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
```

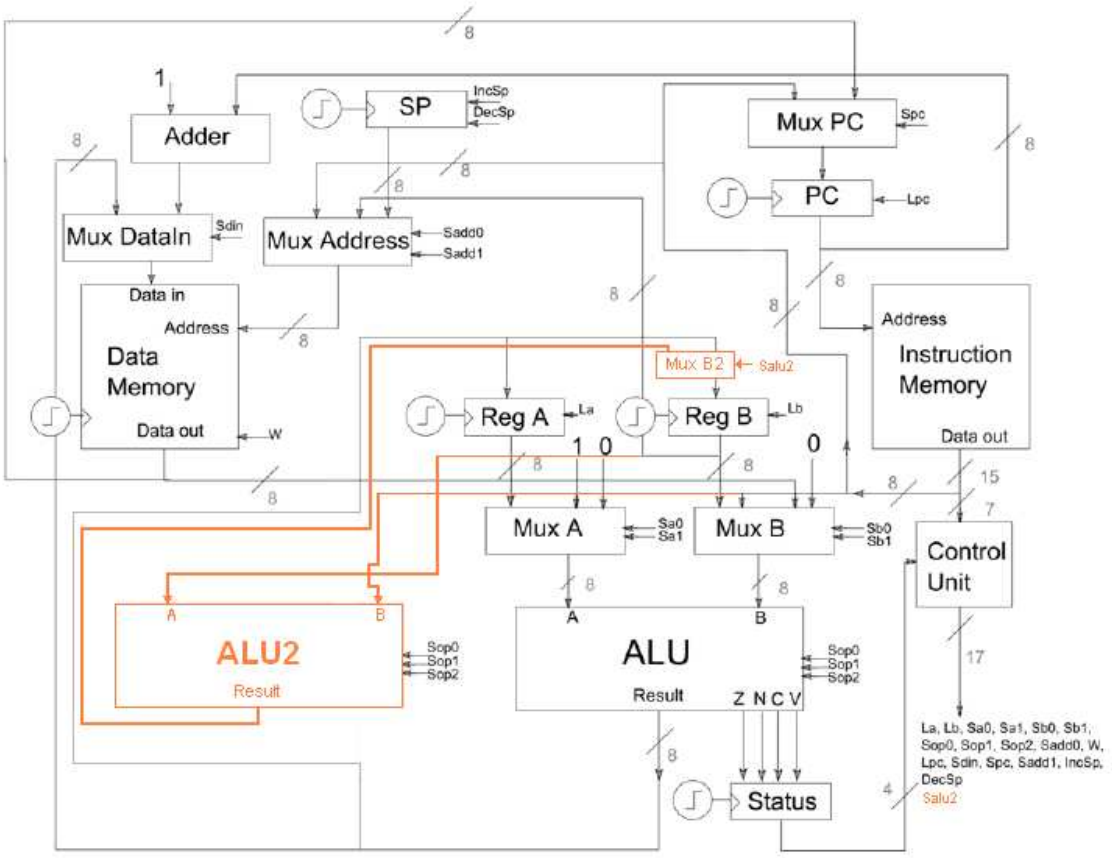
```
        LEA BX, h
        MOV SI, AL
        ADD [BX+SI], 1
        ADD [BP-2], 1
        JMP for
        end_for:

        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:

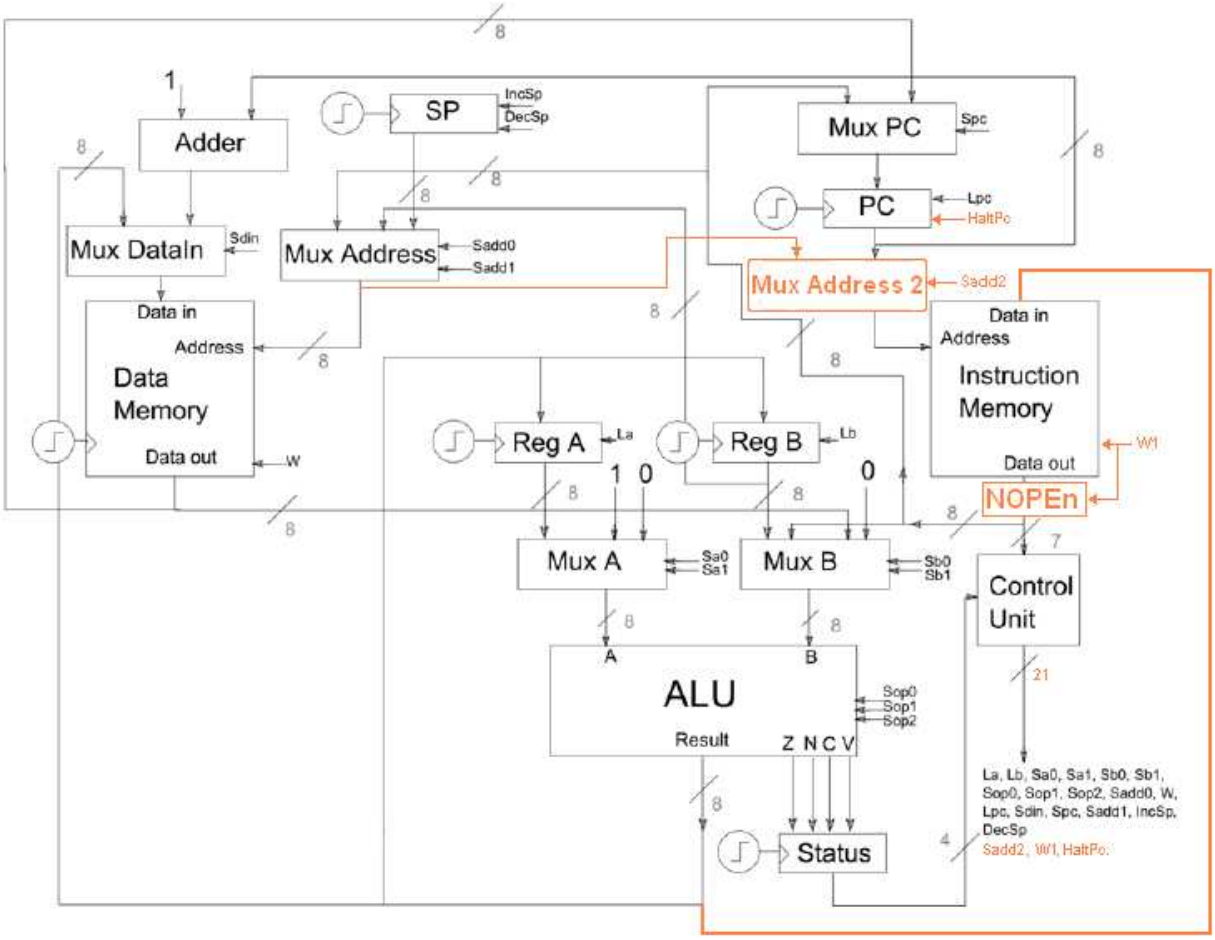


La siguiente tabla presenta los opcodes y señales de control de las nuevas operaciones, excluyendo las sieguiente señales, cuyo valor indicado a continuación se repita de la misma manera en todas las operaciones: Sadd=-, Sdin=-, Spc=-, W=0, IncSp=0 y DecSp=0.

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:



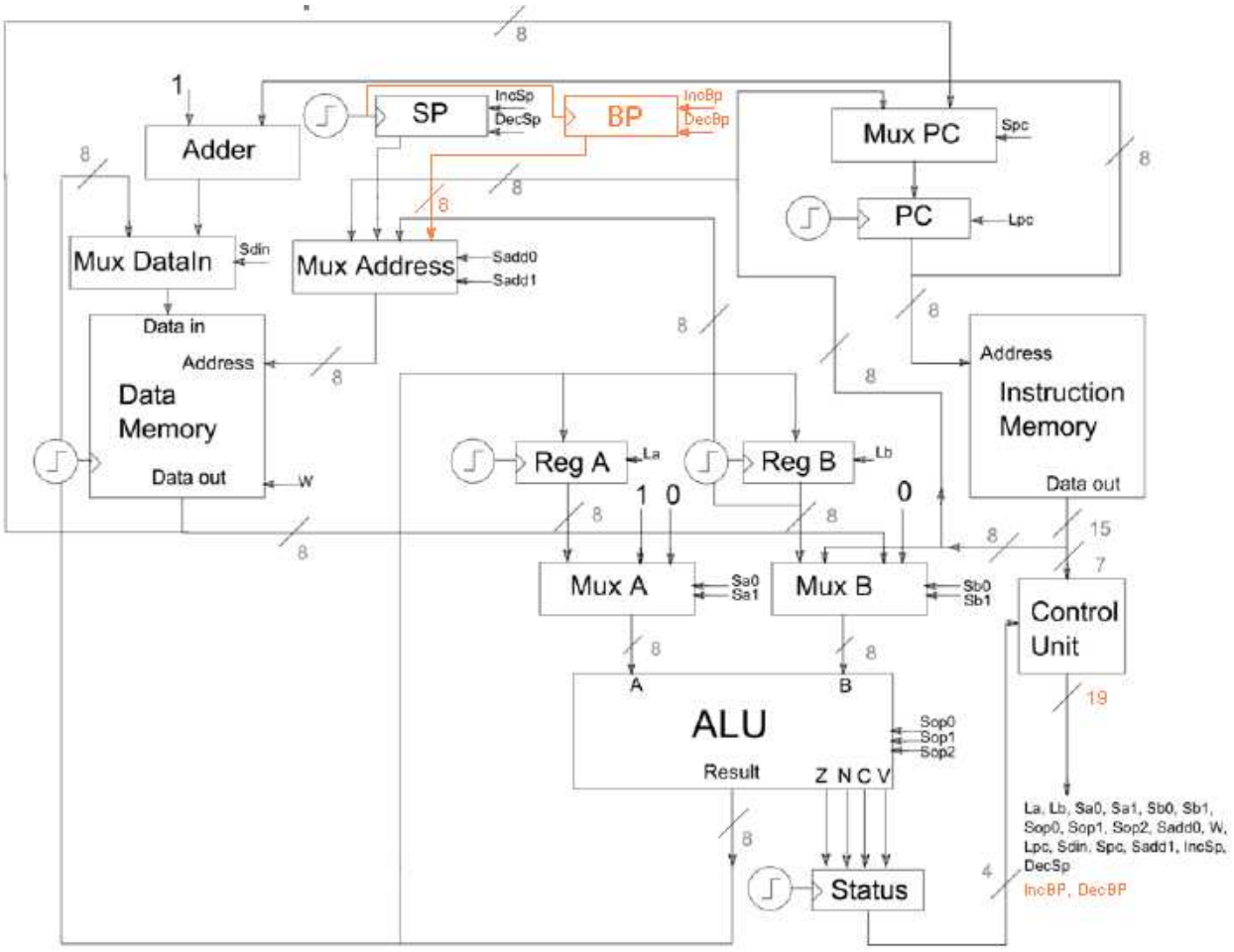
Desde el punto de vista de la ISA, además de NOP, se agregan las instrucciones MOV_DM (Dir), A y MOV_DM (B), A, que escriben el valor almacenado en el registro A en la memoria de datos. Cabe destacar que estas instrucciones toman 2 ciclos, una para escribir el dato en memoria y la siguiente ejecutando 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: