{miss-rate} \times \textit{miss-penalty}$$

Nombre y explique brevemente, para cada uno de los componentes de la fórmula, una técnica para disminuir el tiempo de acceso promedio a memoria. **(1 pto.)**
Solución: Para disminuir el hit-time se podrían ocupar cachés simple, por ejemplo directly-mapped, de manera que se minimize el tiempo para encontrar la línea. Además, si mantenemos el costo fijo, una caché pequeña tendrá un hit-time menor que una más grande.
Para disminuir el miss-rate, se pueden usar cachés más grandes y con mayores asociatividades, de manera de cumplir de mejor maneras los principios de localidad.
Finalmente, para disminuir el miss-penalty, se podrían usar jerarquías de caché multinivel, de manera que bajar para buscar un dato tenga un menor costo.

c) Considere dos computadores idénticos, que sólo difieren en su organización de la memoria caché. El primer computador utiliza una caché 2-way associative y el segundo una caché 4-way associative. Además, en el primer computador, un ciclo del clock toma 1.25ns y la caché tiene un miss rate de 1.0 %. Por otro lado, un ciclo del clock en la segunda máquina toma 1.4ns. Asumiendo que las instrucciones toman en promedio 2 ciclos del clock cuando la caché funciona de manera perfecta, que estas tienen en promedio 1.5 referencias a memoria cada una, la caché tiene un miss-penalty de 75ns y un hit-time de 1 ciclo del clock, determine el miss-rate del segundo computador, de manera que tenga un TAPM menor que el del primer computador. **(2 ptos.)**
Solución:

$$\begin{aligned} TAPM_1 &= 1{,}25 + (0{,}01 \times 75) = 2{,}0ns \\ TAPM_2 &= 1{,}4 + (x \times 75) \\ TAPM_2 < TAPM_1 &\implies 1{,}4 + 75x < 2{,}0 \\ x &< \frac{2{,}0 - 1{,}4}{75} = 0{,}008 = 0{,}8\,\% \end{aligned}$$

d) En un computador, el tiempo que toma un ciclo del clock depende de la asociatividad N de la memoria caché, donde $N = 1$ es una caché con mapeo directo, $N = 2$ una caché 2-way associative, etc. La expresión que describe la dependencia entre la duración de un ciclo del clock y la asociatividad N esta dada por:

$$\text{Duración_Ciclo_Clock}(N) = 1{,}0 + 0{,}02 \times N^2$$

De manera similar, el miss-rate también depende de N y está dado por la siguiente expresión:

$$\text{miss-rate}(N) = 0{,}01\text{– }0{,}002 \times N$$

Teniendo en cuenta que un hit de la memoria caché toma 1 ciclo del clock y el miss-penalty es de 75ns, construya una expresión para el TAPM en función de la asociatividad N. Luego, encuentre el valor entero para N que minimize el TAPM. **(2 ptos.)**
Solución:

$$\begin{aligned}\text{TAPM}(n) &= 1{,}0 + 0{,}02n^2 + (0{,}01\text{–}0{,}002n) \times 75 \\ &= 1{,}0 + 0{,}02n^2 + 0{,}75\text{–}0{,}15n \\ &= 0{,}02n^2 - 0{,}15n + 1{,}75\end{aligned}$$

Luego, el valor óptimo se obtiene derivando la expresión anterior e igualando a 0.

$$\begin{aligned}\frac{d}{dn}\text{TAPM}(n) &= 0{,}04n - 0{,}15 = 0 \\ n &= 3{,}75\end{aligned}$$

De esta manera, el valor entero de n que minimiza el TAPM es 4.

Pregunta 3

A un computador x86 se le conecta un dispositivo para mostrar imágenes. Este dispositivo se encuentra mapeado a memoria y demora 1 segundo en mostrar imágenes en escala de grises con una resolución arbitraria. Para configurar su funcionamiento, el dispositivo posee un registro de comandos y un registro de argumentos para los comandos, ambos de 16 bits, donde se debe ingresar la resolución de la imagen a mostrar. El dispositivo posee además un buffer de tamaño K, donde se escribe el contenido de una fila de la imagen que luego es dibujada. Para comunicarse con el computador, el dispositivo genera una interrupción cada vez que va a comenzar a dibujar un fila de la imagen, *i.e.*, cada $\frac{1}{M}$ segundos, donde M es la cantidad de filas de la imagen. Teniendo esto en consideración, responda las siguientes preguntas.

a) Describa los comandos que requiere el dispositivo para su funcionamiento y un posible mapeo de memoria de éste. Explique además cómo conectaría el dispositivo al controlador de interrupciones, de manera que la ISR asociada siempre sin retrasos, *i.e.*, que sea llamada cada $\frac{1}{M}$ segundos. **(1 pto.)**
Solución: Para controlar el dispositivo, se necesitan tres comendos básicos. Uno para setear la resolución horizontal, no para la vertical y uno para dar la orden de inicio del proceso. El comando para el inicio del proceso será $0x0001$, para la resolución horizontal $0x0002$ y para la resolución vertical $0x0004$. Para los comandos de seteo de resolución, antes de enviar el comando de control, se debe escribir en el registro de parámetros la resolución, dada por la cantidad de pixeles. El mapeo de memoria queda entonces de la siguiente manera:

Dirección	Función asociada
0-15	Vector de interrupciones de hardware
16-17	Registro de comandos del dispositivo
18-19	Registro de argumentos del dispositivo
20-(K+20)	Buffer del dispositivo

Finalmente, asegurar que la interrupción no tenga restrasos, es necesario que esta no sea enmascarable y que tenga la prioridad más alta posible para que no sea encolada, lo que implica que debe conectarse en la señal IRQ0.

b) Asumiendo que todas las instrucciones del computador toman un ciclo del clock y que el procesamiento de una petición de interrupción es instantáneo, *i.e.*, el tiempo gastado en todo el proceso de interrupción es sólo el tiempo que toma la ISR, y que los datos son transferidos mediante un esquema PIO, estime la resolución horizontal máxima que puede mostrar el dispositivo, dada una resolución vertical M, si el clock del computador corre a X Hz. **(2 ptos.)**
Solución: Dado que el dispositivo interrumpe cada $\frac{1}{M}$ segundos para dibujar una línea, este es el tiempo máximo que puede tomar la ISR. Luego, si el clock corre a X Hz y cada instrucción toma un ciclo del clock, o sea, $\frac{1}{X}$ segundos, la cantidad de instrucciones que pueden ejecutarse en una ISR es de $\frac{1/M}{1/X} = \frac{X}{M}$. A continuación es necesario definir la cantidad de instrucciones necesarias para dibujar un pixel. Existen multiples opciones, por lo que acá se describirá una básica basada en un ciclo *do while* y direccionamiento con registro base y registro índice. Dado que se utiliza el esquema PIO, se necesitan al menos 2 instrucciones para copiar elementos entre 2 posiciones de memoria. Si asumimos que las imágenes utilizan pixeles de 1 byte, se necesitan 2 instrucciones para dibujar 2 pixeles, si usamos los registros de 16 bits. A esto hay que agregar las instrucciones necesarias para incrementar dos contadores, la instrucción de comparación y la instrucción de salto. Luego, se necesitan $\frac{6}{2} = 3$ instrucciones por pixel. De esta manera, la resolución horizontal máxima es de $\lfloor \frac{X}{3M} \rfloor$ pixeles. Si evaluamos la fórmula con valores típicos para una imagen, como por ejemplo 640x480 pixeles, se obtiene que la frecuencia del clock debe ser de al menos 921.6KHz

c) Escriba un programa que configure el dispositivo para mostrar una imagen que tiene un tamaño válido y una ISR que escriba en la memoria del dispositivo los valores de los pixeles. Asuma que el inicio del contenido de la imagen se encuentra en la variable *imagen*. **(2 ptos.)**
Solución: Asumimos que la frencuencia del clock es de 768KHz.

```
        JMP   main
        reg_comandos dw 0x0010
        reg_params dw 0x0012
        res_hor dw 0x0280 ;640px
        res_ver dw 0x01E0 ;480px

    main:
        MOV BX, reg_params
        MOV [BX], res_hor
        MOV BX, reg_comandos
        MOV [BX], 0x0002

        MOV BX, reg_params
        MOV [BX], res_ver
        MOV BX, reg_comandos
        MOV [BX], 0x0004

        LEA BX, imagen
        MOV DI, 0

        MOV AX, reg_comandos
        MOV [AX], 0x0000

    wait:
        CMP DI, res_ver
        JNE wait

        RET

    ISR0:
        MOV SI, 0
        INC DI
    start:
        MOV AX, [BX]
        INC BX
        MOV [SI+20], AX
        INC SI
        CMP SI, res_hor
        JLT start
        IRET
```

d) ¿Es posible aumentar notablemente la resolución de las imágenes a mostrar, si se utilizara DMA en vez de PIO como esquema de escritura, asumiendo que el controlador de DMA es capaz de copiar una palabra por ciclo del clock? Argumente su respuesta. **(1 pto.)**
Solución: En el caso de la solución acá propuesta, el DMA ofrecería un gran aumento en la resolución, ya que el costo de pintar cada pixel sería sólo 1 ciclo, a diferencia de las 3 instrucciones necesarias por pixel en caso de usar PIO.

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



IIC2343 – Arquitectura de Computadores (I/2015)

Solución Interrogación 3

Pregunta 1

a) ¿Cómo es el rendimiento de una memoria caché, si el patrón de accesos a memoria distribuye de manera uniforme sobre todas las posibles direcciones? Ejemplifique el o los posibles casos. **(1 pto.)**
Solución: El rendimiento dependerá del patrón temporal de accesos. Si los accesos se realizan linealmente, el *hit-rate* será muy alto, ya que sólo existirán *misses* al comenzar a leer un bloque. Por otro lado, si los accesos se realizan de manera que se elija una palabra de un bloque que no está en la caché, el *hit-rate* será cercano a cero.

b) El *orden de booteo* de un computador es una preferencia que se modifica generalmente en el BIOS. Sin embargo, estas propiedades pueden modificarse, lo que implica que no se guardan en la ROM donde está el BIOS, y se mantienen incluso cuando el computador se apaga. ¿Cómo y donde se pueden almacenar estos datos? **(1 pto.)**
Solución: En general, para guardar estas preferencias, se utilizan una pequeña cantidad de memoria volátil, que es alimentada por una pila de bajo consumo, que se encuentra en la tarjeta madre. Otra posible solución es almacenarlas en un sector predeterminado de un dispositivo de almacenamiento no volátil, como un disco duro.

c) Un computador de 64 bits tiene una memoria caché de 32KB, con 1024 líneas de 32 palabras. ¿Cuanto espacio de esta caché es usado por información distinta de los datos? **(1 pto.)**
Solución: La solución de este ejercicio depende de que función de correspondencia se ocupe. Si usamos *4-way associative*, necesitamos 5 bits para el offset y 8 bits para el conjunto. Luego, se requieren 49 bits para el tag. Si asumimos que hay un tag y un bit de validez por línea, necesitamos en total $50 \times 1024 bits = 50Kb = 6,25KB$, lo que equivale a casi un 20 % de la capacidad de la caché para almacenar datos.

d) ¿Qué es y cuál es la función del *Address Decoder*? **(1 pto.)**
Solución: El *Address Decoder* es una pieza de *hardware* que permite el mapeo a memoria de dispositivos de I/O. Transforma las direcciones mapeadas, a las direcciones de los dispositivos.

e) Describa una política de reemplazo óptima para memorias caché. **(1 pto.)**
Solución: El criterio óptimo para sustituir un bloque, es elegir aquel que no vuelva a ser requerido en la mayor cantidad de tiempo. Esta política es llamada el algoritmo de Bélády. Cualquier solución que implique la selección de un bloque de esas características, es correcta.

f) Un disco duro SSD puede alcanzar velocidades de transferencia de hasta 600MB/s. Donde debería conectarse idealmente un disco de este tipo en un computador x86? **(1 pto.)**
Solución: Debería conectarse al bus PCI Express, idealmente a una entrada mayor o igual a 3x.

Pregunta 2

Un computador compatible con el assembly x86, tiene conectado un controlador de DMA mediante los siguientes puertos:

Puerto	Función
33	Comandos
34-35	Origen
36-37	Destino
38-39	Cont. palabras

Este controlador inicia la copia cuando recibe su registro de comandos la palabra 0x01 y puede utilizarse sólo mediante la interrupción de software 22, que está mapeada a la cuarta palabra del vector de interrupciones.

1. Asumiendo que el vector de interrupciones comienza en la dirección 0x1000, escriba un programa que registre en el sistema una subrutina con *label* ISR_DMA, como la ISR de la interrupción de software para llamar al controlador de DMA. Antes de realizar este proceso, respalde la ISR anterior. **(1 pto.)**
   Solución:

```
MOV AX, [0x1003]
PUSH AX
MOV [0x1003], ISR_DMA
```

2. Escriba el código de la subrutina ISR_DMA. Defina claramente la convención de llamada que usará la ISR. **(2 ptos.)**
   Solución: Asumimos que la convención de llamada para esta interrupción de software, consiste en almacenar en AX la dirección de origen, en BX la dirección de destino y en CX la cantidad de palabras.

```
OUT 34, AL
OUT 35, AH
OUT 36, BL
OUT 37, BH
OUT 38, CL
OUT 39, CH
OUT 33, 0x01
```

3. Asuma ahora que al computador se conecta una unidad de almacenamiento externo de sólo lectura, a la cual se accede mediante su buffer interno, *i.e.*, el dispositivo copia en su buffer de manera instantánea, la cantidad de palabras indicada en su registro de parámetros, desde su dirección interna indicada en su registro de direcciones, cuando se ingresa el comando 0x10 en su registro de comandos. Teniendo esto en consideración, el computador expone el siguiente mapa de memoria:

Dirección	**Función asociada**
0-6	Exception handlers
7	Registro de comandos de dispositivo de almacenamiento externo
8	Registro de direcciones de dispositivo de almacenamiento externo
9	Registro de parámetros de dispositivo de almacenamiento externo
10-2057	Buffer del dispositivo de almacenamiento externo
2058-5119	Espacio reservado del sistema
5120-65535	Espacio de memoria de libre disposición

Asumiendo que puede realizar llamados del tipo CALL Reg, donde Reg es un registro que almacena una dirección de memoria, escriba un programa que ejecute un archivo de 10KB, ubicado en la dirección 0x7A12 del dispositivo de almacenamiento. **(3 ptos.)**

Solución: En estricto rigor, el enunciado tiene un error al asignar sólo una palabra del mapeo para el registro de direcciones y una para el registro de parámetros. Como solución, se asume que es posible asignar palabras de 16 bits a esas direcciones.

```
    MOV DX, 0x0800 ; 0x0800 = 2048
    MOV [0x0008], 0x7A12
    MOV [0x0009], DX
    MOV AX, 0x000A
    MOV BX, 0x1400 ; 0x1400 = 5120
    MOV CX, DX
    MOV SI, 0
  start:
    CMP SI, 5
    JE end
    MOV [0x0007], 0x10
    INT 22
    ADD [0x0008], DX
    ADD BX, DX
    ADD SI, 1
    JMP start
  end:
    MOV AX, 0x1400
    CALL AX
```

Pregunta 3

a) Una memoria caché es inclusiva, cuando todos los bloque contenidos en un nivel, también están contenidos en los inferiores. Por otro lado, una caché es exclusiva, cuando un bloque no se repite en ninguno de sus niveles.

I. En una CPU con 2 niveles de caché, indique cuándo es conveniente tener una caché inclusiva y cuando una exclusiva. **(1 pto.)**
Solución: Si el tamaño de los dos niveles de caché es similar, conviene tener una caché exclusiva, para no tener bloques duplicados y desaprovechar espacio. Por otro lado, si el segundo nivel es bastante más grande que el primer nivel, conviene tener una caché inclusiva.

II. Asuma ahora que se tiene un sistema con 2 procesadores, cada uno con una caché independiente. Una de los procesadores quiere saber si el otro procesador tiene un dato en la caché ¿Cuál de estos dos tipos de caché permite que esta búsqueda sea más rápida? **(1 pto.)**
Solución: La caché inclusiva permite una búsqueda más rápida, ya que sólo es necesario buscar en el segundo nivel, debido a que si no está ahí, tampoco puede estar en el primer nivel. Por otro lado, si la caché es exclusiva, es necesario buscar en ambos niveles.

b) Considere un computador con microarquitectura Von Neumann, donde la tasa de ciclos de clock por instrucción es igual a N, cuando todos los accesos a memoria producen hits en la caché. La memoria caché tiene miss-rate de 4 % y miss-penalty de 25*N ciclos de clock. Si en un programa de K instrucciones, el 50 % de éstas realizan lectura de un dato en memoria, ¿cuántos ciclos de clock menos tomaría la ejecución del programa, si todas las instrucciones produjeran hits en la memoria caché? **(2 ptos.)**
Solución: Como el programa tiene K instrucciones, sabemos que cuando el hit-rate es 100 %, el programa demorará K*N ciclos de clock. Dado que en la arquitectura Von Neumann todas las instrucciones requieren al menos un acceso a memoria (fetch), si nos ponemos en el caso donde el miss-rate es 4 %, se tiene que 0,04*K instrucciones generarán un miss en la memoria caché en la etapa fetch. Además de esto, se sabe que el 50 % de las instrucciones son de escritura o lectura, por lo tanto, otras 0,5*0,04*K instrucciones generarán un miss cuando quieran leer datos. Sumando, tenemos 0,06*K instrucciones que generan miss, por lo que el tiempo total será de K*N+0,06*25*K*N =2,5*K*N ciclos de clock. De esta manera, la ejecución cuando el hit-rate es 100 % tomará 1,5*K*N ciclos de clock menos.

c) Considere el siguiente código que almacena en el vector c el producto entre una matriz A y un vector b:

```
for (i=0; i<96; i++ )
{
  for (j=0; j<96; j++ )
  {
    c[i] = c[i]+ A[i][j] * b[j];
  }
}
```

Asuma que la matriz A está almacenada en orden de filas y que utiliza una memoria caché de 4KB con mapeo directo para ella sola, *i.e.*, b y c usan una memoria caché distinta a esta. Asumiendo que A[0][0] es mapeado a la primer línea de la caché, y que el tamaño de un bloque es de 64 bytes, ¿Cuáles son los valores para **i** y **j**, para los cuales A[i][j] sustituirá a A[0][0] en la primera línea de la caché? **(2 ptos.)**
Solución: La respuesta para esta pregunta depende de el tipo de dato que se asuma para la matriz A. En este caso, si asumimos que A es del tipo double, *i.e.* 64 bits por elemento, en cada línea de la caché podrán almacenarse 8 elementos de la matriz A, por ejemplo, A[0][0], A[0][1] ... A[0][7] estarán en la misma línea. Esto significa que una fila de A utiliza 96/8 = 12 líneas de caché. Como la caché para A tiene 4096 / 64 = 64 líneas, 5 filas completas más 4*8 elementos de la fila siguiente de A la llenarán completamente. Luego, el elemento que sustituye a A[0][0] es A[5][32].

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



IIC2343 – Arquitectura de Computadores (II/2015)

Interrogación 3

Pregunta 1

a) ¿Que mecanismo de hardware se utiliza para informar a un proceso que la copia de memoria que solicitó ya fue terminada? **(1 pto.)**
Solución: Se utiliza una solicitud de interrupción generada por el controlador de DMA.

b) Cuando se atiende una solicitud de interrupción (IRQ), ¿en que momento se deben respaldar los flags de la CPU y por qué? **(1 pto.)**
Solución: Se deben respaldar inmediatamente después de terminar de ejecutar la instrucción actual. Si se hace después, se perderán los flags debido a la ejecución de otras instrucciones.

c) Un computador compatible con el assembly x86 de 16 bits, tiene conectado un controlador de DMA mediante los siguientes puertos:

Puerto	Función
33	Comandos
34-35	Origen
36-37	Destino
38-39	Cont. palabras

Este controlador inicia la copia cuando recibe su registro de comandos la palabra 0x01 y puede utilizarse sólo mediante la interrupción de software 22. Además, este computador cuenta con un timer programable, que es capaz de informar a la CPU cuando hayan transcurrido X ms, mediante una IRQ. El timer se activa enviado la palabra 0x01 al puerto 53, mientras que el tiempo para la interrupción se programa enviando un número natural de 8 bits al puerto 54. Finalmente, ISR_TIMER es la subrutina que maneja la interrupción.

i. Escriba el código de ISR_DMA, que corresponde a la subrutina que se ejecuta cuando se llama a la interrupción de software 22. Defina claramente la convención de llamada que usará la ISR. **(1 pto.)**
Solución: Asumimos que la convención de llamada para esta interrupción de software, consiste en almacenar en AX la dirección de origen, en BX la dirección de destino y en CX la cantidad de palabras.

```
OUT 34, AL
OUT 35, AH
OUT 36, BL
OUT 37, BH
OUT 38, CL
OUT 39, CH
OUT 33, 0x01
```

II. Asuma que el computador ejecuta un sistema operativo con soporte para multiprogramación que implementa *preemptive scheduling*. Teniendo esto en cuenta, escriba el código de ISR_TIMER, que se encargará de respaldar el PCB del proceso saliente, cargar el del proceso entrante y de programar el timer para que genere una nueva interrupción, donde cada PCB tiene un tamaño de 1KB. El PCB del proceso en ejecución se almacena a partir de la posición de memoria 0x0100. Asuma que los parámetros para realizar el cambio de contexto (dir. origen PCB entrante, dir. destino PCB saliente, tiempo para siguiente interrupción), se encuentran disponibles, en el mismo orden, a partir de la dirección de memoria 0x0000. **(3 ptos.)**

Solución:

```
ISR_TIMER:
  MOV AX, 0x0100
  MOV BX, 0x0002
  MOV CX, 0x0400 ; 0x0400 = 1024
  INT 22

  MOV AX, 0x0000
  MOV BX, 0x0100
  INT 22

  MOV AL, [0x0004]
  OUT 54, AL
  OUT 53, 0x01
  IRET
```

Asignación de puntaje: 1 pto. por cada copia de PCB, 0.5 ptos. por programar el timer, 0.5 ptos por terminar correctamente la ISR (IRET).

Pregunta 2

a) Un ISP (Internet Service Provider) tiene un ping de 7 ms y respalda diariamente el 30 % de los recursos requeridos por los usuarios, para acelerar el acceso. En caso de no tener respaldado un recurso, toma X segundos en obtenerlo. ¿Cuál es el tiempo de acceso promedio a un recurso cualquiera en la red, asumiendo que ese recurso también fue requerido el día anterior? **(1 pto.)**
Solución: $TAP = 7 + 0{,}7 \times 1000X = 707\ ms$.

b) ¿Por qué es importante que algunos niveles de memoria caché sean exclusivos para cada núcleo y otros compartidos? **(1 pto.)**
Solución: Es muy posible que cada núcleo ejecute programas distintos, por lo tanto es necesario que los datos e instrucciones que usan se mantengan almacenadas de manera independiente a las de otro núcleo. Así, estas no tienen conflicto por el espacio en la memoria caché. Por otro lado, los programas necesitan generalmente compartir datos, como por ejemplo los parámetros para llamadas a subrutinas. Así, para evitar usar la memoria principal (más lenta), para compartir estos datos, se utiliza una memoria caché ubicada en la parte más bajo de la jerarquía.

c) Las aplicaciones que reproducen audio o video son parte de un tipo de aplicaciones que generan un trabajo en la memoria del tipo *streaming*, que consiste en la lectura de grandes cantidades de datos, pero una muy baja reutilización de estos. Para este ejercicio considere un computador que corre una aplicación que realiza *streaming* para acceder a 512KB de datos. El patrón de acceso a memoria es el siguiente: 0, 4, 8, 12, 16, ...

I. Asuma que el computador tiene una memoria caché de 64 KB con mapeo directo y líneas de 32 bytes. ¿Cuál es el *miss-rate* para el patrón de accesos a memoria descrito anteriormente? ¿Cuán sensible es este *miss-rate* con respecto al tamaño de la memoria caché? ¿Mejora el *hit-rate* si se utiliza una caché 4-way associative? **(1 pto.)**
Solución: Dado que 32 es un múltiplo de 4, se realizan 8 lecturas por bloque/línea. Dado que el patrón de accesos es estrictamente ascendente, el miss rate en un bloque es igual al de toda la caché, sin importar el tamaño de esta. Por lo tanto, el miss-rate es $\frac{1}{\frac{32}{4}} = 12{,}5\,\%$, independiente si la caché usa mapeo directo o 4-way associative.

II. Recalcule el *miss-rate* tomando los siguientes tamaños de línea en una caché con mapeo directo: 16 bytes, 64 bytes, 128 bytes. ¿Qué tipo de localidad aprovecha este tipo de aplicaciones? **(1 pto.)**
Solución: $\frac{1}{\frac{16}{4}} = 25\,\%$, $\frac{1}{\frac{64}{4}} = 6{,}25\,\%$, $\frac{1}{\frac{128}{4}} = 3{,}125\,\%$. Dados estos resultados, queda claro que las aplicaciones de streaming aprovechan la localidad espacial.

III. El *prefetching* es una técnica que saca provecho de patrones de lectura predecibles, para traer desde la memoria de manera especulativa, bloques de memoria adicionales cuando una línea de la caché en particular es accedida. Un ejemplo de esto es el *stream buffering*, que consiste en copiar, en un *buffer* separado, bloques adyacentes a la línea de la caché accedida. Si el dato se encuentra en este *buffer*, se considera como *hit* y es copiado a la caché, mientras que el espacio que este usaba en el *buffer*, es llenado con un nuevo bloque adyacente. Asuma que el computador cuenta con un *stream buffer* de dos entradas y que la latencia de la cache es tal que una línea puede ser cargada antes de que el cálculo realizado sobre la línea anterior ha sido completado. En base a esto, cuál es el *hit-rate* para el patrón de acceso a memoria descrito anteriormente y una caché con líneas de 32 bytes? **(2 ptos.)**
Solución: El hit-rate será casi igual a 1, ya que sólo hay un miss en el primer acceso de memoria. Luego, todas las lecturas encontrarán el dato que necesitan ya sea en la memoria caché o en el stream buffer. Más especificamente, el hit-rate será $\frac{\frac{512\times1024}{4}-1}{\frac{512\times1024}{4}} = 0{,}99999$.

Pregunta 3

a) ¿Cuál es la cantidad mínima de memoria física que debe tener un computador de 32 bits con un esquema de paginación simple como el visto en clases? **(1 pto.)**
Solución: Se necesita como mínimo espacio para una tabla de páginas y para una página.

b) Explique cómo un proceso puede sufrir de inanición si se utiliza el esquema *Fixed priority pre-emptive scheduling*. ¿Es posible que ocurra esto en el esquema *round-robin*? **(1 pto.)**
Solución: Un proceso puede sufrir inanición en el esquema *Fixed priority pre-emptive scheduling* si siempre existen procesos con mayor prioridad. Esto no puede pasar en round-robin, debido a que este esquema despacha los procesos usando FIFO y un tiempo de ejecución fijo.

c) Un computador de 64 bits posee soporte de hardware para paginación, donde cada página es de 4 KB.

I. Describa el proceso que realiza el hardware para obtener el dato buscado, a partir de una dirección virtual. Indique claramente el propósito de cada uno de los bits de la dirección virtual. **(1 pto.)**
Solución: Dado que cada página es de 4KB, se necesitan 12 bits para indicar la palabra dentro de una página. De esta manera, los 52 bits más significativos de la dirección virtual son usados por la MMU para ubicar el marco correspondiente en la tabla de páginas indicada por el registro PRBR. Una vez obtenido el marco, se realiza la petición de memoria usando la dirección fisica, que se construye sustituyendo los 52 bits más significativos de la dirección virtual (número de página), por el número del marco obtenido de la tabla.

II. Si no se ponen restricciones a la cantidad de memoria que puede usar un proceso, ¿cuál es el tamaño de una tabla de páginas (sin contar los bits de validez y disco)? **(1 pto.)**
Solución: Dado que la memoria virtual da acceso a toda la memoria direccionable y que se utiliza un computador de 64 bits, una tabla de páginas tendrá $\frac{2^{64}}{2^{12}} = 2^{52}$ entradas.

III. Proponga un esquema para paginación, que permita que la cantidad de entradas de la estructura que almacena las relaciones marco-página esté acotada por la cantidad de marcos físicos existentes. **(2 ptos.)**
Solución: La solución es utilizar una tabla invertida, que tiene una cantidad de entradas fija e igual a la cantidad de marcos. En cada una de las entradas se indica a que página está asocido el marco. Luego, para obtener un dato desde la memoria, se debe buscar en toda esta tabla la ocurrencia de la página virtual requerida. A pesar de que esto es muy lento, es posible acelerarlo utilizando una tabla de hash.

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



IIC2343 – Arquitectura de Computadores (I/2016)

Solución Interrogación 3

Respuestas sin desarrollo o justificación no tendrán puntaje.

Pregunta 1

a) Un computador tiene una memoria caché de 16KB, con líneas de 32 bytes que almacenan 8 palabras, y un tiempo de acceso de 10ns. La memoria caché está conectada a la memoria principal mediante un bus capaz de transferir 8 bytes en 120ns. ¿Cuál es el hit-rate que debe tener la memoria caché para tener un tiempo de acceso promedio de 20ns? **(1 pto.)**
Solución: $20ns = 10ns + (1 - HR) \times 480ns \Rightarrow HR \approx 0{,}98$

b) Comente sobre las ventajas de tener una memoria caché *split* en vez de una *unified* conectada a la memoria de datos del computador básico. **(1 pto.)**
Solución: No existen ventajas, sólo desventajas, ya que una memoria caché split conectada a una memoria de datos desperdiciará la mitad de su capacidad en instrucciones.

c) Responda las siguientes preguntas considerando el siguiente código, que implementa la multiplicación entre una matriz **A** y un vector **b**:

```
for (i=0; i<96; i++ )
{
    for (j=0; j<96; j++ )
    {
        c[i] = c[i]+ A[i][j] * b[j];
    }
}
```

i. Determine la cantidad de *misses* y el *hit-rate* de la memoria caché, asumiendo que todos los elementos son números del tipo double y que la matriz **A** está almacenada en orden de filas. La memoria caché es *fully associative* y tiene capacidad infinita, con líneas de 64 bytes. Considere además que el vector **c** ha sido inicializado con ceros previamente y que no existen elementos de **A**, **b** y **c** almacenados en la caché. **(2 ptos.)**
Solución: Para el vector **c**, dado que las líneas son de 84 bytes, cada lectura cargará 8 elementos, por lo que se tendrán 12 *misses*. Usando este mismo argumento, se tendrán también 12 *misses* para el vector **b** y $96 \times 12 = 1152$ para la matriz **A**, lo que da un total de 1176 *misses*. Finalmente, dado que por iteración se realizan 4 accesos a memoria, se tienen en total $96 \times 96 \times 4 = 36864$ accesos, lo que implica que el *miss-rate* es $\frac{1176}{36864} = 0.0319 = 3.19\,\%$.

ii. Asuma ahora que la matriz **A** utiliza una memoria caché de 4KB con mapeo directo para ella sola, *i.e.*, **b** y **c** usan una memoria caché distinta a esta. Asumiendo que A[0][0] es mapeado a la primera

línea de la caché y que el tamaño de un bloque es de 64 bytes, indique los valores de **i** y **j**, para los cuales el bloque usado por A[95][95] sustituirá al bloque usado por A[i][j] en la caché, y la línea de la caché donde ocurrirá esto. **(2 ptos.)**
Solución: Dado que las líneas son de 64 bytes, cada fila de la matriz **A** utilizará $\frac{96\times 8}{64} = 12$ líneas. Además, como la memoria caché es de 4KB, esta tendrá $\frac{4096}{64} = 64$ líneas. Luego, la matriz completa utilizará exactamente $\frac{96*12}{64} = 18$ veces la capacidad completa de la memoria caché, lo que implica que el bloque que contiene a A[95][95] será asignado a la línea 63. Finalmente, la distancia entre el bloque que contiene A[95][95] y el que contiene al A[i][j] que será sustituido es de 12 líneas, lo que equivale a $5,\overline{3}$ filas completas de la matriz. Esto implica que el elemento que será sustituido por el bloque de A[95][95] en la caché será A[90][32].

Pregunta 2

a) En un computador con soporte para memoria virtual, ¿cómo se pueden implementar regiones de memoria protegidas y regiones de memoria compartidas? **(1 pto.)**
Solución: Las regiones de memoria compartida se pueden implementar asignando el mismo marco de memoria física a páginas de procesos distintos. Las regiones de memoria compartida se pueden implementar de manera trivial prohibiendo la asignación de los marcos que las contienen a cualquier página virtual.

b) Indique que eventos generan un cambio de contexto en el esquema de *cooperative scheduling*. **(1 pto.)**
Solución: El cambio de contexto puede darse luego de una cesión explícita por parte del proceso (*yield*), o tras a una petición de interacción con un dispositivo de I/O.

c) Considere un computador con un espacio direccionable virtual de 32 bits, espacio de direccionamiento físico de 30 bits y páginas de 8KB.

   I. Describa la composición interna de una dirección virtual. **(1 pto.)**
   Solución: Una dirección virtual se dividirá utilizando los 14 bits menos significativos para indicar el offset dentro de una página y los restantes 18 bits para obtener el mapeo desde página a marco.

   II. ¿Cuál es el máximo número de entradas válidas que pueden existir en una tabla de páginas? **(1 pto.)**
   Solución: Asumiendo que existe un único proceso, se necesita por lo menos un marco que almacene la tabla de páginas de este. Por lo tanto, la cantidad máxima de entradas válidas será $\frac{2^{30}}{2^{14}} - 1 = 2^{16} - 1 = 65535$.

d) En un computador con soporte para memoria virtual, ¿el registro PC almacena direcciones virtuales o físicas? Justifique su respuesta considerando el efecto de los cambios de contexto y del *swapping*. **(2 ptos.)**
Solución: Dado que un proceso sólo genera peticiones de memoria utilizando direcciones virtuales, es conveniente que el PC también almacene direcciones de este tipo. En caso contrario, existen numerosos problemas de consistencia de memoria que se pueden dar. Tomando como consideración los cambios de contexto y *swapping*, es muy posible que un proceso que vuelve a ejecutarse despues de haber estado en espera por una petición de I/O, no tengo su código en la misma ubicación física que antes, debido al swapping de memoria. Si el PC almacena la dirección física, este apuntará ahora a una dirección que no contiene necesariamente el código correcto. Por el contrario, si almacena direcciones virtuales, la transformación de estas a direcciones físicas siempre llevaran a la ubicación correcta del código, independiente si este cambió de ubicación física.

Pregunta 3

a) Estime el tiempo de ejecución de un programa que toma **N** instrucciones en el computador básico con *pipeline*, en base a las siguientes condiciones:

- El 50 % de las instrucciones realizan lecturas o escrituras en memoria.
- El 10 % de las instrucciones son de salto y en el 25 % de estas, el salto finalmente se realiza.
- En el 50 % de las ocasiones, el dato obtenido desde la memoria debe ser utilizado en la instrucción siguiente.

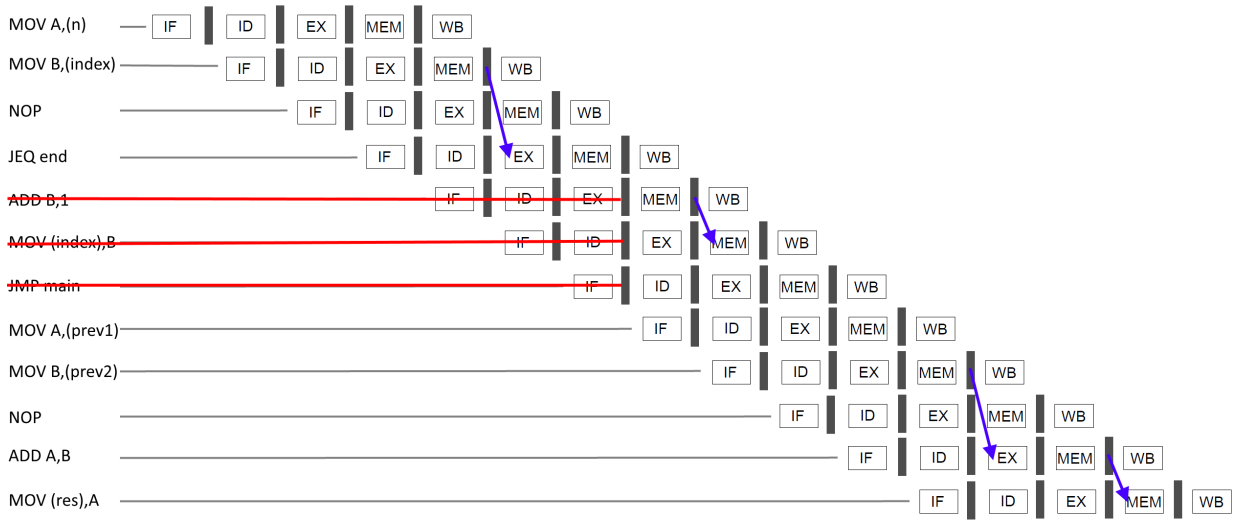
Cualquier supuesto sobre la solución debe quedar claramente explicado. **(1 pto.)**
Solución: Si se obvían las restricciones y sólo se considera la cantidad de instrucciones ejecutadas por el programa, el tiempo de ejecución será de N+4 ciclos. Si se consideran en conjunto la primera y tercera condiciones, por cada una de las instrucciones afectadas, se deberá agregar una burbuja que retrasará la ejecución en un ciclo, lo que en la práctica para la totalidad del programa significa un retraso de $0{,}5 \times 0{,}5 \times N = 0{,}25N$ ciclos. Adicionalmente, si se considera la segunda restricción, se agregarán 3 ciclos por cada salto realizado, retrasando el programa $0{,}1 \times 0{,}25 \times 3 \times N = 0{,}075N$ ciclos. Tomando todo esto en consideración, el tiempo total de ejecución del programa será de $N + 0{,}25N + 0{,}075N + 4 = 1{,}325N + 4$.

b) Determine el número de ciclos que se demora el siguiente código, detallando en un diagrama los estados del pipeline por instrucción. El pipeline tiene forwarding entre todas sus etapas, el manejo de stalling es por software (instrucción NOP) y predicción de salto asumiendo que no ocurre. Indique en el diagrama cuando ocurre forwarding, stalling y flushing. **(2 ptos.)**

```
DATA
    n     1
    index 1
    prev1 0
    prev2 1
    res   0

CODE
    main:
        MOV A, (n)
        MOV B, (index)
        JEQ end
        ADD B, 1
        MOV (index), B
        JMP main
    end:
        MOV A, (prev1)
        MOV B, (prev2)
        ADD A, B
        MOV (res), A
```

Solución: En base al diagrama presentado a continuación, el programa toma 16 ciclos en ejecutarse.



c) Diseñe un computador con *pipeline* de al menos 5 etapas, donde debido a restricciones del hardware, la etapa MEM debe ejecutarse antes que la etapa EX. El computador debe soportar las mismas funcionalidades que el computador básico con *pipeline*. **(3 ptos.)**

Solución: La clave para solucionar este ejercicio consiste en evitar que los saltos mal predecidos afecten el contenido de la memoria, que complica la aplicación de una operación de flushing. En el computador básico con pipeline, esto se evita ubicando la etapa MEM después de EX, dado que la condición de salto se genera en la etapa EX y se evalua en la etapa MEM. Teniendo esto en consideración, una posible solución consiste en un computador con un pipeline de 6 etapas: IF, ID, JMP, MEM, EX, WB, donde la etapa JMP contiene un restador que genera la condición de salto. Luego, tal como en el computador básico con pipeline, en la etapa MEM se evalua la condición, evitando la escritura en memoria de las instrucciones siguientes.

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



IIC2343 – Arquitectura de Computadores (II/2016)

Solución Interrogación 3

Pregunta 1

a) ¿Cuál es la función del bit de validez en una tabla de páginas? **(1 pto.)**
Solución: El bit de validez en la tabla de páginas es utilizado para determinar si una página está mapeada correctamente a un marco físico. Cuando el bit de validez tiene valor 1 entonces esa entrada en la tabla de páginas es válida, mientras que si su valor es 0, no es válida.

b) Indique bajo que condiciones el uso de memoria virtual podría hacer más lenta la ejecución de los procesos en un computador. **(1 pto.)**
Solución: El uso de memoria virtual puede hacer más lenta la ejecución de procesos cuando ocurren page faults en los siguientes casos:

- No hay marcos físicos disponibles para alguna página de un proceso, se guarda alguna otra página de algún proceso en el disco duro para liberar un marco físico (swap out).
- Se quiere ocupar alguna dirección de memoria cuya página está guardada en disco. Se debe hacer swap out de alguna página para luego hacer swap in de la página que tiene la dirección de memoria solicitada.

c) Analice y comente desde los conceptos de justicia e inanición, el algoritmo *fixed priority pre-emptive scheduling*. **(1 pto.)**
Solución: El algoritmo fixed priority pre-emptive scheduling le otorga a cada proceso una prioridad. Como este algoritmo es pre-emptive, cada proceso es interrumpido para pasar el control al sistema operativo y así realizar un cambio de contexto a otro proceso. Cuando se acaba el tiempo de un proceso o llega otro proceso con mayor prioridad, el proceso actual es interrumpido. Al utilizar la prioridad para ordenar la cola de procesos, se intenta asegurar la justicia sobre el uso de recursos para los procesos en el sistema. Por otro lado, puede ocurrir que los procesos con las prioridades más bajas nunca se lleguen a ejecutar, lo que se conoce como inanición.

d) ¿Por qué no es enmascarable la IRQ0 en un computador x86? **(1 pto.)**
Solución: La IRQ0 corresponde al timer del sistema, que permite realizar cambios de contexto entre procesos (pre-emptive scheduling), por lo que no está permitido enmascarar esta IRQ.

e) ¿Cómo se relaciona el tamaño de las páginas con la efectividad de la TLB? **(1 pto.)**
Solución: Mientras mayor sea el tamaño de la página, mayor porcentaje de los marcos totales del proceso podrá ser almacenado en la TLB, ya que el proceso tendrá menos páginas en total. Por otro lado, el tamaño de cada entrada de la TLB no es afectado por el cambio en el tamaño de la página (la TLB solo guarda marcos asociados a páginas de un proceso). Finalmente, aumentará el hit-rate de la TLB.

f) Durante la mayoría de sus tiempo de ejecución, un proceso tiene el 51 % de sus páginas en el *swap file*. Otro proceso que utiliza la misma cantidad de memoria total, tiene el 49 % de sus páginas en el *swap file*, también durante la mayor parte de su tiempo de ejecución. ¿Asumiendo que ambos se ejecutan durante la misma cantidad de tiempo, cuál de los dos procesos generó más *page faults*? **(1 pto.)**
Solución: Dada la información entregada por el enunciado, no se puede saber cual de los 2 procesos generó más page faults, ya que no se conoce las direcciones de memoria a las que accedió cada proceso. Por ejemplo, puede que el proceso con más páginas en el swap file solo haya utilizado direcciones de memoria que están en la memoria principal, o bien que solo haya utilizado direcciones de memoria cuyas páginas están en el swap file. Se puede apreciar que la respuesta depende exclusivamente de las direcciones de memoria utilizadas por cada proceso.

Pregunta 2

a) Un computador tiene una memoria caché 2-way de 16 líneas de 2 bytes cada una, con política de sustitución LRU. Indique para cada acceso de la siguiente lista, si corresponde a un *hit* o un *miss*: 2, 3, 11, 16, 21, 10, 80, 48, 39, 11, 3, 80. Justifique su respuesta diagramando el estado de la memoria caché al finalizar la secuencia. **(2 ptos.)**
Solución: La secuencia tiene los siguiente hits/misses: M, H, M, M, M, H, M, M, M, H, H, H. El estado de la caché al finalizar la secuencia es el siguiente:

Conj.	Lín. A	Lín. B
0	48 49	80 81
1	2 3	- -
2	20 21	- -
3	38 39	- -
4	- -	- -
5	10 11	- -
6	- -	- -
7	- -	- -

b) ¿Tiene sentido tener una memoria caché más grande que la memoria principal (física)? De ejemplos y justifique su respuesta. **(2 ptos.)**
Solución: No, no tiene sentido si se utiliza un esquema donde la memoria caché se ubique después de la MMU, ya que todas las peticiones de memoria tendrán como límite el tamaño de la memoria principal (CPU sólo conoce tamaño de la memoria principal, no de la caché). Tampoco tiene sentido sacar la memoria principal y dejar la caché como si fuera ahora la principal. Una caché no puede funcionar sin la noción de espacio direccionable virtual o espacio direccionable físico, de la manera en que está construida una CPU y la caché dentro de ella (la CPU no sabe que existe la caché, sólo los espacios direccionables).

Por otro lado, si la caché se ubica antes de la MMU, entonces una caché mayor a la memoria principal traería ventajas, ya que podría almacenar el contenido de marcos que se encuentran en el swap-file.

c) Un computador con espacio direccionable de 128 bits tiene una caché *fully-associative* con líneas de 16 bytes. ¿A qué porcentaje de la capacidad total de la caché para almacenar datos corresponde el espacio utilizado por los tags? **(2 ptos.)**
Solución: Por cada línea se necesita unicamente un tag, si se calcula el porcentaje para una línea, este es válido para toda la caché. Si cada línea tiene 16 palabras, el tag tendrá un tamaño de $128 - 4 = 124$ bits. Dado esto, el porcentaje utilizado por los tags sera $\frac{124}{16 \times 8} = \frac{124}{128} = \frac{31}{32} \approx 0{,}97$.

Pregunta 3

a) En promedio, ¿cuántos ciclos por salto pierde un computador con un pipeline de 12 etapas, si su unidad predictora acierta el 75 % de las veces? Asuma que los saltos se realizan en la penúltima etapa. **(1 pto.)**
Solución: Dado que los saltos se realizan en la etapa 11, cada vez que haya un error en la predicción, ser perderán 10 ciclos. Luego, en promedio, el computador perderá $10 \times 0{,}25 = 2{,}5$ ciclos por salto.

b) En caso que la única solución para solucionar un *hazard* de datos sea introducir una burbuja, ¿cuál es el momento más adecuado para hacerlo? **(1 pto.)**
Solución: La idea central es introducir lo antes posible la burbuja para evitar posibles modificaciones a la memoria, que la dejen en un estado inconsistente y que después haya que eliminar. El momento ideal es apenas se conocen las señales de control, o sea, en la etapa ID, justo después de pasar por la unidad de control.

c) Considere el siguiente código escrito para el assembly del computador básico con *pipeline*:

```
ADD A, B
MOV (var0), A
MOV (var1), B
```

a) Indique con un diagrama los *hazards* que se generan al ejecutar el código anterior. **(1 pto.)**
Solución: Hay un hazard entre las líneas 1 y 2 ya que el valor de A no ha alcanzado a hacer WB cuando se requeire su valor para la instrucción siguiente.

b) Indique que sucede al intercambiar el orden de la segunda y tercera línea del código. ¿Cambia el resultado final? En caso de existir *hazards*, indíquelos con un diagrama. **(1 pto.)**
Solución: Hay un hazard entre las líneas 1 y 3, ya que se requiere el valor de A que aún no ha hecho WB cuando pasa la línea 3 por EX.

c) Para los dos casos anteriores, en caso de existir *hazards*, indique como los detectan las *forwarding units* correspondientes, especificando las señales de control participantes. **(2 ptos.)**
Solución: En el primer caso, la FU1 considera las señales ID/EX.AluA, MEM/WB.LoadB que indican que se requiere el valor de A antes que haya hecho WB. Para el segundo caso, la FU2 deberá leer MEM/WB.LoadA, EX/MEM.MemIn y MEM/WB.RegIn. El primer par de señales identifica al Hazard y la tercera indica que la FU2 puede subir la señal FowardDin para hacer el forward que corresponde.

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



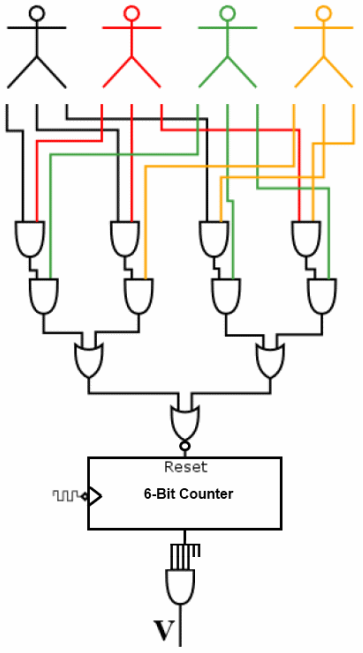
IIC2343 – Arquitectura de Computadores (II/2011)

Solución Examen

Pregunta 1 (I1)

Diseñe un sistema para la toma de decisiones de un directorio de una empresa constituido por 4 personas que cambian constantemente de opinión. Uno de los estatutos de la empresa dice que el directorio vota a favor una moción si al menos 3 de los miembros del directorio mantienen su voto a favor por más de 1 minuto seguido (esto no quiere decir necesariamente que los mismos tres miembros deben mantener su voto por más de un minuto). El circuito digital que diseñe debe estar constituido solamente por compuertas lógicas y elementos de almacenamiento (flip-flops, registros, contadores, etc.) y además dispone de una señal de clock de 1 Hz. Como variables de entrada considere $D1$, $D2$, $D3$, $D4$, que son los votos de cada miembro del directorio (1 voto a favor y 0 voto en contra), cada una de estas variables se mantiene en 1 sólo si el director correspondiente mantiene presionado un botón. La salida debe ser una variable binaria V que señale el voto final del directorio (1 a favor y 0 en contra). **(6 ptos.)**

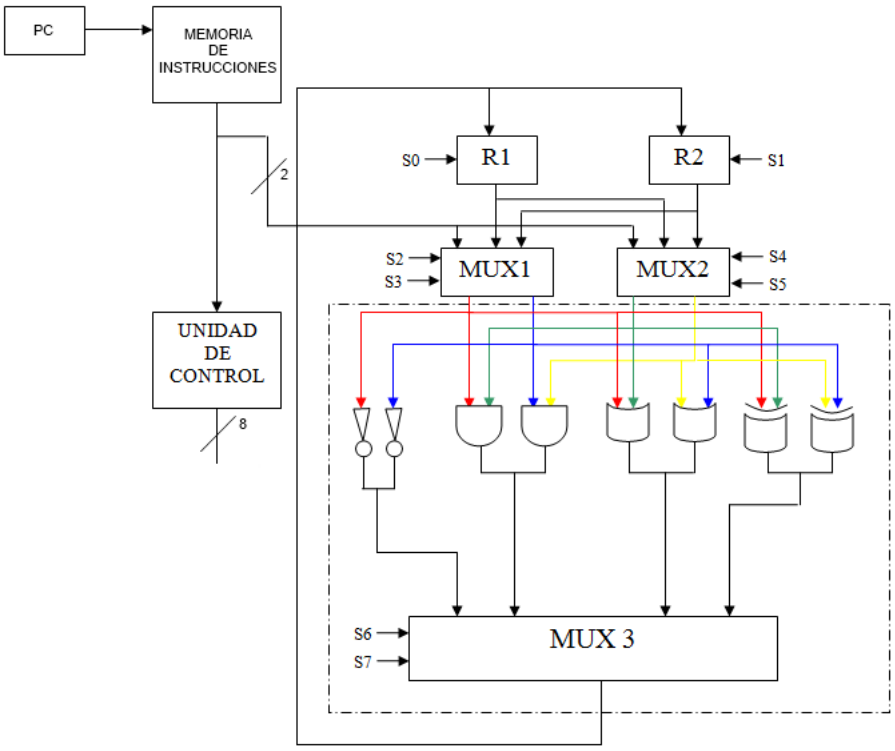
Solución:



Pregunta 2 (I2)

a) Diseñe un computador con arquitectura Harvard, de dos bits y sin memoria de datos, que realice las operaciones AND, OR, XOR y NOT. El computador debe tener dos registros acumuladores, R1 y R2, y permitir el uso de literales. Diseñe la ALU y muestre claramente las señales de control. **(3 ptos.)**

Solución:



b) Escriba un programa en assembly x86 que ordene de mayor a menor una lista de N bytes ($0 < N < 256$) enteros sin signo almacenados en la variable *arreglo*. Por problemas de capacidad de memoria, la lista ordenada debe quedar en el mismo lugar de la memoria y no se puede utilizar memoria RAM adicional para realizar el ordenamiento. El valor de N está almacenado en la variable N. **(3 ptos.)**

Solución:

```
org 100h

LEA BX, arreglo
MOV SI, [N]

outer_loop:
  DEC SI
  JZ end_outer
  MOV DI, [N]

inner_loop:
  DEC DI
  JZ outer_loop

compare:
  MOV AL, [BX + DI - 1]
  MOV DL, [BX + DI]
  CMP AL, DL
  JNL inner_loop

swap:
  MOV [BX + DI], AL
  MOV [BX + DI - 1], DL
  JMP inner_loop

end_outer:
```

Pregunta 3 (I3)

a) Describa el flujo completo de manejo de interrupciones en un computador x86. **(3 ptos.)**
Solución:

i. Dispositivo envía señal IRQ al controlador.
ii. PIC revisa su registro IMR, si la interrupción no está enmascarada la atiende, marcando un 1 en el bit correspondiente del Interrupt Request Register.
iii. PIC decide cual de las interrupciones actuales es prioritaria (menor IRQ) y marca un 1 en el bit correspondiente del In-Service Register.
iv. PIC envía interrupción (INT) a la CPU.
v. CPU termina de ejecutar la instrucción actual.
vi. CPU revisa si el flag de interrupciones está activo (IF = 1), en cuyo caso va a atender a la interrupción.
vii. CPU deshabilita la atención de más interrupciones (IF=0).
viii. CPU guarda en el stack los condition codes.
ix. CPU envía INTA para saber quien interrumpió.
x. PIC revisa In-Service Register para saber el id del IRQ que está siendo atendido, y envía mediante bus de datos.
xi. CPU usa el id para buscar dirección de ISR en el vector de interrupciones.
xii. CPU llama a la ISR asociada al dispositivo (CALL Mem[id]).
xiii. ISR respalda el estado actual de la CPU.
xiv. ISR ejecuta su código.
xv. ISR envía un comando EOI al PIC, con lo cual éste setea en 0 el In-Service Register, indicando que terminó la atención de la interrupción.
xvi. ISR devuelve el estado previo a la CPU.
xvii. ISR retorna.
xviii. CPU recupera condition codes desde el stack.
xix. CPU rehabilita la atención de interrupciones (IF=1).

b) La memoria principal de un computador tiene 16 bloques y la caché 4. La CPU ejecuta un programa que requiere los bloques 5, 7, 5, 4, 10, 5, 3, 12, 12, 15, 0, 5, 10, 5, 4 en el orden señalado. Suponiendo que inicialmente la memoria caché está vacía y que la función de correspondencia de la memoria caché es directa, indique cuál es la tasa de aciertos de la memoria caché. **(3 ptos.)**
Solución:

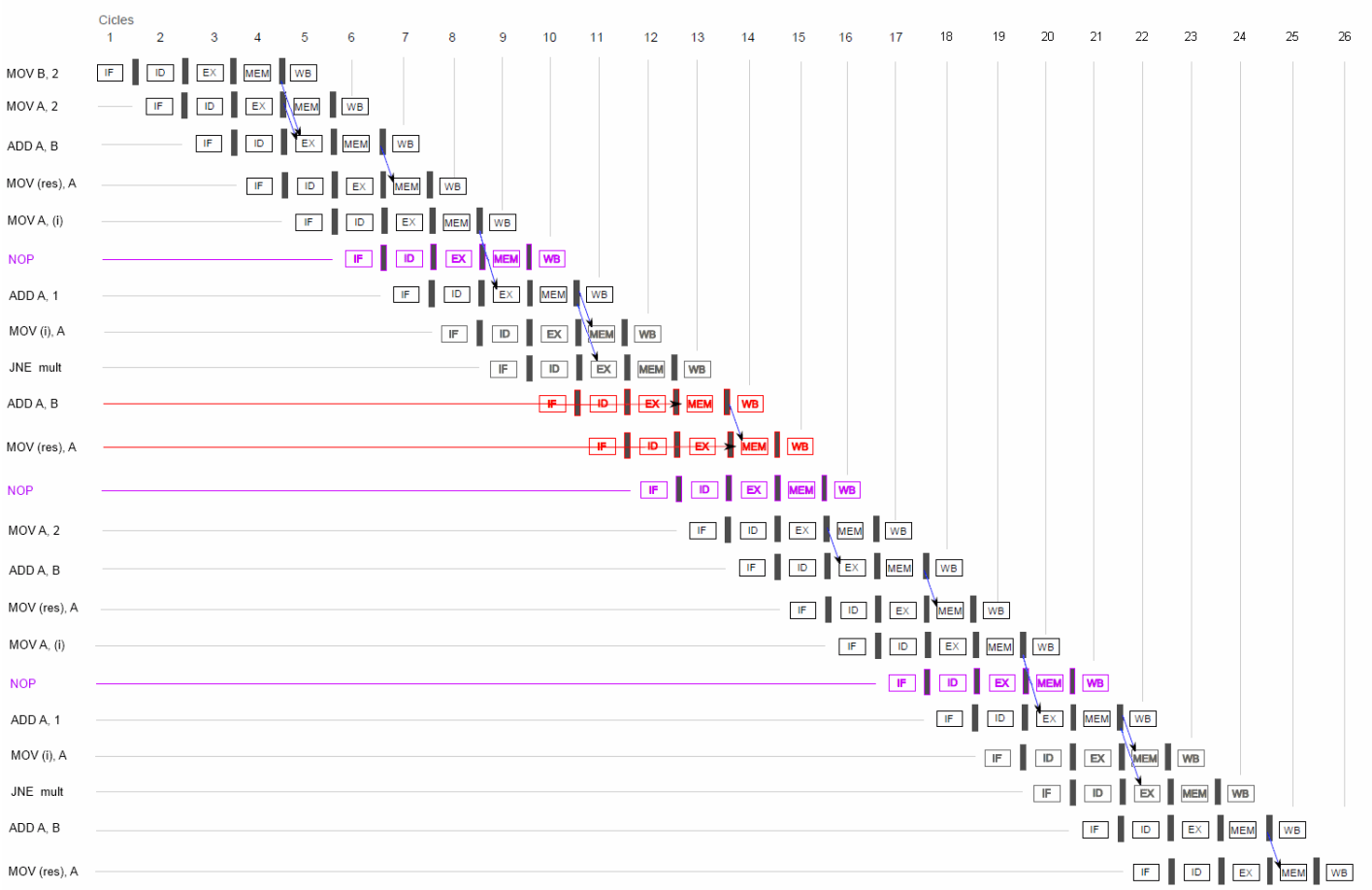
Bloque	5	7	5	4	10	5	3	12	12	15	0	5	10	5	4
0				4				12	12		0				4
1	5		5			5						5		5	
2					10								10		
3		7					3			15					
hit/miss	m	m	h	m	m	h	m	m	h	m	m	h	h	h	m
hit rate=	0.4														

Pregunta 4: ILP

a) Para el siguiente código, determine el número de ciclos que se demora, detallando los estados del pipeline para cada instrucción. El pipeline tiene forwarding implementado entre todas sus etapas, el manejo de stalling se realiza por software y se tiene predicción de salto asumiendo que no ocurre. Detalle en el diagrama cuando ocurre forwarding, stalling y flushing. **(5 ptos.)**

```
    DATA
            res     0
            i       0
    CODE
            MOV B,2
    mult:   MOV A,2
            ADD  A, B
            MOV (res), A
            MOV A, (i)
            ADD  A,1
            MOV (i), A
            JNE  mult
            ADD  A, B
            MOV (res), A
```

Solución:



b) Describa una modificación a los mecanismos de stalling y flushing usados anteriormente, para disminuir el número de ciclos que toma ejecutar el código. **(1 pto.)**
Solución: Existen múltiples alternativas para esta pregunta. Una posible respuesta es que la predicción de saltos alterne entre predecir salto la primera vez, la segunda predecir que no hay salto y así sucesivamente. Con esto se logra evitar el flushing, por lo que se ahorran 3 ciclos.

Pregunta 5: Resumen del Curso

Responda verdadero o falso y justifique. Cada correcta es $0,6$ ptos. y cada incorrecta $-0{,}3$ ptos.

a) El número decimal –45 en complemento a dos de 8 bits se representa como: 1101 0001
Solución: Falso, 45 en binario es 0010 1101, su inverso 11010010 y su complemento a 2 es 1101 0011.

b) Los bits del exponente en el estándar IEEE754 representan se interpretan como un número en complemento a 2.
Solución: Falso, el exponente se interpreta como un número binario sin signo.

c) Los condition codes permiten la existencia de salto condicionales en el computador básico.
Solución: Verdadero, la verificación de estos flags, obtenidos en base a comparaciones, permite decidir el salto.

d) OISC (One Instruction Set Computer) es un caso particular de la arquitectura Harvard.
Solución: Falso, es un caso particular de una ISA RISC.

e) Las instrucciones de la ISA x86 toman 2 ciclos en ejecutarse.
Solución: Falso, al ser una ISA CISC, las instrucciones son complejas por lo que pueden tomar más de 2 ciclos, como por ejemplo la instrucción MUL.

f) En un computador con acceso directo a memoria, la CPU le delega al DMA la transferencia de datos entre la memoria y un periférico.
Solución: Verdadero, el controlador de DMA se encarga de la transferencia y le informa cuando esta ha terminado.

g) Dos formas de establecer la correspondencia entre la caché y la memoria principal son: directa e indirecta.
Solución: Falso, las funciones de correspondencia son directa, fully associative y n-way associative.

h) La MMU es parte integral del manejo de caché en un computador.
Solución: Falso, la MMU es parte integral del manejo de la memoria virtual.

i) La función del ciclo fetch de una CPU es ejecutar una instrucción.
Solución: Falso, su función es obtener la instrucción desde la memoria.

j) La arquitectura Harvard permite evitar completamente los hazards estructurales.
Solución: Falso, la arquitectura Harvard evita el hazard estructural de los accesos simultáneos a memoria, pero no evita los que tienen relación con otras unidades funcionales.

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



IIC2343 – Arquitectura de Computadores (II/2012)

Solución Examen

Pregunta 1

a) ¿Cuáles fueron los aporte de Shannon al desarrollo de los computadores? **(1 pto.)**

Solución: Shannon logró implementar físicamente circuitos lógicos de lógica booleana mediante relés. Para esto definió, por cada conectivo lógico binario, una compuerta lógica que luego era implementada mediante relés.

b) Comente sobre los problemas relacionados con los números de punto flotante y los números reales. **(1 pto.)**

Solución: Los números de punto flotante no pueden representar de manera completa y correcta todos los números reales, debido a que los números de punto flotante tienen un espacio acotado para representar una cantidad posiblemente infinita de cifras. Además, el problema de números infinitos inherente a los sistemas posicionales agrava esta situación.

c) Describa al menos 2 ventajas de la arquitectura Harvard por sobre la Von Neuman. **(1 pto.)**

Solución: La arquitectura Harvard, al tener memorias de datos e instrucciones separadas, puede realizar la lectura de instrucciones y de datos de manera simultánea. Además, esto permite evitar hazards estructurales por acceso simultáneo a instrucciones y datos.

d) ¿Qué implicancia tiene en el tamaño de los programas el eliminar la conexión entre memoria de datos y PC en el computador básico? **(1 pto.)**

Solución: El eliminar esta conexión implica eliminar el soporte para subrutinas, por lo que en la gran mayoría de los casos los programas tendrán un mayor tamaño debido a la repetición de código.

e) ¿Es siempre el esquema de interrupciones el más eficiente para transferir datos con los dispositivos? Justifique su respuesta en cualquiera de los dos casos. **(1 pto.)**

Solución: No necesariamente. Existen dispositivos como el teclado donde las transferencias son tan pocas y pequeñas que el *overhead* del uso de interrupciones hace el proceso más lento.

f) ¿En que se basa el concepto de memoria caché para ser efectivo en la práctica?. **(1 pto.)**

Solución: Se basa en los principios de localidad espacial y localidad temporal.

Pregunta 2

a) Un computador tiene un espacio direccionable de 36 bits y una memoria física de 2GB, dividida en marcos de 2KB. ¿Cuantas páginas virtuales y marcos físicos tiene este computador? Considerando los flags vistos en clases, ¿que tamaño tiene una tabla de páginas? **(2 ptos.)**

Solución:

- Nº páginas: $\frac{2^{36}}{2^{11}} = 2^{25}$
- Nº marcos: $\frac{2^{31}}{2^{11}} = 2^{20}$
- Tamaño páginas: $2^{25} \times (20 + 2)$ bits

b) Un procesador usa direcciones de memoria de 16 bits, páginas de 256 palabras y una TLB fully-associative de 8 entradas con política de reemplazo LRU. En un instante, la TLB de un proceso se encuentra en el estado indicado a continuación:

Nº página	Nº marco	Validez	LRU
0x01	0x11	1	0
0x00	0x00	0	7
0x10	0x13	1	1
0x20	0x12	1	5
0x00	0x00	0	6
0x11	0x14	1	4
0xAE	0x15	1	2
0xFF	0x16	1	3

Asumiendo que todas las páginas utilizadas actualmente por el proceso están en la TLB y que los marcos 0x17, 0x18, 0x19 se encuentra libres, actualice la tabla anterior después de ejecutar la siguiente secuencia de accesos a memoria **(4 ptos.)**:

- 0x11F0 (Lectura)
- 0x1301 (Escritura)
- 0x20AE (Escritura)
- 0x2332 (Escritura)
- 0x20FF (Lectura)
- 0x3415 (Escritura)

Solución: Para la solución sólo es necesario actualizar los valores de la tabla al final de todos los accesos, no después de cada uno.

- 0x11F0 (Lectura): Hit.
- 0x1301 (Escritura): Miss sin sustitución (se utiliza entrada no válida dada por LRU), se mapea página 0x13 a marco 0x17.
- 0x20AE (Escritura): Hit.

- 0x2332 (Escritura): Miss sin sustitución (se utiliza entrada no válida dada por LRU), se mapea página 0x23 a marco 0x18.
- 0x20FF (Lectura): Hit.
- 0x3415 (Escritura): Miss con sustitución dada por LRU, se mapea página 0x34 a marco 0x19.

Nº página	Nº marco	Validez	LRU
0x01	0x11	1	5
0x13	0x17	1	3
0x10	0x13	1	6
0x20	0x12	1	1
0x23	0x18	1	2
0x11	0x14	1	4
0xAE	0x15	1	7
0x34	0x19	1	0

Pregunta 3

a) Determine el número de ciclos que se demora el siguiente código, detallando en un diagrama los estados del pipeline por instrucción. El pipeline tiene forwarding entre todas sus etapas, el manejo de stalling es por software (instrucción NOP) y predicción de salto asumiendo que no ocurre. Indique en el diagrama cuando ocurre forwarding, stalling y flushing. **(4 ptos.)**

```
DATA
    var1    2
    var2    2
    res      0
    i        0
CODE
    start:  MOV A,(res)
            MOV B,(var2)
            ADD A,B
            MOV (res),A
            MOV A,(i)
            ADD A,1
            MOV (i),A
            MOV B,(var1)
            JNE start
            MOV A,(res)
            ADD A,3
            MOV (res),A
```

Solución: Por cada instrucción ejecutada se indica la existencia de forwarding (recibido), flushing o stalling.

1. MOV A,(res):
2. MOV B,(var2):
3. NOP: Stalling para permitir forwarding entre instrucciones 2. y 4.
4. ADD A,B: Forward recibido desde registro MEM/WB de instrucción 2. a entrada etapa EX.
5. MOV (res),A: Forward recibido desde registro MEM/WB de instrucción 4. a entrada etapa MEM.
6. MOV A,(i):
7. NOP: Stalling para permitir forwarding entre instrucciones 6. y 8.
8. ADD A,1: Forward recibido desde registro MEM/WB de instrucción 6. a entrada etapa EX.
9. MOV (i),A: Forward recibido desde registro MEM/WB de instrucción 8. a entrada etapa MEM.
10. MOV B,(var1)
11. NOP: Stalling para permitir forwarding entre instrucciones 10. y 12.
12. JNE start: Forward recibido desde registro MEM/WB de instrucción 10. a entrada etapa EX.
13. FLUSH $\Rightarrow$ ~~MOV A,(res):~~
14. FLUSH $\Rightarrow$ ~~NOP: Stalling para permitir forwarding entre instrucciones 13. y 15.~~

15. FLUSH ⇒ ~~ADD A,3: Forward recibido desde registro MEM/WB de instrucción 10. a entrada etapa EX.~~
16. MOV A,(res):
17. MOV B,(var2):
18. NOP: Stalling para permitir forwarding entre instrucciones 2. y 4.
19. ADD A,B: Forward recibido desde registro MEM/WB de instrucción 2. a entrada etapa EX.
20. MOV (res),A: Forward recibido desde registro MEM/WB de instrucción 4. a entrada etapa MEM.
21. MOV A,(i):
22. NOP: Stalling para permitir forwarding entre instrucciones 6. y 8.
23. ADD A,1: Forward recibido desde registro MEM/WB de instrucción 6. a entrada etapa EX.
24. MOV (i),A: Forward recibido desde registro MEM/WB de instrucción 8. a entrada etapa MEM.
25. MOV B,(var1)
26. NOP: Stalling para permitir forwarding entre instrucciones 10. y 12.
27. JNE start: Forward recibido desde registro MEM/WB de instrucción 10. a entrada etapa EX.
28. MOV A,(res)
29. NOP: Stalling para permitir forwarding entre instrucciones 28. y 30.
30. ADD A,3: Forward recibido desde registro MEM/WB de instrucción 28. a entrada etapa EX.
31. MOV (res),A: Forward recibido desde registro MEM/WB de instrucción 30. a entrada etapa MEM.

También es valido entregar como respuesta para las instrucciones 13, 14 y 15 la siguiente secuencia MOV A,(res); ADD A,3; MOV (res),A sin indicar forwarding. Dado que la cantidad de instrucciones ejecutadas fue 31, la cantidad de ciclos que tomó el programa fue 35.

b) Reordene el código anterior para disminuir el número de ciclos que demora, pero manteniendo el mismo resultado. Indique explícitamente cuanto es el ahorro. **(2 ptos.)**

Solución: Para disminuir la cantidad de ciclos utilizados, basta mover la instrucción MOV B,(var1) un línea hacia arriba, lo que produce que no se deba introducir un NOP para asegurar la consistencia. De esta manera, el programa puede ejecutarse en 33 ciclos.

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



IIC2343 – Arquitectura de Computadores (I/2013)

Solución Examen

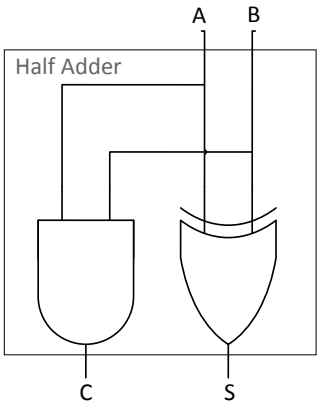
Pregunta 1

a) ¿Cómo se puede calcular la cantidad de bits necesarios para representar a los elementos de un conjunto acotado de números enteros? **(1 pto.)**
Solución: Se necesita tomar el número con mayor valor absoluto del conjuto, τ, y ver cuanto bits son necesarios para representar este valor absoluto. Luego, dado que estamos trabajando con números enteros, se agrega un bit más para poder utilizar el complemento a 2. La expresión final para la cantidad de bits es entonces: : $2 + \lfloor \log_2 \mid \tau \mid \rfloor$. En caso que el número con mayor valor absoluto sea 0, sólo se necesita un bit.

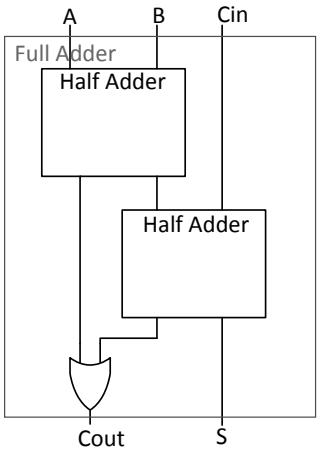
b) Explique el concepto de endianness en memoria. ¿Es necesario especificar el endianness cuando se lee desde un registro? **(1 pto.)**
Solución: El endianness indica el orden en que se almacenan los bytes de un dato en una memoria. Esto es imprescindible, ya que en las memorias, cada direccion indica una palabra (byte). En los registros no es necesario definir endianness, ya que los valores se almacenan como un todo y el acceso es al registro completo.

c) ¿Es posible eliminar completamente el problema con los números infinitos usando una representación de punto flotante? **(1 pto.)**
Solución: No es posible eliminar este problema, ya que es inherente a las representaciones posicionales y depende de la base. Es posible aplacarlo usando formatos numéricos avanzados, pero nunca se puede eliminar del todo.

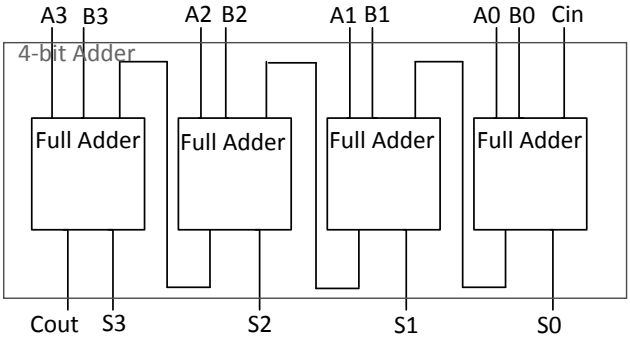
d) Diseñe en detalle, a partir de compuertas básicas, un sumador/restador de 4 bits. **(3 ptos.)**
Solución: El primer paso corresponde al diseño de un half-adder de 1-bit:



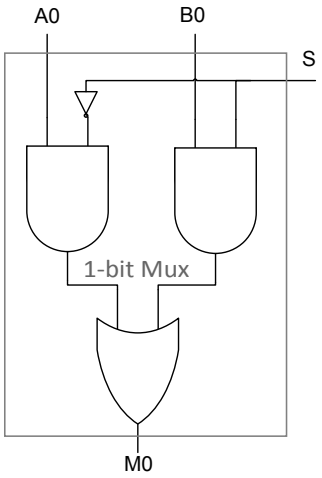
Luego, usando half-adders, diseñamos un full-adder:

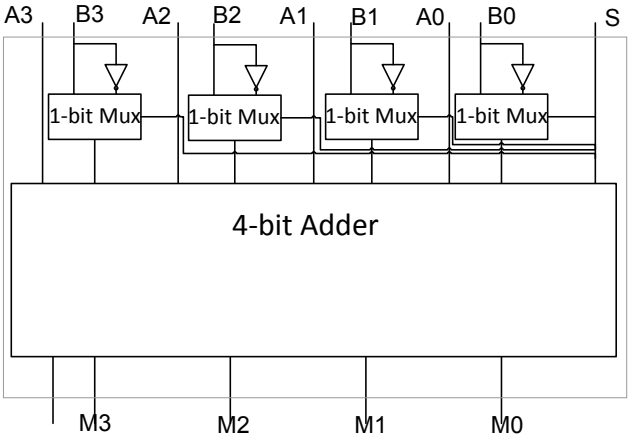


Usando 4 full-adders, se construye de manera trivial un sumador de 4 bits:



Finalmente, diseñamos un mux de 1 bit y junto con un sólo adder y aprovechando al relación entre complemento a 2 y la negación de bits, construimos el sumador/restador de 4 bits:


A0
B0
S
1-bit Mux
M0


A3
B3
A2
B2
A1
B1
A0
B0
S
1-bit Mux
1-bit Mux
1-bit Mux
1-bit Mux
4-bit Adder
M3
M2
M1
M0

Pregunta 2

a) ¿Que modificación realizaría al computador básico para reducir el número de instrucciones de los programas? Describa brevemente como lo implementaría. **(1 pto.)**
Solución: Una modificación sencilla y muy efectiva es agregar a la ALU unidades de multiplicación y división. De esta manera, cada vez que se necesite realizar una de estas operaciones, se necesitará sólo una instrucción, a diferencia del caso actual, en donde se necesita realizar un loop.

b) ¿Qué es stdcall y por qué es necesario que exista? **(1 ptos.)**
Solución: stdcall es una de las muchas convenciones de llamada a subrutina existentes. Son necesarias para específicar sin ambigüedades la manera en que se entregan y reciben datos desde una subrutina.

c) Escriba en assembly x86 un programa que encuentre la moda de un arreglo de número naturales. El tamaño del arreglo es arbitrario y en caso de haber mútiples modas, se debe retornar la de menor valor. **(4 ptos.)**
Solución:

```
        MOV CX, 0
        MOV SI, 0
        start:
        CMP SI, n
        JE end
        LEA BX, array
        MOV CL, [BX + SI]   ;CL almacena el valor a analizar
        MOV DI, CX
        SHL DI, 1           ;los elementos de freq son de tipo dw
        LEA BX, freq
        MOV AX, [BX + DI]   ;esta linea y las dos siguientes
        ADD AX, 1           ;actualizan en freq las apariciones
        MOV [BX + DI], AX   ;del valor almancenado en CL
        CMP AX, mode_count  ;si las apariciones son mayores que
        JL next             ;las de la moda actual, se compara su valor
        CMP CL, mode        ;si el valor en CL es menor que la moda
        JG next             ;actual y tambien tiene mas apariciones, se
        MOV mode, CL        ;actualiza la informacion en las variables
        MOV mode_count, AX  ;mode y mode_count
        next:
        ADD SI, 1
        JMP start
        end:

        mode        db    127
        mode_count  dw    0
        n           dw    ?
        array       db    ?,?,...
        freq        dw    0,0,...
```

Pregunta 3

a) ¿Cuál es la función del controlador de DMA? Describa en detalle con qué elementos se conecta y comunica. **(1 pto.)**
Solución: El controlador de DMA se encarga de realizar transferencias de datos entre la memoria y dispositivos de I/O mapeados a memoria. Se comunica con la memoria, los dispositivos y con la CPU. Debe estar conectado al menos al bus de I/O (datos), al de direcciones y al controlador de interrupciones o la CPU.

b) Describa en detalle y compare las interrupciones por software y las excepciones. **(1 pto.)**
Solución: Una interrupción de software es una interrupción generada mediante el llamado, por parte de un programa, a una instrucción especial (INT) que a su vez llama a la ISR correspondiente. Por otra lado, una excepción es generada por la CPU cuando detecta una evento especial o un error que no puede o no debe ser manejado por el programa.

c) El tamaño de páginas/marcos es un parámetro esencial para tener un buen rendimiento. ¿Cómo fijaría este parámetro en la práctica? **(1 pto.)**
Solución: Una manera de fijar este parámetro es ejecutar una gran cantidad de programas, variando el tamaño de las páginas, y medir su rendimiento y cantidad de memoria utilizada como función del tamaño de las páginas. Luego, se utiliza el tamaño que presente la mejor combinación de estos valores.

d) ¿Cuál es la función de la TLB? ¿Es necesario reiniciarla cuando hay un cambio de contexto y en caso de ser necesario, como se puede evitar esto? Todas las respuestas deben ser justificadas en detalle. **(1 pto.)**
Solución: La TLB es una memoria caché muy pequeña utilizada para acelerar el funcionamiento de la MMU. Al almacenar la transformación página/marco y al ser esta diferente para cada programa, es necesario limpiarla en los cambios de contexto, a menos que se agregue a cada bloque de esta caché una columna que indique a que proceso pertenece la información almacenada. De esta manera, no es necesario borrar la TLB, ya que los bloques con un identificador de proceso diferente al actual será tomados como no validos.

e) En un computador, la memoria principal tiene 16 bloques y la memoria caché tiene sólo 4. La CPU ejecuta un programa que requiere de la memoria principal la primera palabra de los bloques 5, 7, 5, 4, 10, 5, 3, 10, 5, 4 en el orden señalado. Suponiendo que inicialmente la memoria caché está vacía y que la función de correspondencia de la memoria caché es de mapeo directo, indique cuál es la tasa de aciertos de la memoria caché. **(2 ptos.)**
Solución:

- 5 (Bloque 1) - Miss
- 7 (Bloque 3) - Miss
- 5 (Bloque 1) - Hit
- 4 (Bloque 0) - Miss
- 10 (Bloque 2) - Miss
- 5 (Bloque 1) - Hit
- 3 (Bloque 3) - Miss

- 10 (Bloque 2) - Hit
- 5 (Bloque 1) - Hit
- 4 (Bloque 0) - Hit

Luego, el hit rate es $\frac{5}{10} = 0{,}5$.

Pregunta 4

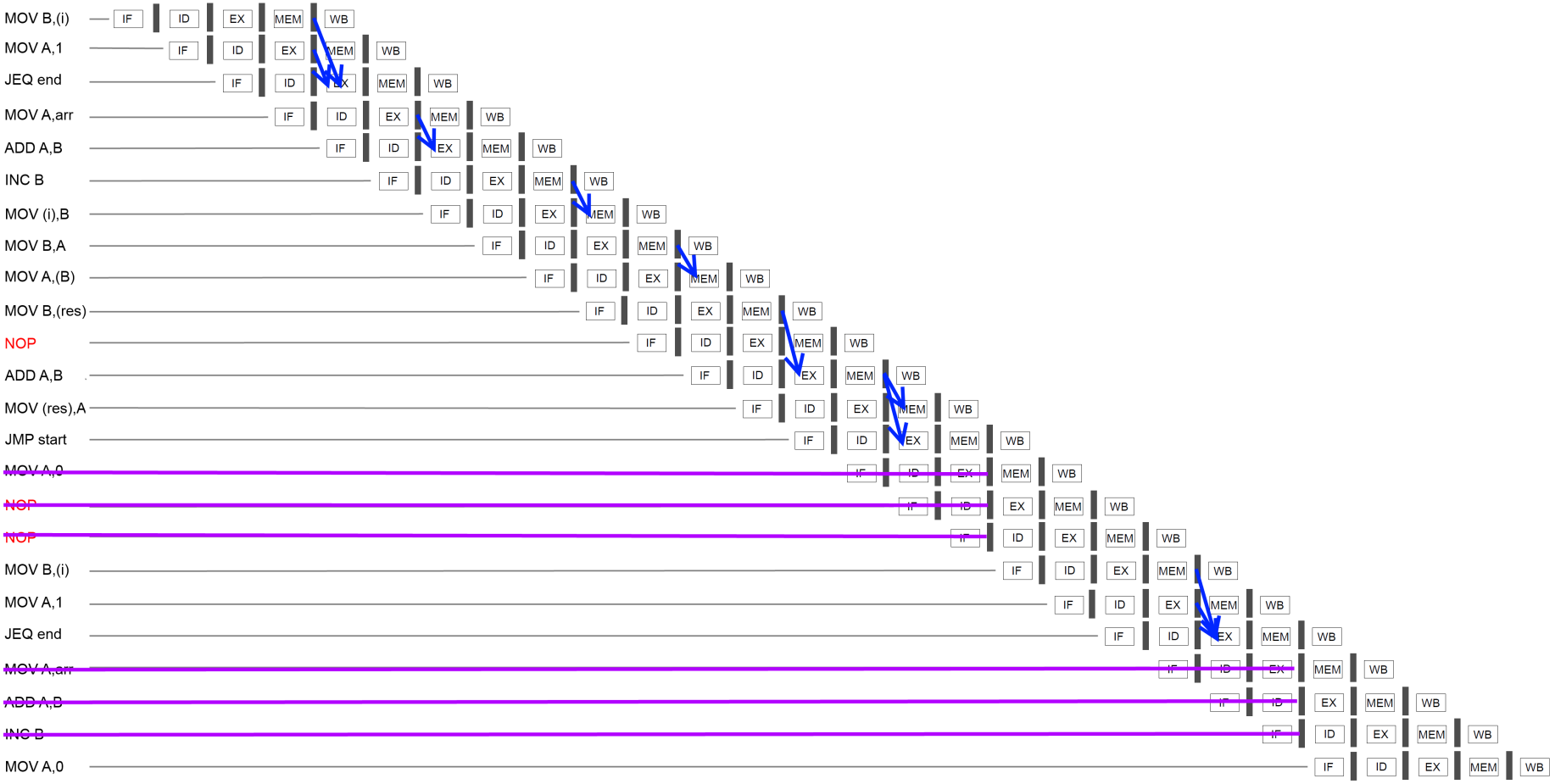
a) Indique qué capacidades/intrucciones gana y pierde el computador básico al agregar soporte para paralelismo a nivel de instrucción. **(1 pto.)**
Solución: Logicamente, el computador gana la capacidad de procesar multiples instrucciones en paralelo. Dentro de las características que se pierden está i) el llamado a subrutinas, ya que no hay SP ni conexión entre memoria y PC, i) los opcodes N, C y V, la conexión entre la salida de memoria y el MUX B, la entrada de datos de la memoria ahora sólo puede provenir de los registros A y B y no de la ALU y la dirección de la memoria ahora proviene de la ALU.

b) Un computador con microarquitectura avanzada posee un pipeline de 30 etapas. ¿Cuál es el elemento de hardware, sin contar los registros entre etapas, que tiene el mayor impacto (positivo y negativo) en el rendimiento ? Justifique su respuesta. **(1 pto.)**
Solución: Dado que se tiene un pipeline muy profundo, la unidad más importante será la de salto, más especificamente la predicción de saltos, ya que si está mal implementada, cada salto mal estimado tendrá un costo muy alto en ciclos, logicamente dependiendo de la profundidad a la que se encuentre la unidad de salto.

c) Determine el número de ciclos que se demora el siguiente código y la cantidad de ciclos perdidos, si se utiliza el computador básico con pipeline de 5 etapas. Indique además la cantidad de ciclos que hubiese tomado en un computador sin pipeline. Detalle en un diagrama los estados del pipeline por instrucción. El pipeline tiene forwarding entre todas sus etapas, el manejo de stalling es por software (instrucción NOP) y predicción de salto asumiendo que no ocurre. Indique en el diagrama cuando ocurre forwarding, stalling y flushing. **(4 ptos.)**

```
DATA
    arr     2
            3
    i       0
    res     0
CODE
    start:  MOV B, (i)
            MOV A, 1
            JEQ end
            MOV A, arr
            ADD A, B
            INC B
            MOV (i), B
            MOV B, A
            MOV A, (B)
            MOV B, (res)
            ADD A, B
            MOV (res), A
            JMP start
    end:    MOV A, 0
```

Solución: Basado en el siguiente diagrama, los ciclos totales que toma el proceso son 28, mientras que los ciclos perdidos son 7. Finalmente, sin usar un computador con pipeline, el proceso hubiera tomado 17 ciclos.


MOV B,(i)
MOV A,1
JEQ end
MOV A,arr
ADD A,B
INC B
MOV (i),B
MOV B,A
MOV A,(B)
MOV B,(res)
NOP
ADD A,B
MOV (res),A
JMP start
MOV A,0
NOP
NOP
MOV B,(i)
MOV A,1
JEQ end
MOV A,arr
ADD A,B
INC B
MOV A,0
IF
ID
EX
MEM
WB

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

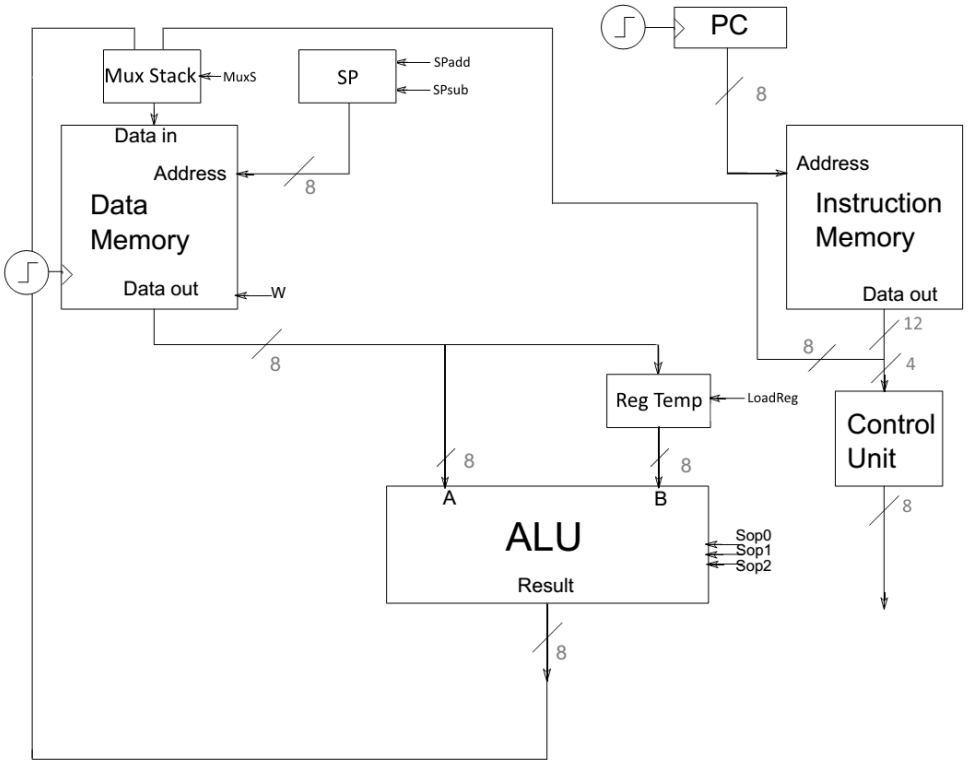
IIC2343 – Arquitectura de Computadores (II/2014)

Solución Examen

Pregunta 1

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 sólo sustituye en 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.

a) 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. **(4 ptos.)**
Solución:



b) Describa una ISA para la máquina del ítem anterior. Indique opcodes, señales de control y las instrucciones del assembly correspondientes. **(2 ptos.)**
Solución: En la tabla, se asume que las señales de control no indicadas se encuentran en 0. El SP siempre apunta al último elemento agregado al stack.

Instrucción	Opcode	Señales Control
PUSH Lit	0000	SPsub=1
	0001	W=1,MuxS=1
POP	0010	SPadd=1
ADD	0011	SPadd=1, LoadReg=1
	0100	Sop=ADD, W=1, MuxS=0
SUB	0101	SPadd=1, LoadReg=1
	0110	Sop=SUB, W=1, MuxS=0
AND	0111	SPadd=1, LoadReg=1
	1000	Sop=AND, W=1, MuxS=0
OR	1001	SPadd=1, LoadReg=1
	1010	Sop=OR, W=1, MuxS=0
XOR	1011	SPadd=1, LoadReg=1
	1100	Sop=XOR, W=1, MuxS=0
SHL	1101	Sop=SHL, W=1, MuxS=0
SHR	1110	Sop=SHR, W=1, MuxS=0
NOT	1111	Sop=NOT, W=1, MuxS=0

Pregunta 2

Una máquina acumuladora, es un tipo de computador en el cual se utiliza un sólo registro para almacenar los resultados de las operaciones aritméticas y lógicas. Una máquina acumuladora puede tener más registros (PC, SP, etc), pero sólo se puede usar el registro acumulador y la memoria o literales para realizar las operaciones. Para este ejercicio, asuma que tiene una máquina acumuladora de 8 bits, donde el registro acumulador es llamado A. Esta máquina posee las mismas instrucciones que el assembly x86, con la excepción que no existe la instrucción MOV, la cual es reemplazada por la instrucción para cargar datos en el registro acumulador, LOAD, y la instrucción para escribir datos en memoria, STORE. Recuerde que TODAS las operaciones aritméticas y lógicas almacenan su resultado en el registro acumulador, por lo que las instrucciones del assembly sólo reciben un parámetro.

a) Escriba en assembly de la máquina acumuladora, un programa que calcule el promedio de un conjunto de números enteros almacenados en memoria. **(2 ptos.)**
Solución:

```
        JMP    main
        suma     db 0
        res      db 0
        i        db 0
        dir      db
        n        db ?
        arreglo  db ?
    main:
        LEA arreglo
        ADD i
        STORE dir
        LOAD suma
        ADD [dir]
        STORE sum
        LOAD  i
        ADD   1
        STORE i
        CMP n
        JLT main
        LOAD suma
        DIV n
        STORE res
```

b) Escriba en assembly de la máquina acumuladora, un programa que calcule un histograma de un conjunto de números enteros no negativos, almacenados en memoria. Asuma que los números se encuentran el intervalo $[0, 255]$ y que el número de casilleros del histograma siempre será un divisor de 256. Cualquier detalle de implementación o aspecto que se asuma debe quedar claramente explicado. **(4 ptos.)**

Solución:

```
        JMP   main
        i           db 0
        bin         db
        dir         db
        casilleros  db ?
        ancho       db
        n           db ?
        arreglo     db ?
        .
        .
        .
        res         db 0
        .
        .
        .
    main:
        LOAD 256
        DIV  casilleros
        STORE ancho
    loop:
        LEA arreglo
        ADD i
        LOAD [A]
        DIV ancho
        STORE bin

        LEA res
        ADD bin
        STORE dir
        LOAD [A]
        ADD 1
        STORE [dir]

        LOAD  i
        ADD   1
        STORE i
        CMP n
        JLT loop
```

Pregunta 3

a) Describa la relación entre direcciones virtuales y físicas, páginas y marcos. Describa además el proceso que relaciona a todos estos elementos, nombrando todas las piezas de hardware que participan. **(1 pto.)**
Solución: La direcciones virtuales son posiciones de memoria que no existen fisicamente y que pueden estar asociadas a una dirección de memoria física, si este mapeo existe en la tabla de páginas. El espacio direccionable de la memoria virtual es igual al especio direccionable del computador. Las páginas son espacios contiguos de memoria virtual (direcciones virtuales), que están asociadas a un marco físico (direcciones físicas), mediante un mapeo dado por una tabla de páginas. La tranformación es realizada por una unidad ubicada en la CPU llamada MMU, que lee el contenido de su caché (TLB) o desde la tabla de páginas.

b) Un computador utiliza palabras de 8 bits, tiene un espacio direccionable de 64GB y una memoria principal de 2GB dividida en marcos de 2KB. ¿Cuántos bits se necesitan para describir el número de página virtual y el número de marco físico? Si todos los flags de la tabla de páginas toman 12 bits por entrada, ¿cuantos bits de espacio utiliza una tabla de páginas? **(1 pto.)**
Solución: Se necesitan 25 bits para describir el número de página y 20 bits para el número de marco físico. La tabla de páginas utiliza $2^25 \times (12 + 20) = 2^25 \times 2^5 = 2^30$ bits.

c) Complete la siguiente tabla asumiendo que por cada entrada de la tabla de páginas, se utilizan 4 bits para flags: **(1 pto.)**
Solución:

Bits Dir. Virt.	Bits Dir. Fís.	Tam. Pág.	Bits Pág.	Bits Marco	Bits por entrada
32	32	16KB	18	18	22
32	26	8KB	19	13	17
36	32	32KB	21	17	21
40	36	32KB	25	21	25
64	40	64KB	48	24	28

d) ¿Qués es un cambio de contexto y un PCB? ¿Que relación existe entre ellos? **(1 pto.)**
Solución: Un cambio de contexto es el proceso que realiza el sistema operativo para cambiar la ejecución de un proceso a otro. El PCB (Process Control Block) es una estructura almacenada por el sistema operativo, donde cada proceso tiene una y contiene todos los datos relevantes que definen su estado. El sistema operativo utiliza el PCB para respaldarrestaurar los procesos cuando hay un cambio de contexto y para tomar decisiones relacionadas con scheduling.

e) Usando la siguiente tabla, calcule los momentos en que se ejecuta y termina cada proceso, usando los esquemas FiFo y Round Robin con un intervalo de ejecución de 60. Además, calcule para cada esquema el tiempo total de espera por atención. ¿Cuál de los dos esquemas es más eficiente con respecto a esta métrica? **(2 ptos.)**

Proceso	P1	P2	P3	P4	P5
Tpo. Ejecución	120	60	180	50	300

Solución:
Fifo:

- P1: Inicio=0, Fin=120 – Espera=0
- P2: Inicio=120, Fin=180 – Espera=120
- P3: Inicio=180, Fin=360 – Espera=180
- P4: Inicio=360, Fin=410 – Espera=360
- P5: Inicio=410, Fin=710 – Espera=410

Espera total = 1070

Round Robin:

- P1: Inicio=0, Fin=60 – Inicio=290, Fin=350 Espera=230
- P2: Inicio=60, Fin=120 – Espera=60
- P3: Inicio=120, Fin=180 – Inicio=350, Fin=410 – Inicio=470, Fin=530 – Espera=350
- P4: Inicio=180, Fin=230 – Espera=180
- P5: Inicio=230, Fin=290 – Inicio=410, Fin=470 – Inicio=530,Fin=590 – Inicio=590, Fin=650 – Inicio=650, Fin=710 – Espera=410

Espera total = 1230
Luego, para este caso, es más eficiente FiFo.

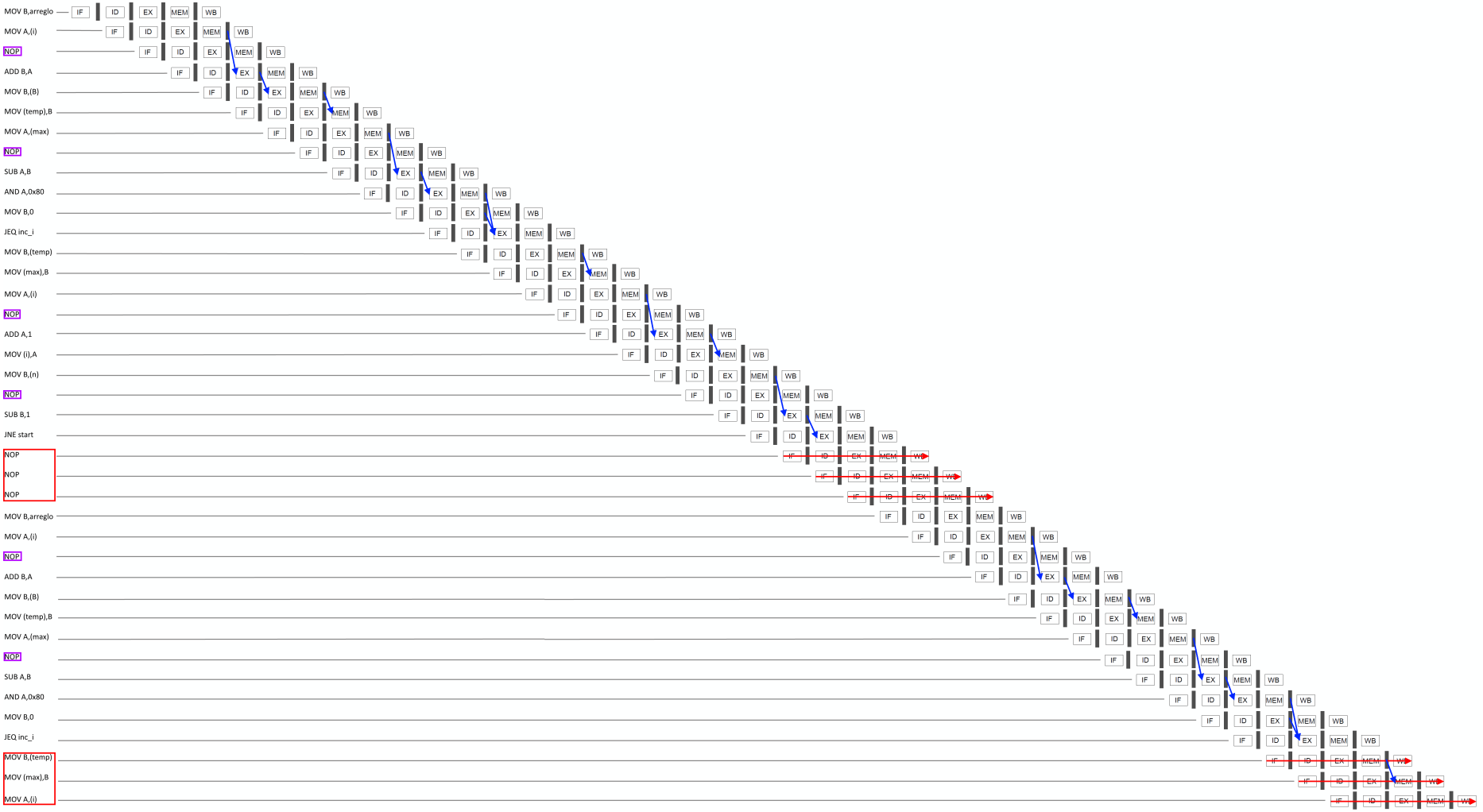
Pregunta 4

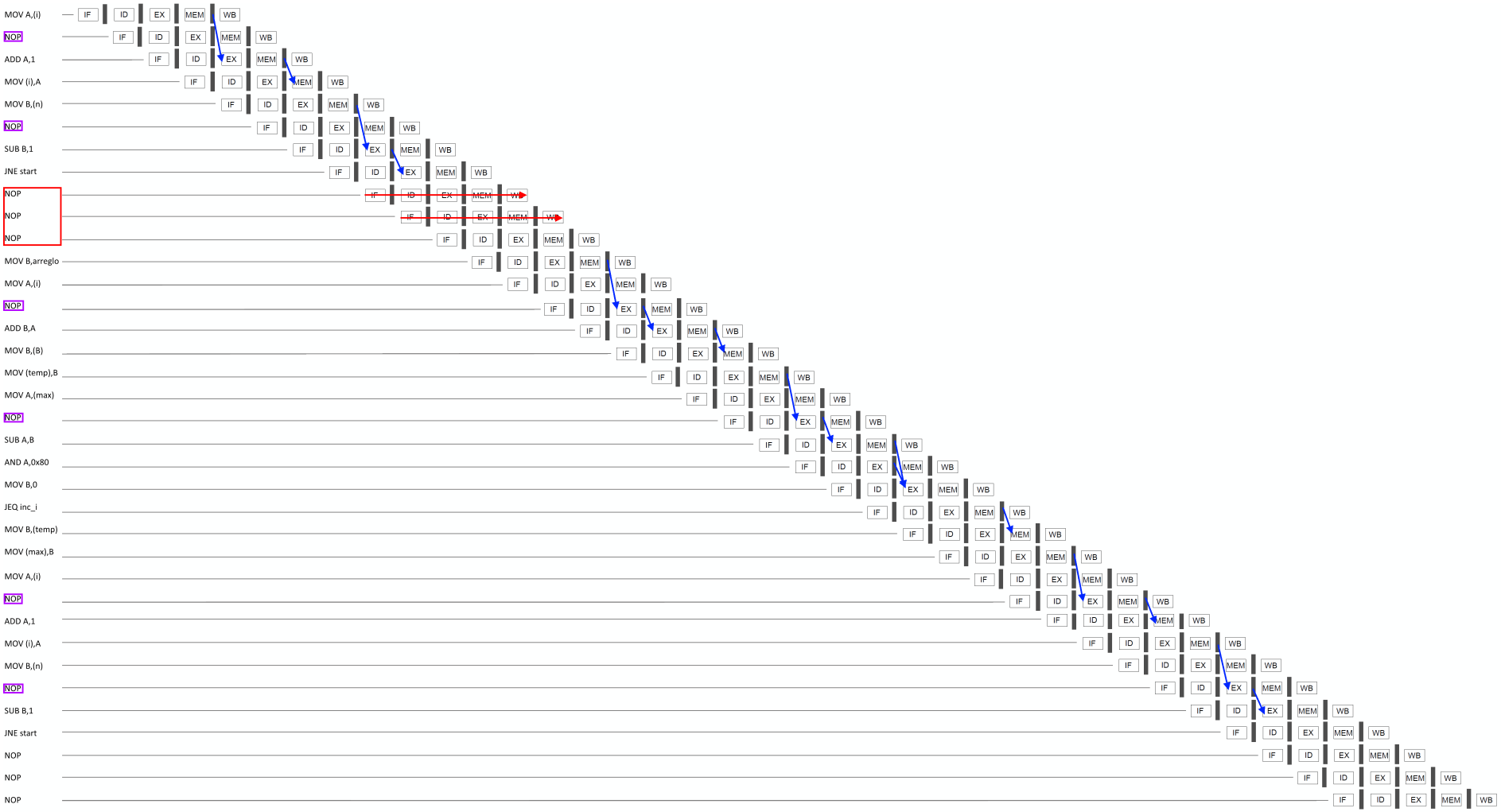
a) Describa los tipos de paralelismo SIMD y SISD. ¿Qué tipo de operaciones son las que más beneficio sacan del paralelismo SIMD?¿Es posible tener simultáneamente paralelismo del tipo SISD y SIMD? **(1 pto.)**
Solución: El paralelismo SIMD (Single Instruction Multiple Data) ocurre cuando la misma instrucción/-programa debe ser aplicado a datos distintos. Las operaciones vectoriales y matriciales son las que mayor provecho sacan de esto.
SISD ocurre cuando se realiza la ejecución paralela de un programa sobre un mismo dato. Generalmente este se logra mediante pipelining. Operaciones largas que pueden ser divididas en sub-operaciones son las que mayor ventaja obtienen de SISD.
Para que ambos esquemas, SISD y SIMD, puedan ejecutarse simultaneamente, basta que las unidades de ejecución del pipeline, sean capaces de aplicar la misma operación a múltiples datos distintos.

b) Escriba un programa para el computador básico con pipeline, que encuentre el máximo valor en un arreglo de n números enteros. **(2 ptos.)**
Solución:

```
DATA
    i         0
    max       0
    temp
    n         ?
    arreglo   ?
CODE
    start:
              MOV B, arreglo
              MOV A, (i)
              ADD B, A
              MOV B, (B)
              MOV (temp), B
              MOV A, (max)
              SUB A, B
              AND A,0x80
              MOV B, 0
              JEQ inc_i
              MOV B, (temp)
              MOV (max), B
    inc_i:
              MOV A, (i)
              ADD A, 1
              MOV (i), A
              MOV B, (n)
              SUB B, 1
              JNE start
```

c) Determine el número de ciclos que se demora el código anterior, usando el arreglo [3, 1, 7]. Detalle en un diagrama los estados del pipeline por instrucción. El pipeline tiene forwarding entre todas sus etapas, el manejo de stalling es por software (instrucción NOP) y predicción de salto asumiendo que no ocurre. Indique en el diagrama cuando ocurre forwarding, stalling y flushing. ¿Cuántos ciclos tomaría el programa si se agrega el número 2 al final del arreglo?**(3 ptos.)**
Solución: En base al siguiente diagrama, el programa toma 80 ciclos. Si se agrega un 2 al final del arreglo, el comportamiento es el mismo de la segunda iteración, la cual toma 26 ciclos. Luego el programa tomaría 106 ciclos.


MOV B,arreglo
MOV A,(i)
NOP
ADD B,A
MOV B,(B)
MOV (temp),B
MOV A,(max)
NOP
SUB A,B
AND A,0x80
MOV B,0
JEQ inc_i
MOV B,(temp)
MOV (max),B
MOV A,(i)
NOP
ADD A,1
MOV (i),A
MOV B,(n)
NOP
SUB B,1
JNE start
NOP
NOP
NOP
MOV B,arreglo
MOV A,(i)
NOP
ADD B,A
MOV B,(B)
MOV (temp),B
MOV A,(max)
NOP
SUB A,B
AND A,0x80
MOV B,0
JEQ inc_i
MOV B,(temp)
MOV (max),B
MOV A,(i)
IF
ID
EX
MEM
WB


MOV A,(i)
NOP
ADD A,1
MOV (i),A
MOV B,(n)
NOP
SUB B,1
JNE start
NOP
NOP
NOP
MOV B,arreglo
MOV A,(i)
NOP
ADD B,A
MOV B,(B)
MOV (temp),B
MOV A,(max)
NOP
SUB A,B
AND A,0x80
MOV B,0
JEQ inc_i
MOV B,(temp)
MOV (max),B
MOV A,(i)
NOP
ADD A,1
MOV (i),A
MOV B,(n)
NOP
SUB B,1
JNE start
NOP
NOP
NOP
IF
ID
EX
MEM
WB

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



IIC2343 – Arquitectura de Computadores (II/2015)

Examen

Pregunta 1

a) Describa detalladamente que pasa si se intenta ejecutar un archivo ejecutable binario del computador básico, en un computador x86 de 16 bits. **(1 pto.)**
Solución: Puede pasar cualquier cosa. Ya que son códigos ejecutables para computadores distintos, lo más probable es que no tengan los mismos opcodes. Luego, al intentar ejecutar (si es posibel evitar la prohibición del sistema operativo), el computador tendrá un comportamiento no predecible.

b) Un *disassembler* es un programa que transforma código binario ejecutable en *assembly*. Describa como funcionaría un *disassembler* para el computador básico. ¿Es posible obtener el *assembly* original a partir de un programa ubicado en la memoria de instrucciones? **(2 ptos.)**
Solución: Un *disassembler* para el computador básico trabajaría tomando cada uno de los opcodes en las correspondientes instrucciones. Esto es posible gracias a que los opcodes tienen una estructura fija y que cada uno corresponde a sólo una instrucción del assembly. En el caso de las instrucciones que usan 2 opcodes, el disassembler deberá ser capaz de reconocer esta secuencia. Los literales son también sencillos de reconocer, dada la estructura de los opcodes. Finalmente, a partir de un programa ubicado en la memoria de instrucciones, no es posible reconstruir el codigo assembly original, ya que se pierde la información de los labels.

c) Describa detalladamente un mecanismo para generar un archivo ejecutable en x86, a partir del resultado de un *assembler* para el computador básico. Describa claramente como trabaja cada una de las transformaciones intermedias. **(3 ptos.)**
Solución: El primer paso consiste en generar el assembly del computador básico usando el *disassembler* a partir del código ejecutable almacenado en la memoria de instrucciones. Luego, a partir de este assembly, se genera assembly x86, traduciendo cada instrucción a su simil en x86. Dado que el assembly corresponde a un computador de 16 bits, no hay problema en usar la siguiente conversión: el registro A se transforma en AL y el registro B en BL. Para poder utilizar las variables declaradas en la memoria de datos, se copia el contenido de esta en el código assembly x86, después de las instrucciones. De esta manera, cada vez que en el assembly del computador básico se referencia al contenido de la memoria de datos, se debe transformar esta dirección a la nueva ubicación, de acuerdo a lo expuesto anteriormente. Finalmente, el código assembly x86 se procesa con un *assemler* x86, para generar el código ejecutable.

Pregunta 2

a) Un computador x86 monoprocesador posee un pipeline de 5 etapas que se ejecutan en el siguiente orden:

   a) Fetch: Obtiene desde la memoria el opcode de la instrucción a ejecutar.

   b) Decode: Decodifica el opcode, enviando las señales correspondientes a cada componente.

   c) Read: Lee desde registros y memoria los datos requeridos para ejecutar la operación.

   d) Execute: Ejecuta la operación aritmética/lógica de la instrucción usando la ALU.

   e) Write: Almacena en registros o memoria el resultado de la operación aritmética/lógica.

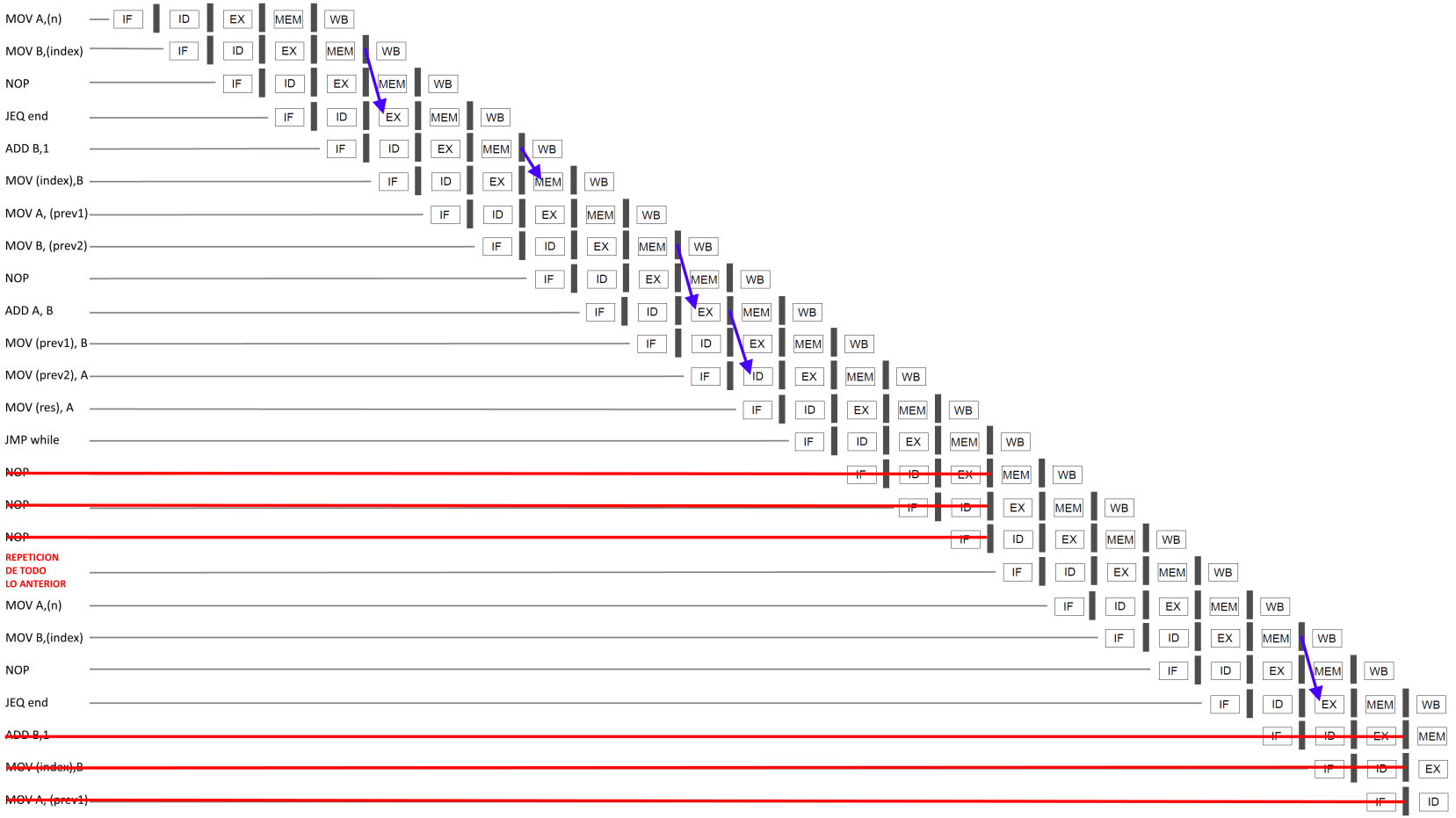
   Además de los mecanismos tradicionales para combatir hazards de datos y de control, el computador evita hazards estructurales de acceso a memoria entre las etapas Fetch, Read y Write, al utilizar una memoria RAM que permite realizar de manera simultánea tres solicitudes de distintas. A pesar de esto, es posible generar un hazard estructural de ejecución entre las etapas Read y Execute, cuando se procesa una instrucción del tipo MOV Reg1, [Reg2+offset]. ¿Cómo es posible evitar este hazard ? **(2 ptos.)**
   Solución: El hazard se genera al existir una operación artimética para calcular el direccionamiento. Esto implica que se utiliza la ALU en dos etapas. Para solucionar el hazard, se puede agregar un sumador en la etapa encargada de ir a la memoria, de manera que el direccionamiento sea calculado sin generar colisiones con otras etapas.

b) Determine el número de ciclos que se demora el siguiente código, detallando en un diagrama los estados del pipeline por instrucción. El pipeline tiene forwarding entre todas sus etapas, el manejo de stalling es por software (instrucción NOP) y predicción de salto asumiendo que no ocurre. Indique en el diagrama cuando ocurre forwarding, stalling y flushing. **(4 ptos.)**

```
DATA
  n     2
  index 0
  prev1 0
  prev2 1
  res   1

CODE
  while:
    MOV A, (n)
    MOV B, (index)
    JEQ end
    ADD B, 1
    MOV (index), B
    MOV A, (prev1)
    MOV B, (prev2)
    ADD A, B
    MOV (prev1), B
    MOV (prev2), A
    MOV (res), A
    JMP while
  end:
```

Solución: El programa toma 46 ciclos.


MOV A,(n)
MOV B,(index)
NOP
JEQ end
ADD B,1
MOV (index),B
MOV A, (prev1)
MOV B, (prev2)
NOP
ADD A, B
MOV (prev1), B
MOV (prev2), A
MOV (res), A
JMP while
NOP
NOP
NOP
REPETICION
DE TODO
LO ANTERIOR
MOV A,(n)
MOV B,(index)
NOP
JEQ end
ADD B,1
MOV (index),B
MOV A, (prev1)
IF
ID
EX
MEM
WB

Pregunta 3

a) ¿En qué se diferencian los tipos de paralelismo SIMD y SIMT? Complemente las diferencias con ejemplos para cada uno de los casos. **(1 pto.)**
Solución: SIMD (Single Instruction Multiple Data) consiste en utilizar un mismo programa (o función) para una serie de de datos. Un ejemplo de esto son las instrucciones SSE de la ISA x86. SIMT (Single Instruction Multiple Threads) consiste en utilizar un mismo programa sobre múltiples datos, pero utilizando múltiples threads. Un ejemplo de esto son las GPUs.

b) Asuma que cuenta con un computador x86 de 16 bits con soporte para instrucciones SSE de 64 bits. Cada uno de los 8 registros extra, XMM0 a XMM7, puede ser utilizado también como 4 registros independientes de 16 bits, *i.e.*, los subregistros de XMM0 son XMM0_0, XMM0_1, XMM0_2 y XMM0_3. Las instrucciones aritméticas que permiten utiliazar estos registros son MULPS Reg0, Reg1 y ADDPS Reg0, Reg1, mientras que la instrucción MOV es utilizada para las transferencias. Tomando todo esto en consideración, implemente en assembly x86 de 16 bits + SSE64, un programa que calcule el producto punto entre 2 vectores de enteros de 16 bits de largo arbitrario N. **(2 ptos.)**
Solución: Se asume que los vectores están almacenados en A y B.

```
    JMP    main
    N      dw    ?
    A      dw    ?,?,?,...
    B      dw    ?,?,?,...
    res    dw    0
  main:
    MOV    SI, 4
    MOV    DI, 8
    MOV    XMM2_0, 0
    MOV    XMM2_1, 0
    MOV    XMM2_2, 0
    MOV    XMM2_3, 0
    LEA    BX, A
    LEA    CX, B
  for:
    CMP    SI, N
    JG    resto
    MOV    XMM0_0, [BX+DI-8]
    MOV    XMM0_1, [BX+DI-6]
    MOV    XMM0_2, [BX+DI-4]
    MOV    XMM0_3, [BX+DI-2]
    MOV    XMM1_0, [CX+DI-8]
    MOV    XMM1_1, [CX+DI-6]
    MOV    XMM1_2, [CX+DI-4]
    MOV    XMM1_3, [CX+DI-2]
    MULPS  XMM0, XMM1
    ADDPS  XMM2, XMM0
    ADD    SI, 4
    ADD    DI, 8
    JMP    for
  resto:
    SUB    SI, 4
    SUB    DI, 8
  for_resto
    CMP    SI, N
    JEQ    end
```

```
    MOV    AX, [BX+DI]
    MUL    [CX+DI]
    ADD    res, AX
    ADD    SI, 1
    ADD    DI, 2
    JMP    for_resto
  end
    ADD    res, XMM2_0
    ADD    res, XMM2_1
    ADD    res, XMM2_2
    ADD    res, XMM2_3
```

c) Se desea programar en una GPU un algoritmo que procese una imagen en escala de grises, guardada por filas, sustituyendo el valor de cada pixel por el promedio de los valores de sus 8 vecinos.

I. ¿Cuál sería la tarea de cada uno de los threads de este algoritmo? **(1 pto.)**
Solución: Cada thread se encargaría de calcular el valor de un pixel, *i.e.*, calcular el promedio de los 8 vecinos.

II. ¿Cómo definiría un *warp* para este algoritmo? **(1 pto.)**
Solución: Dado que la matriz está guardada por filas, un petición de memoria no alcanza a traer los datos de los 8 vecinos. En promedio, se necesitarán 3 peticiones distintas para los 8 vecinos (datos en 3 filas distintas). Por lo tanto, tun warp debería definirse como un grupo de threads que procesen pixeles en la misma fila y que las peticiones de lecturas del primer thread del warp traigan también a la caché L1 todos los datos que necesitan los otros threads del warp

III. ¿Cómo se podrían evitar los problemas de divergencia de código en este algoritmo? **(1 pto.)**
Solución: Una posibilidad para evitar la divergencia (control de flujo) es tener 5 funciones distintas para procesar los pixeles de los 4 bordes y los del que no estan en los bordes.

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



IIC2343 – Arquitectura de Computadores (I/2016)

Solución Examen

Respuestas sin desarrollo o justificación no tendrán puntaje.

Pregunta 1

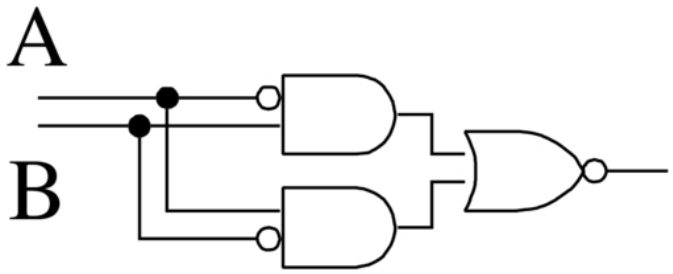
a) Explique por qué el número $2^{50} + 5$ no puede representarse de manera exacta usando el tipo de dato `float` de 32 bits. **(1 pto.)**
Solución: El tipo de datos `float` de 32 bits sólo contempla 23 bits para el significante (normalizado), por lo que para cualquier número mayor o igual a 2^{25}, la diferencia entre números consecutivos será mayor a 1. De esta manera, la diferencia entre 2^{50} y el siguiente número representable en `float` es de $2^{50-23} = 2^{27}$. Por lo tanto, el número $2^{50} + 5$ no puede representarse como `float`.

b) ¿Cuál es la función del registro PTBR? **(1 pto.)**
Solución: El registro PTBR almacena la dirección de inicio de la tabla de paginas del proceso en ejecución.

c) En la arquitectura x86, ¿por qué el South Bridge se conecta al North Bridge y no directamente a la CPU? **(1 pto.)**
Solución: Con el fin de no hacer más lenta la comunicación entre los dispositivos rápidos y la memoria, ya que el rango de velocidades de los dispositivos que se conectan al South Bridge es amplio y en general más lentos que los conectados al North Bridge.

d) Indique qué es el modo supervisor de una CPU, para qué sirve y como y cuando se activa. **(1 pto.)**
Solución: El modo supervisor es un mecanismo para restringir el acceso a elementos que sólo deben estar bajo control del sistema operativos, como las tablas de páginas o los dispositivos de I/O. Se activa cada vez que interviene el sistema operativos, ya sea por una interrupción de software, hardware o excepción.

e) Utilizando sólo compuertas AND, OR y NOT, implemente un circuito que calcule **A** NXOR **B**, donde NXOR = negación del XOR. **(1 pto.)**
Solución:

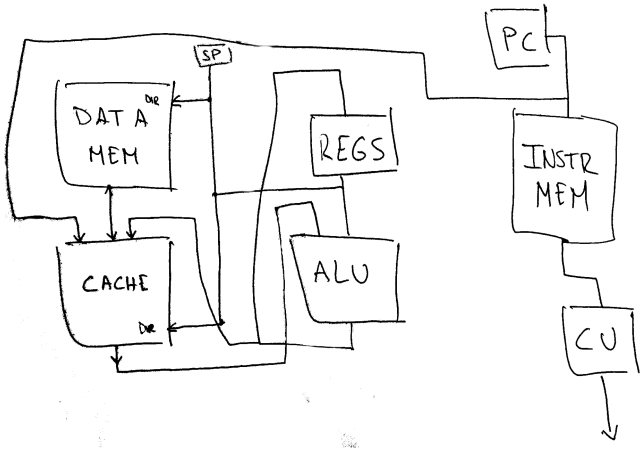


f) Indique qué son las convenciones de llamada y por qué son fundamentales para el correcto funcionamiento de los programas. **(1 pto.)**
Solución: Las convenciones de llamada son reglas que describen de manera estricta y sin ambigüedad la manera en que deben llamarse las subrutinas, tomando en cuanta el paso de parámetros, uso del stack y valores de retorno entre otros. Sin la existencia de estas reglas, códigos de distintos origen no podrían comunicarse a través del llamado a subrutinas, ya que no habría claridad con respecto a como hacerlo.

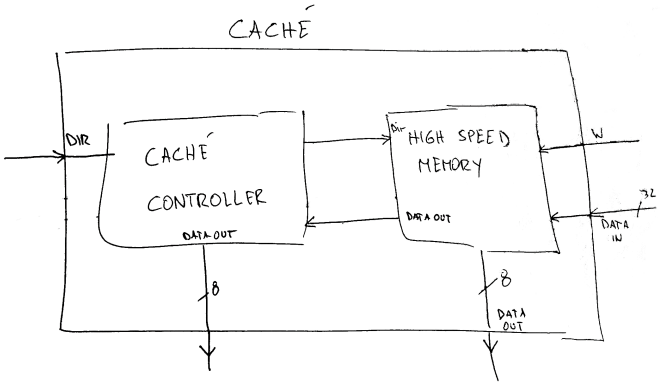
Pregunta 2

En esta pregunta deberá aplicar varios tópicos cubiertos en el curso para diseñar una memoria caché de mapeo directo para la memoria de datos del computador básico. Asuma que esta memoria caché tiene 8 líneas de 4 palabras cada una.

a) Haga un diagrama de alto nivel de la microarquitectura del computador básico, incluyendo la memoria caché. Tenga especial cuidado con el tamaño de los buses. **(1 pto.)**
Solución:



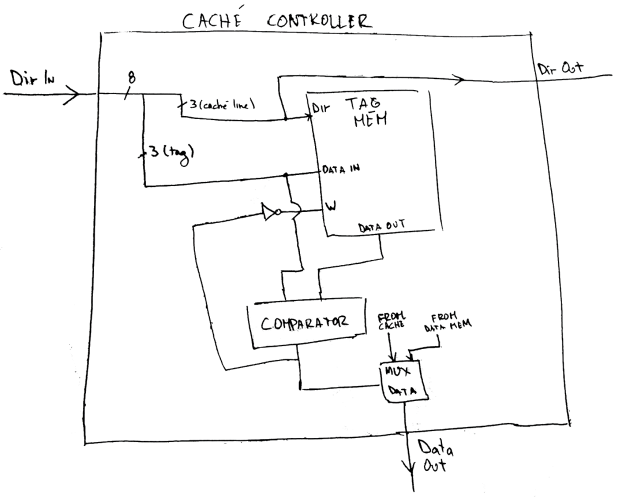
b) Haga un diagrama de alto nivel de la microarquitectura de la memoria caché. **(1 pto.)**
Solución: Se asume que la caché utiliza política write-through.



c) Diseñe la microarquitectura del controlador de la memoria caché. Describa detalladamente el funcionamiento de esta:

I. Diseñe los elementos que intervienen en la copia de datos desde la memoria caché hacia la CPU, *i.e.*, todos los elementos que intervienen cuando hay un *hit*. **(2 ptos.)**

II. Diseñe los elementos que intervienen en la copia de datos desde la memoria de datos hacia la memoria caché, *i.e.*, todo lo que ocurre luego de un *miss*. **(2 ptos.)**
Solución a) y b):



Pregunta 3

a) Describa un esquema para manejar múltiples procesos utilizando sólo una (1) tabla de páginas. **(2 ptos.)**
Solución: Una posibilidad es unir todas las tablas de páginas, agregando a cada entrada una nueva columna con el identificador del proceso al cuál pertenece. Al momento de hacer una búsqueda en esta nueva tabla, sólo se realizan sobre las entradas que pertenecen al proceso en ejecución.

b) En una memoria caché *N-way associative*, describa una política de sustitución basada en LRU (Pseudo-LRU), que utilice sólo N bits por conjunto para seleccionar la línea a sustituir. **(2 ptos.)**
Solucion: Una manera simple de aproximar LRU es teniendo un bit por línea dentro de un conjunto. Este bit parte en 0 y es modificado a cuando la línea es accedida. Si todas las líneas dentro de un conjunto tienen su bit en 1, se resetean todas dejándolas en 0. Para decidir cuál es la línea que se sustituirá, se selecciona la línea con el índice más bajo que tiene su bit en 0.

c) Describa paso a paso la interrupción de software que debe ejecutar un sistema operativo cuando recibe una petición para leer de disco (considere el uso de DMA). **(2 ptos.)**
Solución:

a) (opcional) desactivar interrupciones de hardware.

b) extraer parámetros relevantes para la transferencia (dirección de inicio en disco, dirección de destino en memoria, cantidad de palabras a leer).

c) cargar en el controlador de DMA los parámetros.

d) enviar al controlador de DMA el comando para iniciar la transferencia.

e) dormir al proceso que realizó la petición hasta que el DMA indique que ha terminado la transferencia

f) (opcional) activar interrupciones de hardware.

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



IIC2343 – Arquitectura de Computadores (II/2016)

Solución Examen

Pregunta 1

a) Modifique la arquitectura del computador básico, para que funcione con lógica ternaria en vez de binaria. Más específicamente, modifique los tamaños de los elementos (buses, registros, señales de control, etc) de modo que el nuevo computador tenga una capacidad similar a la versión binaria. Asuma que existen todos los componentes vistos en clases en versión ternaria. **Nota**: No es válido utilizar los valores ternarios como si fueran binarios. **(3 ptos.)**
Solución: Si bien la principal modificación era el redimencionamiento de los buses, se esperaba mencionaran que cambia el tamaño de los registros y algunas señales de control. La fórmula usada es $\lceil log_3(2^n) \rceil$ para un bus binario de n bits. En general no debe perderse rango de números posibles, ganar rango es indiferente. Se aceptó para los buses de 8 bits tamaños de 5 o 6 trits, sin embargo, lo más correcto es 6 trits. Deben reducir la cantidad de señales de control de la ALU a 2 trits, los muxer de 3 elementos o menos a 1 trit, u muxer de 4 elementos a 2 trits. NO deben redimensionar el reg. status.

b) Agregue a la microarquitectura del computador básico los elementos necesarios para soportar multiprogramación, tanto desde el punto de vista de memoria como de CPU. **(3 ptos.)**
Solución: e debe agregar una MMU, TLB, PTBR, Supervisor mode reg., Soporte para I/O, RTC.

Debe añadirse algún mecanismo para respaldar el estado de los registros para cambios de contexto, donde lo más sencillo es respaldar a memoria.

Se espera que mencionen las conexiones de los elementos, donde el Supervisor mode Reg. se coencta a la CU y con un enabler a las señales de write para la TLB y el PTBR, adicionalmente esta señal debe ser leida por la MMU para activar o desactivar la traducción de direcciones de memoria.

La MMU se debe conectar antes de la memoria principal, interviniendo principalmente el bus de Address. La MMU se conecta con la TLB y el PTBR. Idealmente deberían haber diseñado la MMU, como un computador similar al PC Básico y que corre un programa simple para realizar traducciones.

Para el soporte I/O, se puede añadir un address decoder al bus de direcciones de memoria, y una conexión directa a algún mux de entrada al PC. Para interrupciones, por ejemplo del RTC, es posible conectar un bit de interrupción a la CU, decir que esta ahora será de tipo microcode y que detonará una serie de operaciones.

Note que el PCB es un problema de software y por tanto, dando un mecanismo válido para respaldar el estado de la CPU, no se considera para esta pregunta. La conexión de un HDD para realizar swapping no es obligatoria y por tanto no se considera.

Pregunta 2

a) En un computador con múltiples niveles de memoria caché, ¿es necesario que todos compartan las mismas características (tamaños totales, tamaño de línea, correspondencia, sustitución, división datos/instrucciones, etc.)? En cualquiera de los dos casos, indique y justifique cuáles podrían ser valores para estas características. ¿Cambiarían estos criterios si el computador tiene más de un procesador? ¿Qué partes de la caché compartiría? **(3 ptos.)**
Solución: No, no es necesario que en un computador con múltiples niveles de memoria caché se compartan características:

- Tamaños de la memoria: Conviene que la capacidad de cada nivel aumente mientras aumente el nivel para aprovechar de mejor manera la localidad espacial.
- Tamaño de línea: Conviene que se comparta el tamaño de línea entre niveles de caché para que sea más sencillo el traspaso de información entre niveles.
- Correspondencia y sustitución: Da igual que correspondencia y sustitución utilicen los distintos niveles de memoria caché.
- División datos/instrucciones: Conviene que la memoria más cercana al procesador (L1) sea split y el resto de los niveles unified.

Si hay más de un procesador no cambiarían las características, pero si convendría que cada procesador tuviera al menos su propio nivel L1 y que se comparta al menos uno de los niveles superiores de memoria entre procesadores.

b) Un sistema operativo implementa un protocolo de seguridad de 2 etapas. En la primera, los **programas** son analizados por un antivirus/antimalware, mientras que en la segunda, cada una de las instrucciones de los **procesos** es analizada antes de ser ejecutada por el procesador. Indique como implementaría la segunda etapa en un computador con soporte para todos los esquemas de I/O vistos en clases. **(3 ptos.)**
Solución: Para la segunda etapa basta con el siguiente procedimiento:

1. Se guarda el valor de la instrucción en la posición PC + 1.
2. Se genera una interrupción de hardware que le entrega el control al sistema operativo.
3. El sistema operativo realiza todas las acciones necesarias para analizar la instrucción guardada.
4. Si el análisis arroja que la instrucción no es maliciosa, se devuelve el control al programa actual y se ejecuta la instrucción analizada. En caso contrario, se toma alguna medida que evite la ejecución del programa.

Pregunta 3

a) Las instrucciones en una ISA RISC, sin tomar en cuenta su etapa *fetch*, toman un ciclo de clock. Asumiendo que puede realizar cambios a la microarquitectura de un computador x86, indique como podría transformar en tiempo de ejecución cada instrucción de la ISA x86 a una serie de instrucciones RISC. Relacione esto con la implementación de un pipeline. **(3 ptos.)**
Solución: Una posible solución es incluir la capacidad por hardware, posiblemente dentro de la Control Unit, que registre una instrucción compleja como varias instrucciones sencillas del tipo mover de registros a memoria o viceversa y operar registros y regule el paso de estas, por ejemplo, deteniendo el flujo de la lectura de instrucciones mientras prosigue con . Esta separación en sub-instrucciones se asemeja a la idea del *pipeline*.

No es una opción hacer ésto desde el *assembler*, pues se pide expresamente que sea una implementación en *hardware*. Tampoco lo es alterar los tiempos de ejecución de las instrucciones más lentas/rápidas para equipararlas con las demás, pues sigue teniéndose el mismo esquema a fin de cuentas.

En general, los puntajes se asignan de la siguiente forma:

- Identificar por qué las instrucciones toman más tiempo (i.e acceder varias veces por memoria), *[0.5 pts]* ó
- identificar a qué tipo de instrucciones se subdividen las complejas (ver solución propuesta). *[0.5 pts]*
- Menciones a la solución propuesta de forma incompleta (i.e dejar muy en el aire cómo se logra), *[0.75 pts]* ó
- responder correctamente, sea lo propuesto u otra solución válida. *[1.5-2 pts, según se asignó el parcial de 0.5]*
- Relacionar con el *pipeline* correctamente. *[1 pts, y 0.5 si no está muy bien]*

b) Un fabricante de dispositivos móviles de reproducción de audio, está intentando realizar una importante reducción de costo en sus equipos, mediante la eliminación del soporte de hardware de números de punto flotante, sólo dejando soporte nativo para números enteros. Tomando esto en consideración, indique y diagrame cómo se puede implementar la multiplicación de números de punto flotante utilizando sólo elementos que trabajen con números enteros (sumadores, multiplicadores, etc.). Ponga especial cuidado en los tamaños de los números, para no perder precisión ni rango. **(3 ptos.)**
Solución: La respuesta es evidente: diseñar un componente que realice la operación de la multiplicación en números de punto flotante, es decir, implemente en *hardware* el algoritmo con el que se realiza esta operación:

- Manejo del signo *[0.5 pts]*
- Suma de los exponentes (sin olvidar que $exp_{registrado} = exp_{real} + 127$ en el caso de 32 bits) *[1 pts]*
- Multiplicación de las mantisas (considerando que no se registra el primer 1) *[1.5 pts]*

Por supuesto, la respuesta varía si es que se define otro formato de punto flotante, en cuyo caso también debe estar explicitado.

Arquitectura de Computadores – IIC2343

Examen

21 noviembre 2017

1. Un lenguaje de programación de alto nivel para multiprocesadores (varias CPUs que comparten una memoria común a través de un bus) ofrece la siguiente instrucción indivisible **exchange**, definida como el intercambio indivisible de los valores de dos variables (indivisible o ininterrumpible: si un proceso inicia la ejecución de **exchange**, entonces nada lo detiene o interrumpe hasta que haya terminado de ejecutarla, es decir, hasta que haya intercambiado los valores de las variables):

```
exchange(a, b):
   temp = a
   a = b
   b = temp
```

a) Desarrolla una solución al problema de la *sección crítica* para dos procesos usando **exchange**. Recuerda: la sección crítica es una secuencia de una o más instrucciones (no nos importa cuáles) que puede ejecutar un proceso, tal que si el proceso está ejecutando su sección crítica, entonces ningún otro proceso puede estar ejecutando su propia sección crítica al mismo tiempo (para mantener la corrección de los datos compartidos).

Respuesta. La solución clásica; puede haber otras:

```
C = 1; local1 = 0; local2 = 0
```

process P1:	process P2:
`while (true):`	`while (true):`
`  while (local1 == 0):`	`  while (local2 == 0):`
`  exchange(C, local1)`	`  exchange(C, local2)`
sección crítica	*sección crítica*
`exchange(C, local1)`	`exchange(C, local2)`

b) Da argumentos lógicos que aseguren que tu solución cumple las propiedades de i) *exclusión mutua* (a lo más un proceso puede estar ejecutando su sección crítica al mismo tiempo); y ii) *ausencia de deadlocks* (en ningún momento ocurre que un proceso está esperando que el otro haga algo y al mismo tiempo el segundo proceso está esperando que el primero haga algo).

Respuesta.

i) Se satisface exclusión mutua. La única forma en que el proceso `Pi` puede entrar a su sección crítica es que su variable `locali` valga 1; pero la única forma en que esa variable puede llegar a valer 1 es que adquiera ese valor al ejecutarse la instrucción `exchange(C, locali)` dentro del `while (locali == 0)`; y, si esto ocurre, entonces la variable `C` queda con el valor 0. Como la instrucción `exchange` es indivisible, aunque ambos procesos estén ejecutando el `while` "al mismo tiempo", solo uno va a tener éxito en cambiar su `locali` a 1 y dejar `C` en 0, con lo cual el otro proceso no va a poder poner su `locali` en 1. Además, la única forma en que `C` vuelva a valer 1 es que el proceso que sale de su sección crítica ejecute la instrucción `exchange` nuevamente.

ii) Los procesos esperan a que algo pase únicamente cuando ejecutan el `while (locali == 0)` (la sección crítica, por definición, son sentencias que se deben ejecutar bajo exclusión mutua, por lo que la espera ocurre antes, justamente para garantizar la exclusión mutua; y el último `exchange`, al salir de la sección crítica, claramente no depende de ninguna condición para su ejecución). Así, para que ambos procesos estén esperando, necesariamente deben estar ejecutando **`while (locali == 0)`**; pero la única forma en que ambos **`locali`** sean permanentemente 0 es que este sea el valor que tiene la variable **`C`** cada vez que se ejecuta el **`exchange`** (supeditado al **`while`**), y la única forma en que **`C`** valga 0 es que al menos una de las **`locali`** valga 1 (*), contradiciendo la suposición de que ambos **`locali`** son permanentemente 0.

(*) Esto es cierto al comienzo, por la asignación inicial a **`C`**, **`local1`** y **`local2`**; y es cierto cada vez que un proceso termina de ejecutar su sección crítica, instante en el cual su variable **`locali`** es todavía 1, y ejecuta **`exchange`**.

2. En un multiprocesador (varias CPUs que comparten una memoria común a través de un bus), cada CPU puede tener su propia *memoria caché* para así evitar tener que recurrir frecuentemente a la memoria principal a través del bus. Sin embargo, esto normalmente da origen al *problema de coherencia de cachés*. Para solucionar estos problemas, los controladores de cachés son diseñados de manera que "observen" (*snoop*) las solicitudes que pasan por el bus (y que fueron hechas por alguna otra caché) y que hagan algo en ciertos casos. El conjunto de reglas que define qué hacer y cuándo se llama *protocolo de consistencia de cachés*.

a) Considera el protocolo *write through* estudiado en clase, cuya esencia es que todas las operaciones de escritura resultan en que la palabra que está siendo escrita en la caché también es escrita en la memoria para mantener la memoria actualizada todo el tiempo. Si el controlador de caché solo pudiera observar las líneas de dirección del bus, y no las de datos, ¿se vería el protocolo *write through* afectado por esta situación? Explica.

Respuesta.

No. En el protocolo *write through* básico, cuando un controlador de caché observa que una dirección de memoria está siendo escrita por otra CPU, chequea su propio caché para ver si tiene esa misma dirección, y, si la tiene, entonces invalida la línea correspondiente, pero no la actualiza (hasta que su propia CPU lea esa misma dirección más adelante, en cuyo caso el controlador va a buscar la línea a la memoria). Por lo tanto, para que el protocolo funcione correctamente, lo que importa es que el controlador de caché sepa cuál dirección está siendo escrita, pero no es necesario que sepa con qué valor.

Nota. La respuesta podría ser sí, si se estuviera hablando de la versión del protocolo en que el controlador actualiza la línea de su caché en lugar de invalidarla; en este caso, el protocolo se vería afectado, ya que tendría que ser modificado para funcionar como en la versión básica.

b) En los protocolos de tipo *write-back* no todas las escrituras van directamente a la memoria: cuando una línea de la caché es modificada, se pone un bit de la caché en 1 indicando que la línea de la caché está correcta pero la memoria no; finalmente, la línea es escrita en la memoria, pero posiblemente después de sufrir varias escrituras. El protocolo *MESI* estudiado en clase define cuatro estados para cada línea de la caché: *modified*, *exclusive*, *shared* (compartido), *invalid*. Si solo pudiéramos tener tres estados, ¿cuáles estados podrían ser eliminados (solo uno a la vez) y cuáles serían las consecuencias en cada caso?

Respuesta.

Eliminar el estado *invalid* es una mala idea: exigiría mantener todas las líneas de todas las cachés permanentemente actualizadas.

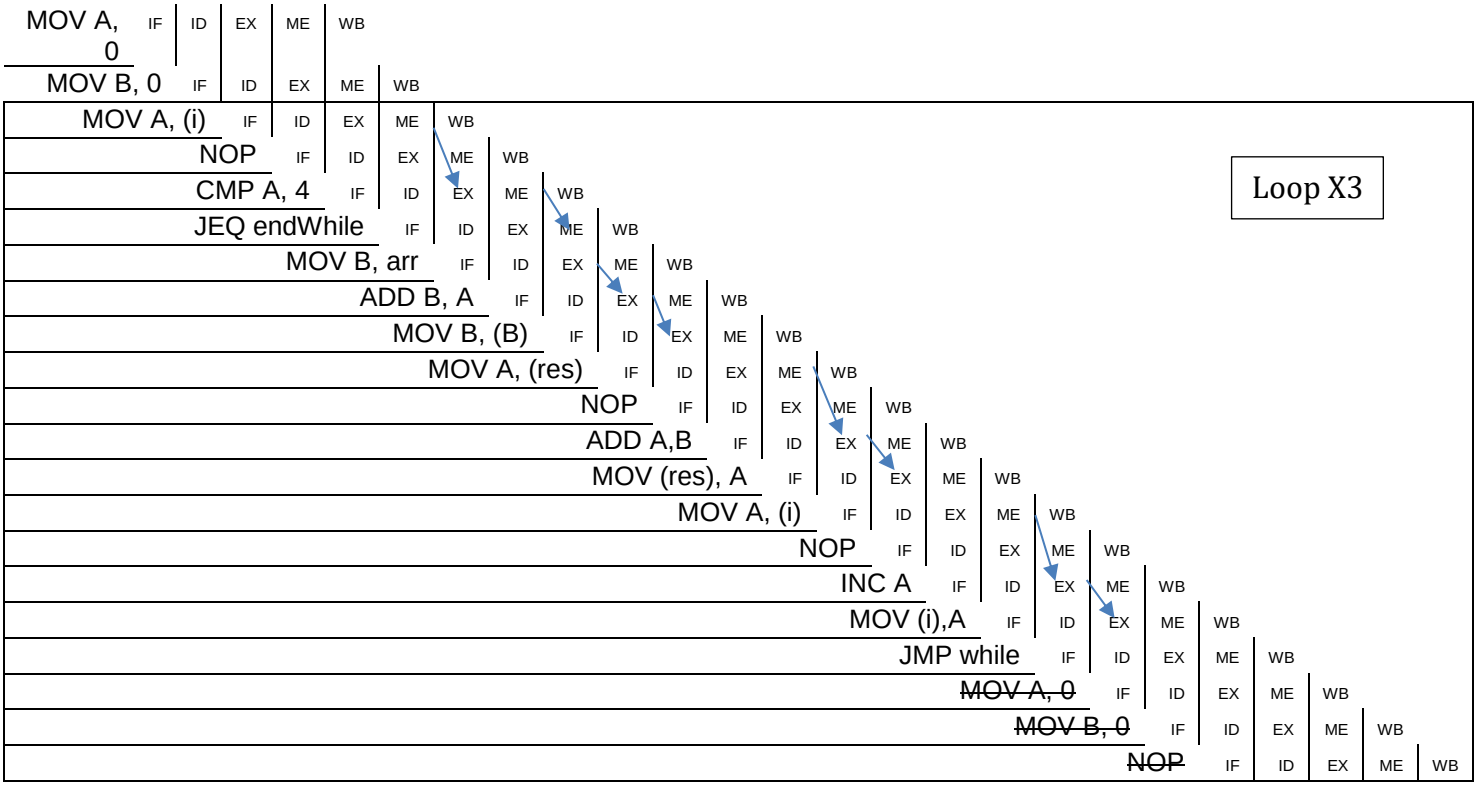
En cambio, los estados *exclusive* y *shared* pueden combinarse en uno, con comportamiento *shared*: la línea con el valor más reciente está en una o más cachés y la memoria está actualizada; si cualquiera de las CPUs involucradas escribe (cambia un valor de) la línea en su caché, entonces le avisa a las otras (aunque no haya "otras") que invaliden sus líneas, y pasa a estado *modified*, es decir, solo mi línea es válida e incluso la memoria es inválida.

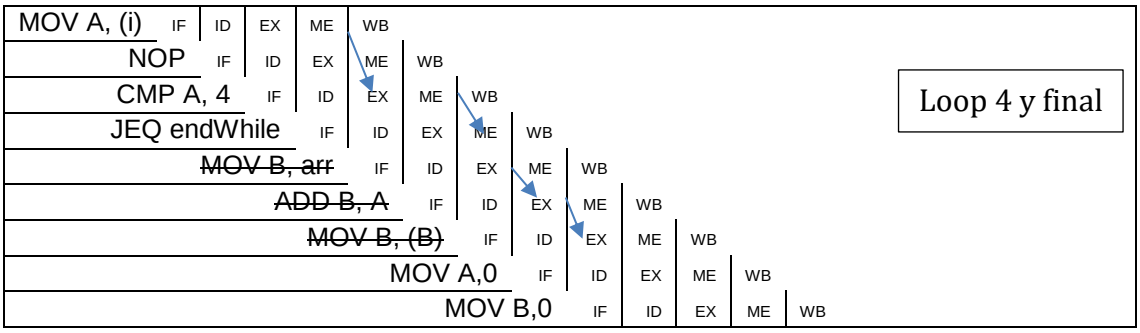
El estado *modified* puede eliminarse —haciendo que la memoria esté siempre actualizada— pero el costo es que el protocolo deja de ser *write-back* y pasa a ser *write through*.

3. Para el siguiente programa en *assembly*, rellena el diagrama del pipeline del computador básico e indica todos los *forwards* necesarios durante la ejecución. Considera que el manejo de *stalling* se realiza mediante NOP en tiempo de compilación, la predicción de saltos es (estáticamente) que no salta, y hay *forwarding* entre todas las etapas. Indica claramente todos los *stalls*, *flushs* y *forwards*.

```
DATA                          CODE
        res   0                       MOV A,0
        arr   1                       MOV B,0
              2                 while:
              3                       MOV A, (i)
              4                       CMP A, 4
        i     0                       JEQ endwhile
                                      MOV B, arr
                                      ADD B, A
                                      MOV B, (B)
                                      MOV A, (res)
                                      ADD A, B
                                      MOV (res), A
                                      MOV A, (i)
                                      INC A
                                      MOV (i), A
                                      JMP while
                                endwhile:
                                      MOV A, 0
                                      MOV B, 0
```

Respuesta.





4. Tú tienes un escáner que puede digitalizar imágenes de 256x256 pixeles en colores. Para ello, el escáner produce para cada pixel tres números entre 0 y 255 (correspondientes a las intensidades de los colores rojo, verde y azul) que coloca en tres *buffers* de datos (uno para cada color). El escáner se conecta mediante *memory-mapped I/O* al computador. El escáner cuenta con un digitalizador que digitaliza una línea de la imagen a la vez; la posición del digitalizador sobre la imagen es controlada por un motor, que recibe como parámetro la línea que debe digitalizar y posiciona al digitalizador donde corresponda; el digitalizador envía de forma contínua lo que ve.

   a) [0.5 pts] Diseña la tabla de comandos y direcciones que utilizaría el escáner.

   Respuesta.

   No hay una tabla dada, se espera una donde asignen direcciones de memoria para el registro de estado del escáner, el registro de comandos y el buffer de datos. Deberían tener cuidado con el vector de interrupciones respecto de las direcciones de memoria asignadas. Para el registro de comandos, deberían indicar solo un comando idle y un comando escanear, al igual que en estado un estado idle y un estado escaneando.

   b) [1 pt] Para el escáner diseñado, escribe una ISR que lo controle. Si haces supuestos, señálalos claramente.

   Respuesta.

   No se ahondará en detalle, se espera que hagan supuestos sobre el cómo pasar parámetros solamente y uso de la DMA para copias de memoria, en alto nivel debería tener un comando de inicialización y otro de escaneo que reciba como parámetro la dirección base donde dejar el contenido del buffer. Es correcto que en este paso cambien de un buffer por color a un buffer unificado con los 3 colores para cada pixel, pero deben detallarlo.

   Se debe vigilar que especifique de dónde adquieren los parámetros y cuales son, descontar si no lo hacen. Deben usar IRET o de lo contrario descontar 0.3 pts

   c) [1.5 pts] Escribe un programa que sea capaz de escribir en disco duro de un computador una imagen desde el escáner. El computador posee una DMA memoria a memoria, con cuatro registros: un registro de comandos, un registro de dirección base, un registro de dirección destino, y un registro de tamaño de copia; la DMA es controlada por la ISR 15h. Por su lado, el HDD tiene un buffer de datos de 2 MB y tres registros: un registro de comandos para saber si debe leer o escribir, un registro para dirección base de HDD, y un registro para tamaño de copia; el HDD es controlado por la ISR16h. En el HDD la imagen debe tener de forma consecutiva los valores de los colores rojo, verde y azul de cada pixel.

   Respuesta.

0.5 pts de descuento si uso no es consistente con lo hecho anteriormente pero se entiende. Se espera que llamen a los traps correspondientes en orden y esperen al DMA para escribir en HDD, luego esperen al HDD. Descontar 0.5 si no reordenan la data.

d) [3 pts] Diseña el microcontrolador del escáner mediante un diagrama y especifica en pseudocódigo qué debe hacer tu controlador. Sé explícito con tus supuestos e incluye los comentarios necesarios para que se entienda cómo funciona tu diseño.

Respuesta.

1.5 pts por diagrama Y/O explicación detallada. 1.5pts por pseudocódigo donde la idea es que usan un contador para ir de 0 a 256 y poner el digitalizador, y con eso listo (no hay descuento por asumirlo instantaneo) leen el digitalizador como si fuera un registro grande o mejor 3 registros (uno por color) y lo ponen en el registro de salida, al terminar marcan el registro de estado y esperan hasta que les llegue instrucción de digitalizar nuevamente. Otras soluciones que hagan sentido son válidas.

Se espera que en diagrama presenten algo parecido al computador básico.

Elige dos preguntas entre las preguntas 5 a 7:

5. Con respecto a la representación de números enteros:

a) Si en vez de bits (dígitos binarios, o en base 2), tuviéramos dígitos en base 4, ¿cuál es el rango de valores que podría ser representado usando *N* de estos dígitos? Considera que queremos representar tanto números positivos como negativos.

Respuesta.

$$[-\frac{4^n}{2}, \frac{4^n}{2} - 1]$$

Puntaje pregunta completa 2 puntos.
Por cada error se baja 0.2. Si tiene más de 4 errores o está todo malo se pone 0 puntos.

b) Si ahora tuviéramos *N* dígitos en base 3, ¿cuál es el rango de valores que podríamos representar? ¿Qué diferencia, además de las obvias, existe con el caso a)?

Respuesta.

$$[-\left\lfloor\frac{3^n}{2}\right\rfloor, \left\lfloor\frac{3^n}{2}\right\rfloor]$$

La principal diferencia es que en base 4, el primer digito me indica el signo ya que si es 0 o 1 entonces es positivo o 0 y si es 2 o 3 entonces es negativo. Mientras que acá debemos fijarnos en los primeros 2 dígitos para poder ver si es positivo o negativo.

1 Punto por el rango (se baja 0.2 por cada error en el rango, si tiene 4 errores o esta malo se pone 0).

1 Punto por la explicación.

c) Para cada una de las representaciones anteriores, a) y b), explica cómo se puede saber si un número es positivo o negativo a partir de la representación utilizada.

Respuesta.

Para saber si un número es negativo o positivo debemos primero fijarnos en la base.

- Si la base es par, si el primer digito es menor a $\left(\frac{b}{2}\right)$ entonces es positivo, en caso contrario negativo.
- Si la base es impar, debemos todos los que sean menores o iguales a $\left\lfloor\frac{b^n}{2}\right\rfloor$ son positivos, negativos en otro caso. Otra manera es sacar el digito central d que es $\left\lfloor\frac{b}{2}\right\rfloor$ y ver si el número es menor o igual al numero $dddd \ldots d$ que es el número conformado por n veces el digito central d. Si es menor o igual entonces es positivo, negativo en otro caso.

1 Punto por la explicación de los pares (o explicar en base 4).

1 Punto por la explicación de los impares (o explicar en base 3). No se baja si no puso el caso de 1 solo digito.

6. Describe, en función de los registros, unidades funcionales, señales de control y conexiones involucradas, la ejecución de la instrucción **`LOAD reg1, offset(reg2)`** (es decir, con direccionamientos por registro e indexado):

a) Describe, en orden, los 5 pasos principales en que se descompone la ejecución de la instrucción.

b) Dibuja el diagrama de los registros, unidades funcionales, señales de control y conexiones involucradas, de manera que quede clara la correspondencia con la descripción dada en a).

c) Explica qué diferencias principales habría si la instrucción fuera **`STORE reg1, offset(reg2)`**.

Respuestas.

a)

1) Se lee una instrucción desde la memoria de instrucciones, y se incrementa el PC (o *instruction pointer*).

2) Se lee el valor del registro 2.

3) La ALU calcula la suma del valor obtenbido del registro 2 más el *offset* (especificado en la propia instrucción).

4) El resultado de esta suma se usa como la dirección en la memoria de datos.

5) El dato así obtenido desde la memoria de datos se escribe en el registro 1.

b)

c) La unidad de control debería indicar que se trata de un *write* en lugar de un *read*; el valor del registro 1 sería el valor que se almacenaría en la memoria de datos; y la operación final de escribir en el registro 1 no ocurriría.

7. Responde:

a) ¿Por qué se prefiere usar complemento de 2 para representar números enteros negativos, en lugar de otras representaciones tales como complemento de 1 o bit de signo?

Respuesta.

Se prefiere el complemento de 2 porque es barato de implementar, además, tiene la propiedad de que se puede operar de forma normal sin consideraciones especiales, como es el caso de bit de signo.

b) ¿Cuál es el rol de la unidad de control del computador básico?

Respuesta.

La unidad de control es la encargada de orquestar el flujo de datos dentro de los componentes que conforman el computador con el fin de ejecutar las instrucciones que se le pasan. Incluye lo que es el decoding de las instrucciones desde opcodes a palabras de control completas. Además, en caso de que se tengan multiples modos de operación en el CPU es el encargado de lidiar con esa complejidad, por ejemplo, modos virtual y real de x86.

c) ¿Qué ventajas y desventajas tiene que la comunicación con los dispositivos de I/O sea *memory-mapped*, en lugar de usar el concepto de *ports* ?

Respuesta.

La ventaja es que no requiere introducir nuevo hardware para manejar las solicitudes de IO además del Address-Decoder y permite que las DMA trabajen sin modificaciones con ellos, la desventaja es que ocupa direcciones de memoria que ya no serán usables para los datos de los programas y eso es poco deseable. 0.5 ventaja/desventaja. Lo importante es que haga sentido.

d) ¿Cuáles son los principios —y en qué consisten en la práctica— en los que se basa el uso de memoria caché?

Respuesta.

Localidad espacial: Un programa tiende a utilizar memoria que está contigua.

Localidad temporal: Un programa que accede a una variable tiene una alta probabilidad de volver a accederla en el future.

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



IIC2343 – Arquitectura de Computadores (II/2018)

Examen

El examen se compone de 5 preguntas. Debe responder cada una en una hoja separada, aunque puede utilizar más de una hoja para responder a una pregunta. Si no responde a una pregunta, debe entregar una hoja en blanco para ella. Recuerde poner su nombre en todas las hojas de respuesta.
Al momento de entregar, **evite doblar bordes o corchetear las hojas**, si no que sencillamente indique cuántas hojas son para esa pregunta y qué hoja es cada una en caso de utilizar más de una, procurando que esté lo más lisa posible. Todas las hojas serán escaneadas y las recorrecciones del examen se harán sobre lo escaneado.

Pregunta 1

Considere el siguiente programa:

```
a = 10
b = 20
res = 0
while a > 0:
    res = res+b
    a = a-1
```

Asumiendo que las variables `a`, `b` y `res` almacenarán números de a lo más 8 bits, responda las siguientes preguntas:

a) Construya un circuito secuencial que implemente el programa. **(3 ptos.)**
Respuesta:
Una solución aceptada con todo el puntaje es que indiquen que usarán el computador básico con el programa escrito en assembler. No es necesario que escriban el código.
Otra es que implementen un circuito ad-hoc que debiese implementar dos registros para los inputs y un acumulador para el output y una ALU.

b) Construya un circuito combinacional que implemente el programa **(3 ptos.)**
Respuesta:
Acá se admite la opción de que expresen parcialmente que usarán algún decoder para señalar 2^8 pines donde cada uno estará asociado a una celda de mux que selecciona con el pin o un operando o un 0 y eso conectado a un sumador. Al conectarlos en serie se obtiene lo pedido.
Otra opción es que implementen los sumadores de forma escalonada usando los bits de un lado para saber si suman en esa zona el otro operando o rellenan con 0, y los bits que hagan falta se rellena con 0.

Pregunta 2

Considere un computador Von Neumann, al que se le desea conectar un acelerador matemático que permita soportar de manera nativa operaciones de punto flotante.

a) Diseñe la microarquitectura del coprocesador, de manera que contenga al menos 8 registros de 80-bits, y soporte para operaciones aritméticas. Puede utilizar cualquier componente visto en clases. **(2 ptos.)**
Respuesta:
En general acá no se puede dar una pauta concreta, si no que se debe analizar que la respuesta entregada tenga sentido. Se permite el uso de un floque `FPU` y en general se espera que hayan instanciado 8 registros de 80-bits conectados todos a 2 muxes, y cada mux es una entrada de la `FPU`. Luego la salida de resultado debiese retroalimentar los 8 registros.
Además algunos alumnos en función de la parte c) podrían haber incorporado algunos buffers y registros de comunicación.

b) Indique como conectaría el dispositivo al computador Von Neumann. Incluya un diagrama de alto nivel. **(1 pto.)**
Respuesta:
Se espera el diagrama de IO general que se pone en clases. Además debiesen explicar como planean conectar el coprocesador (ports o memory-mapping, ambos son admitidos).

c) Indique cómo serían entregados los datos y comandos al coprocesador. Detalle el tipo de comunicación, modo de interacción e incluya un diagrama. Es posible que, al contestar este ítem, deba modificar su respuesta de los anteriores. **(3 ptos.)**
Respuesta:
Se debe vigilar que esta respuesta sea concordante consigo misma. Además pueden proponer cambios o extensiones a lo anterior.
En general se esperaría que definan un registro de comandos y dos de parámetros para realizar las operaciones y establezcan una tabla de comandos y estado (75% del puntaje). Lo restante se debe asignar en función de cómo planteen el problema de transferir los 80 bits.

Pregunta 3

Sobre un computador Von Neumann con *memory-mapped* I/O se desea implementar memoria virtual con la capacidad de designar páginas en las que se puede ejecutar código y otras en las que no se puede ejecutar. Al respecto:

a) ¿Cómo lograría que una página en particular no pueda ser ejecutada? **(3 ptos.)**
Respuesta:
La respuesta buscada es que agreguen un bit de metadata a la tabla de páginas que señale si dicha página es ejecutable o no. En la realidad dicho bit es el `NX`. Luego para el avance del PC deberían verificar (o al enviar el address) dicho bit en la tabla y si es no-ejecutar levantar una excepción.

b) ¿Qué piezas deben ser agregadas para soportar este esquema? **(2 ptos.)**
Respuesta:
Para soportar memoria virtual deberían agregar al menos la MMU, que se compone de registro PTBR y un Adder, además de `TLB`. Ausencia de TLB descontar 0.1. Además para la revisión del bit se requiere un comparador (puerta AND).

c) ¿Tiene sentido hacer esto para un computador Harvard? Comente. **(1 pto.)**
Respuesta:
La respuesta es no, dado que las memorias de instrucciones y datos están completamente separadas.

Pregunta 4

Considere un computador `x86` monoprocesador, que posee un *pipeline* de N etapas. En base a esto, conteste las siguientes preguntas:

a) ¿Qué tipo de *hazard* podría generarse entre las etapas `Fetch` y `Mem`? **(1 pto.)**
Respuesta:
Hazard estructural ya que ambas etapas compiten por la `RAM`

b) Considere que se agrega soporte para multiprogramación al computador (*preemptive scheduling*), con la restricción de que sólo se puede hacer cambio de contexto una vez que todas las etapas de la **instrucción en ejecución** hayan terminado. ¿Qué modificaciones se deben hacer al computador para soportar esto? Considere que el computador ahora debe soportar memoria virtual y CPU *scheduling*. **(2 ptos.)**
Respuesta:
Además de que deben señalar que en `MEM` deberán agregar la MMU y TLB, deben señalar que tendrán que agregar una unidad de control para context-switch que haga stall del pipeline hasta que la última instrucción pasa por todo el pipeline antes de hacer el cambio (cambiar PTBR, cambiar TLB, cambiar PC).

c) Considere que ahora se relaja la restricción del cambio de contexto, permitiendo que se pueda hacer en cualquier momento (no es necesario que todas las etapas de la instrucción actual hayan terminado). En base a esto conteste lo siguiente:

   i. Desde el punto de vista del contenido del *pipeline*, ¿cuál es la diferencia con el caso anterior? **(1 pto.)**
   Respuesta:
   Ahora habrá mas de un proceso en el pipeline potencialmente, pero no se perderan más ciclos que los estrictamente necesarios por el cambio de contexto

   ii. Desde el punto de vista de tiempo de ejecución, ¿cuál esquema es preferible? **(1 pto.)**
   Respuesta:
   Este segundo esquema es preferible ya que pierde menos ciclos.

   iii. ¿Qué modificaciones se deben hacer al computador para soportar este esquema? **(1 pto.)**
   Respuesta:
   O bien en los registros intermedios ahora se deberán transportar la TLB y el PTBR o bien deberán hacer que en el cambio de contexto se guarden los registros intermedios en el PCB.

Pregunta 5

El siguiente programa se ejecutó en un computador con arquitectura `x86` que tiene una *caché* de 4 líneas de 2 palabras cada una:

Dirección	Label		
0		MOV	var2,1
1	loop1:	MOV	AL,var2
2		MUL	var2
3		CMP	var1,AL
4		JL	end
5		INC	var2
6		JMP	loop1
7	end:	DEC	var2
8		RET	
9	var1	db	**?**
10	var2	db	0

Al ejecutar el programa completo se obtuvo la siguiente secuencia de accesos a memoria:

0-10-1-10-2-10-3-9-4-5-10-6-1-10-2-10-3-9-4-5-10-6-1-10-2-10-3-9-4-7-10-8

Esta secuencia de accesos a memoria generó los siguiente estados en la *caché*:

Dir	L0	L1	L2	L3
0	0-1			
10	0-1	10-11		
1	0-1	10-11		
10	0-1	10-11		
2	0-1	10-11	2-3	
10	0-1	10-11	2-3	
3	0-1	10-11	2-3	
9	0-1	10-11	2-3	8-9
4	0-1	10-11	2-3	4-5
5	0-1	10-11	2-3	4-5
10	0-1	10-11	2-3	4-5
6	6-7	10-11	2-3	4-5
1	6-7	10-11	0-1	4-5
10	6-7	10-11	0-1	4-5
2	6-7	10-11	0-1	2-3
10	6-7	10-11	0-1	2-3

Dir	L0	L1	L2	L3
3	6-7	10-11	0-1	2-3
9	8-9	10-11	0-1	2-3
4	8-9	10-11	4-5	2-3
5	8-9	10-11	4-5	2-3
10	8-9	10-11	4-5	2-3
6	6-7	10-11	4-5	2-3
1	0-1	10-11	4-5	2-3
10	0-1	10-11	4-5	2-3
2	0-1	10-11	4-5	2-3
10	0-1	10-11	4-5	2-3
3	0-1	10-11	4-5	2-3
9	8-9	10-11	4-5	2-3
4	8-9	10-11	4-5	2-3
7	6-7	10-11	4-5	2-3
10	6-7	10-11	4-5	2-3
8	8-9	10-11	4-5	2-3

En base a esta información, responda lo siguiente:

a) ¿Qué valores puede tener la variable `var1` para que efectivamente se genere la secuencia de accesos detallada previamente? **(2 ptos.)**

b) ¿Cuál es el *hit-rate*? **(1 pto.)**

c) ¿Qué tipo de *caché* es: *unified* o *split*? **(1 pto.)**

d) ¿Qué función de correspondencia y algoritmo de reemplazo utiliza esta *caché* ? **(2 ptos.)**