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:

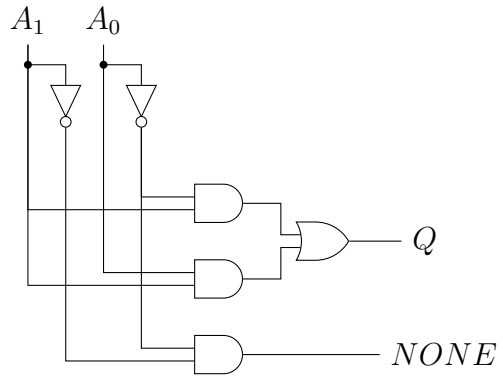
   - Tabla de verdad:

| $A_1$ | $A_0$ | $Q$ | 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

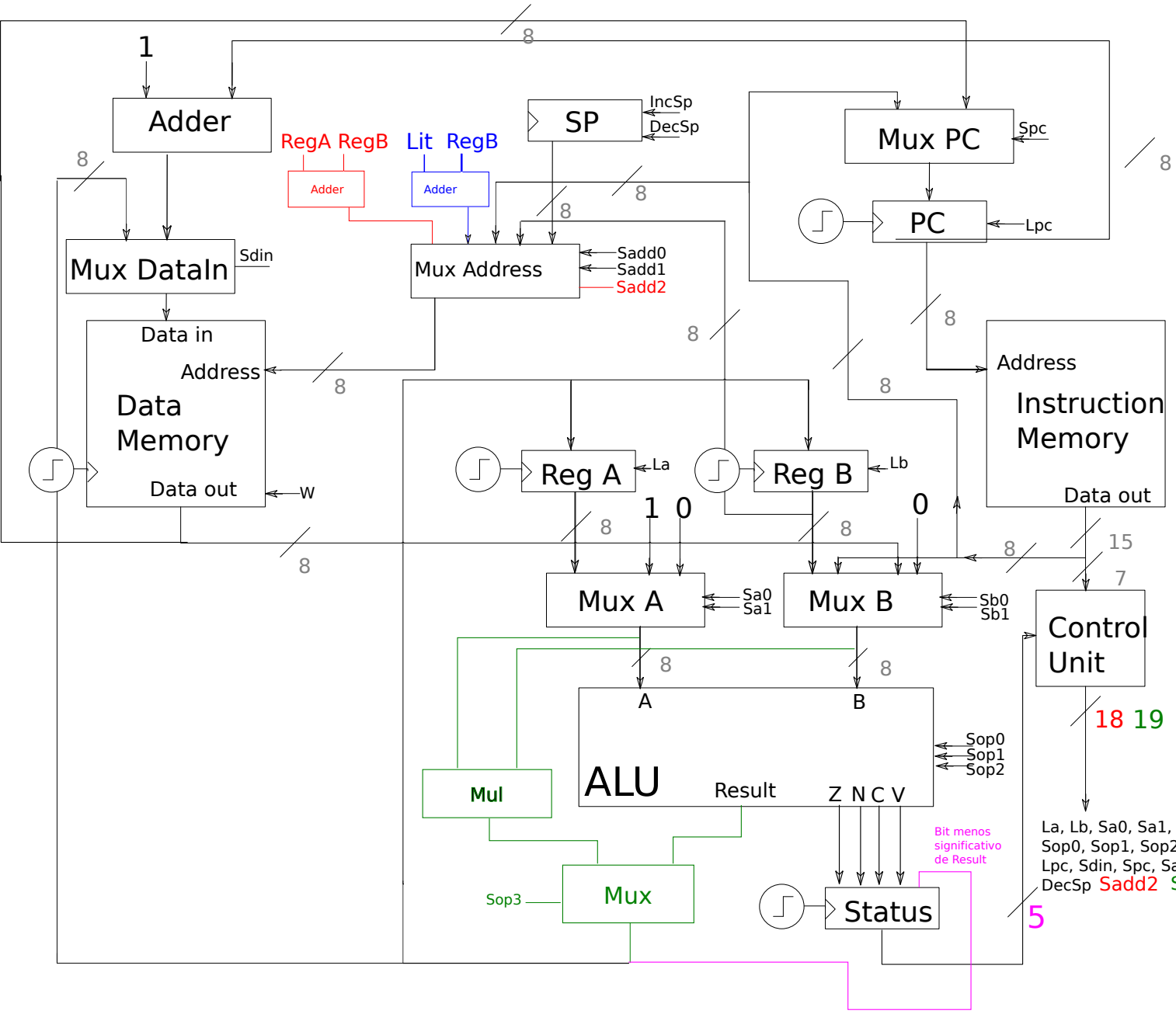
Revisar PDF adjunto, se utilizaron los siguientes colores para detallar los cambios:

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%**