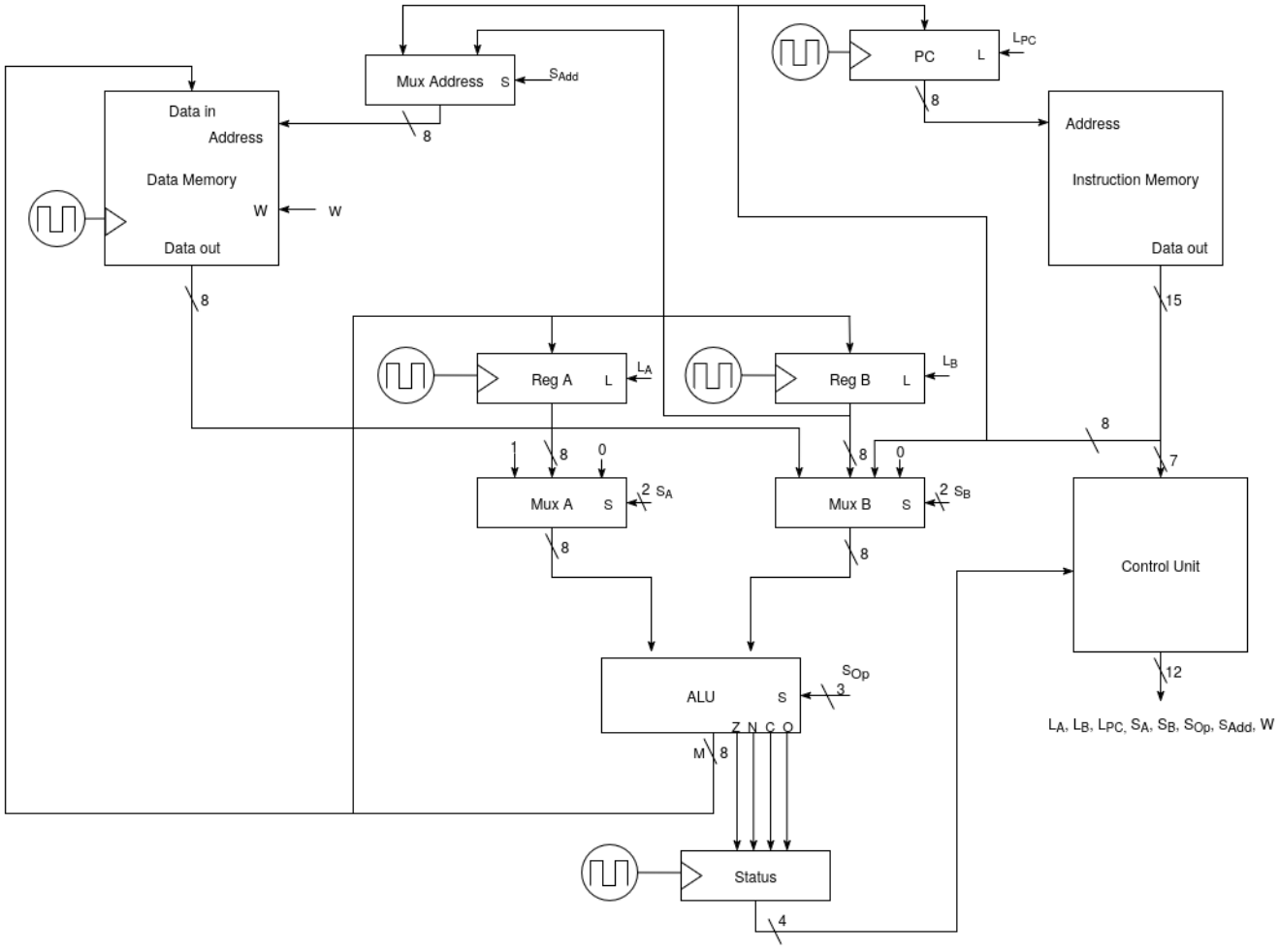
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

**IIC2343 - Arquitectura de Computadores (II/2025)**

# Ayudantía 5

**Ayudantes: Daniela Ríos (danielaarp@uc.cl), Alberto Maturana (alberto.maturana@uc.cl), José Mendoza (jfmendoza@uc.cl)**

## Computador básico



## Pregunta 1: Assembly - Saltos

(a) Describa qué hace el fragmento de código, escrito en el Assembly del computador básico del curso. Indique los valores de los registros `A` y `B` al finalizar la ejecución del código.

```
DATA:
  x -5
  r 2

CODE:
    MOV A,(x)
    CMP A,0
    JEQ zero
    CMP A,0
    JLT neg
    JMP power

    zero:
        MOV A,(r)
        SHR A,A
        MOV (r),A
        JMP end

    neg:
        NOT A,A
        ADD A,1
        JMP power

    power:
        CMP A,1
        JEQ end
        MOV B,A
        MOV A,(r)
        SHL A,A
        MOV (r),A
        MOV A,B
        SUB A,1
        JMP power

    end:
        MOV A,(r)
```

**Solución:** El fragmento computa 2 elevado al valor absoluto de la variable $x$. Esto lo podemos ver ya que, para $x < 0$, lo que se hace es calcular su inverso aditivo y luego procede a calcular la potencia, mientras que para $x \geq 0$ se calcula inmediatamente.

Como `A` tiene el valor de $x$, que es -5, al calcular el inverso aditivo esto nos deja un valor de 5, luego tenemos que `A` tiene el valor de `r`, que es 2. Finalmente se computa el nuevo valor del registro `A`, el cual es $\texttt{A} = r^{|x|} = 2^5 = 32$, y el valor del registro `B` termina siendo $\texttt{B} = 2$, ya que lo utilizamos para guardar la variable con la que iterabamos.

(b) Programe en *assembly* un código que calcule la multiplicación de dos números positivos. Para ello, se proporciona la sección `DATA`, la cual contiene los números $x$ e $y$. Guarde el resultado final en la dirección **r**.

```
DATA:
    r 0
    x 9
    y 7
```

**Solución:** Para resolver el problema, nos aprovechamos del hecho de que la multiplicación de $x$ e $y$, es la suma de $x$ realizada $y$ veces consigo mismo o vice-versa, y aunque no tengamos una instrucción para multiplicar, de esta forma conseguimos realizar la operación con dos números positivos.

```
DATA:
    r 0
    x 9
    y 7
CODE:
    MOV B, (x)

    loop:
    MOV A, (r)
    ADD A, B
    MOV (r), A
    MOV A, (y)
    SUB A, 1
    MOV (y), A
    CMP A, 0
    JEQ end
    JMP loop

    end:
```

## Pregunta 2: Análisis - Saltos

El siguiente fragmento, escrito en el Assembly del computador básico del curso, se encarga de revisar si un arreglo `arr1` es igual al doble del inverso de otro arreglo `arr2`, es decir, que el primer elemento de `arr1` sea igual al doble del último elemento de `arr2`; que el segundo elemento de `arr1` sea igual al doble del penúltimo elemento de `arr2`, y así sucesivamente. En caso de cumplirse, se tendrá que `res == 1` al finalizar la ejecución, y `res == 0` en otro caso. No obstante, el código da un cómputo equivocado. Identifique qué instrucción(es) está(n) errada(s) y explique justificadamente cómo corregirla(s) para que el código actualice `res` correctamente.

```
DATA:
  res   1
  len   4
  arr1  6
        10
        16
        4
  arr2  2
        8
        5
        3
  iter  0
  i1    0
  i2    0
  temp  0
```

```
CODE:
  MOV A, arr1
  MOV (i1), A
  MOV A, arr2
  ADD A, (len)
  MOV (i2), A

  loop:
  MOV B, (i1)
  MOV A, (B)
  SHR A,A
  MOV (temp), A

  MOV B, (i2)
  MOV A, (B)
  CMP A, (temp)
  JNE no_son_iguales

  INC (iter)
  INC (i1)
  MOV A, (i2)
  SUB A, 1
  MOV (i2), A

  MOV A, (iter)
  CMP A, (len)
  JEQ end
  JNE loop

  no_son_iguales:
  MOV A, 0
  MOV (res), A

  end:
```

**Solución:**

1. La dirección de memoria del último elemento de `arr2` se obtiene como la suma entre su dirección base y su largo. No obstante, esta suma genera la dirección de la variable `iter`, ya que se excede en una unidad. Esto se arregla insertando la instrucción `SUB A,1` antes de `MOV (i2), A`, quinta instrucción del fragmento original.
2. Luego de realizar la comparación `CMP A,(len)`, se ejecuta `JEQ end`. Es decir, se termina

la ejecución si `A` es igual al largo de los arreglos. En caso de no cumplirse, se quiere seguir iterando a través del salto a `loop`. No obstante, la instrucción `JNE loop` hará uso de las *flags* condicionales computadas por `JEQ end` y no las de `CMP A,(len)`, dado que estas se actualizan en cada ciclo. Por lo tanto, no se evaluará correctamente la condición de salto. Esto se puede resolver de dos formas: cambiando la instrucción `JNE loop` por `JMP loop`; o insertando nuevamente la instrucción `CMP A,(len)` posterior a `JEQ end`.

Dos opciones para corregir el código serían las siguientes:

(a)

```
CODE:
  MOV A, arr1
  MOV (i1), A
  MOV A, arr2
  ADD A, (len)
  //Modificacion 1
  SUB A, 1
  MOV (i2), A

  loop:
  MOV B, (i1)
  MOV A, (B)
  SHR A,A
  MOV (temp), A

  MOV B, (i2)
  MOV A, (B)
  CMP A, (temp)
  JNE no_son_iguales

  INC (iter)
  INC (i1)
  MOV A, (i2)
  SUB A, 1
  MOV (i2), A

  MOV A, (iter)
  CMP A, (len)
  JEQ end
  //Modificacion 2
  CMP A, (len)
  JNE loop

  no_son_iguales:
  MOV A, 0
  MOV (res), A

  end:
```

(b)

```
CODE:
  MOV A, arr1
  MOV (i1), A
  MOV A, arr2
  ADD A, (len)
  //Modificación 1
  SUB A, 1
  MOV (i2), A

  loop:
  MOV B, (i1)
  MOV A, (B)
  SHR A,A
  MOV (temp), A

  MOV B, (i2)
  MOV A, (B)
  CMP A, (temp)
  JNE no_son_iguales

  INC (iter)
  INC (i1)
  MOV A, (i2)
  SUB A, 1
  MOV (i2), A

  MOV A, (iter)
  CMP A, (len)
  JEQ end
  //Modificación 2
  JMP loop

  no_son_iguales:
  MOV A, 0
  MOV (res), A

  end:
```

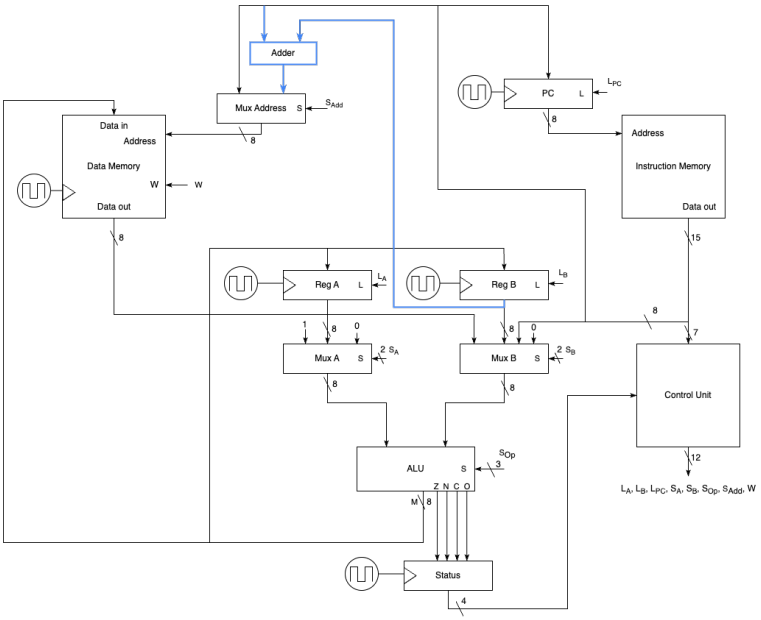
## Pregunta 3: Modificación computador básico con saltos

Modifique la arquitectura del computador básico para implementar las instrucciones:

- `MOV A,(B+offset)`
- `MOV (B+offset),A`
- `MOV B,(B+offset)`
- `MOV (B+offset),B`

Es decir, instrucciones de direccionamiento indirecto con registro `B` y *offset*, siendo este último un literal. Para cada instrucción, deberá incluir la combinación completa de señales que la ejecutan. Por cada señal de carga y escritura, deberá indicar si se activan (1) o no (0); en las señales de selección, deberá indicar el nombre de la entrada escogida ("-" si no afecta).

**Solución:** Para la implementación de estas instrucciones, se puede modificar la conexión entre el registro `B` y el componente `Mux Address` para que ahora reciba el resultado de un sumador que sume el valor de `B` y el literal de la instrucción:




De esta forma, no es necesario agregar ni modificar señales, pero sí asegurar que el compilador haga uso del literal 0 para las instrucciones `MOV A,(B)`, `MOV B,(B)`, `MOV (B),A` y `MOV (B),B`. La tabla de señales es como sigue para las instrucciones implementadas:

| **Instrucción** | $L_A$ | $L_B$ | $L_{PC}$ | W | $S_A$ | $S_B$ | $S_{OP}$ | $S_{Add}$ |
|---|---|---|---|---|---|---|---|---|
| `MOV A,(B+offset)` | 1 | 0 | 0 | 0 | ZERO | DOUT | ADD | B+offset |
| `MOV B,(B+offset)` | 0 | 1 | 0 | 0 | ZERO | DOUT | ADD | B+offset |
| `MOV (B+offset),A` | 0 | 0 | 0 | 1 | A | ZERO | ADD | B+offset |
| `MOV (B+offset),B` | 0 | 0 | 0 | 1 | ZERO | B | ADD | B+offset |

## Feedback ayudantía

Escanee el QR para entregar feedback sobre la ayudantía.