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

IIC2343 - Arquitectura de Computadores (II/2025)

Ayudantía 4

Ayudantes: Daniela Ríos (danielaarp@uc.cl), Alberto Maturana (alberto.maturana@uc.cl), Tomás López Massaro (tomas.lopezm20@uc.cl)

Pregunta 1: Preguntas Conceptuales

(a) Un *assembler* es una herramienta de software que traduce un programa escrito en *assembly* a su representación equivalente en código máquina (binario), que puede ser ejecutado directamente por el procesador. Describa en detalle los pasos que debe realizar un *assembler* para procesar código *assembly* del computador básico.

(b) Un *disassembler* es un programa que transforma código binario ejecutable en *assembly*. Describa cómo funcionaría un *disassembler* para el computador básico. ¿Es posible obtener el *assembly* original a partir de un programa ubicado en la memoria de instrucciones?

Solución:

(a) 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*.

(b) Un *disassembler* para el computador básico trabajará 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 solo 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 código *assembly* original, ya que los *labels* son traducidos como direcciones (números), **perdiendo la información** sobre sus nombres explícitos. Sin embargo, el código reconstruido funcionaría de igual manera que el código original, solo que los nombres de las variables y *labels* serían distintos.

Pregunta 2: Programabilidad

(a) Describa qué hace el siguiente fragmento de código, escrito en el *assembly* del computador básico del curso. Indique los valores de los registros `A` y `B`, y el de las variables al finalizar la ejecución del código. Recuerde que el computador básico del curso realiza operaciones con 8 bits.

***Bonus*:** Adicionalmente, indique los valores finales de las *flags* C, Z y N.

```
DATA:
  var1 4
  var2 8
  var3 0

CODE:
  MOV A, (var1)
  SHR B,A
  ADD B,A
  MOV (var1), B
  MOV A, (var2)
  SHL B,A
  ADD B,A
  MOV (var2), B
  MOV A, (var1)
  AND A, B
  NOT A, A
  MOV (var3), A
  INC (var3)
```

(b) Programe en *assembly* un código que calcule el perímetro de un rectángulo. Para ello, se proporciona la sección `DATA`, la cual contiene las coordenadas de los vértices del rectángulo almacenadas en dos *arrays* de forma ordenada. Cada índice i de estos arreglos corresponde al punto (x_i, y_i). Utilice **direccionamiento indirecto** y guarde el resultado final en la dirección **p**.

Propuesto: Resuelva este mismo problema calculando el largo de cada lado en cualquier orden. *Hint*: Utilice saltos.

```
DATA:
  p 0
  x 2
    6
    6
    2
  y 1
    1
    4
    4
CODE:
```

Solución:

(a) A continuación, se muestra el valor que cambia después de cada instrucción:

```
DATA:
  var1 4
  var2 8
  var3 0

CODE:
  MOV A, (var1)  // A      = 00000100
  SHR B,A        // B      = 00000010
  ADD B,A        // B      = 00000110
  MOV (var1), B  // (var1) = 00000110
  MOV A, (var2)  // A      = 00001000
  SHL B,A        // B      = 00010000
  ADD B,A        // B      = 00011000
  MOV (var2), B  // (var2) = 00011000
  MOV A, (var1)  // A      = 00000110
  AND A, B       // A      = 00000000
  NOT A, A       // A      = 11111111
  MOV (var3), A  // (var3) = 11111111
  INC (var3)     // (var3) = 00000000
```

El fragmento de código realiza operaciones de desplazamiento y suma sobre los valores iniciales de las variables. Dado que `var1` = 4, se carga el valor de `var1` en `A`. Primero, con la instrucción `SHR B, A`, se realiza un corrimiento a la derecha de una posición sobre el valor de `A`, es decir, `SHR B, A` = 2, y se guarda en el registro `B`. Luego se ejecuta `ADD B, A`, con lo cual `B` = 4 + 2 = 6. Este resultado se almacena en `var1`, por lo que ahora `var1` = 6.

Posteriormente, se carga el valor de `var2` = 8 en `A` y se aplica `SHL B, A`, es decir, un corrimiento a la izquierda de una posición: `SHL B, A` = 16, que se almacena en el registro `B`. A continuación, con la instrucción `ADD B, A`, se obtiene `B` = 16 + 8 = 24, valor que se almacena en `var2`.

Luego, se carga el valor actualizado de `var1` en el registro `A` y se ejecuta `AND A, B` = 1, resultando `A` = 6 `AND` 24 = 0. A este valor se le aplica la operación `NOT A`, obteniendo `A` = 255. Este resultado se almacena en `var3` y se termina incrementando en 1, obteniendo `var3` = 0. Esto debido a que se está operando con 8 bits.

Finalmente, después de estas operaciones, los registros y variables quedan con los valores:

Reg `A`: 11110b = 30d

Reg `B`: 11000b = 24d

`var1`: 00110b = 6d

`var2`: 11000b = 24d

`var3`: 11111b = 31d

***Bonus*:** Los valores de las *flags* son `C` = 1, `Z` = 1 y `N` = 0.

(b) Para resolver el problema, se aprovecha el hecho de que los lados del rectángulo son paralelos a los ejes x e y del plano cartesiano. Primero, se calcula la longitud de uno de sus lados horizontales y se duplica, lo que equivale a sumar también su lado opuesto. De manera análoga, se aplica la misma estrategia para obtener la suma de los lados verticales. Finalmente, se suman estos resultados para obtener el perímetro.

```
DATA:
  p 0
  x 2
    6
    6
    2
  y 1
    1
    4
    4
CODE:
  // Suma en el eje x
  MOV A, x     // Se obtiene la dirección del primer elemento del array x
  ADD A, 1     // Se avanza una posición en el array
  MOV B, A     // Se copia la dirección deseada en B
  MOV A, (B)   // Se lleva a cabo el direccionamiento indirecto para obtener el primer punto
  MOV B, (x)   // Se obtiene el segundo punto por direccionamiento directo
  SUB B, A     // Se obtiene largo de primer lado horizontal
  MOV A, B     // Se duplica el valor del lado en A
  ADD A, B     // Se suma el segundo lado en x
  MOV (p), A   // Se almacena el resultado
  // Suma en el eje y
  MOV A, y     // Se obtiene la dirección del primer elemento del array y
  ADD A, 2     // Se avanzan dos posiciones en el array
  MOV B, A     // Se copia la dirección deseada en B
  MOV A, (B)   // Se lleva a cabo el direccionamiento indirecto para obtener el primer punto
  MOV B, (y)   // Se obtiene el segundo punto por direccionamiento directo
  SUB B, A     // Se obtiene largo de primer lado vertical
  MOV A, B     // Se duplica el valor del lado en A
  ADD A, B     // Se suma el segundo lado en y
  ADD A, (p)   // Se suma al resultado ya almacenado
  MOV (p), A   // Se guarda el resultado final
```

***Disclaimer*:** Existen múltiples formas de resolver este ejercicio, por lo que un código diferente que llegue al mismo resultado es igual de válido.

Pregunta 3: Modificación del computador básico

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

a) Direccionamiento indirecto por registro base + *offset*. Por ejemplo, `MOV A, (B + offset)`.

b) Además del direccionamiento anterior, agregar direccionamiento indirecto por registro base + registro índice. Por ejemplo, `MOV A, (B + A)`.

c) Agregar instrucción `MUL A, B`, la cual multiplica las salidas del mux A y mux B, para luego usar el resultado como salida de la ALU. Se puede utilizar un bloque como multiplicador.

d) *Condition code* I, que indica cuando un número es impar. Indique como 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.

Solución:

(a) Para añadir una nueva forma de direccionamiento, se agrega una nueva entrada al Mux Address de la siguiente forma (la modificación de este item se muestra con azul):

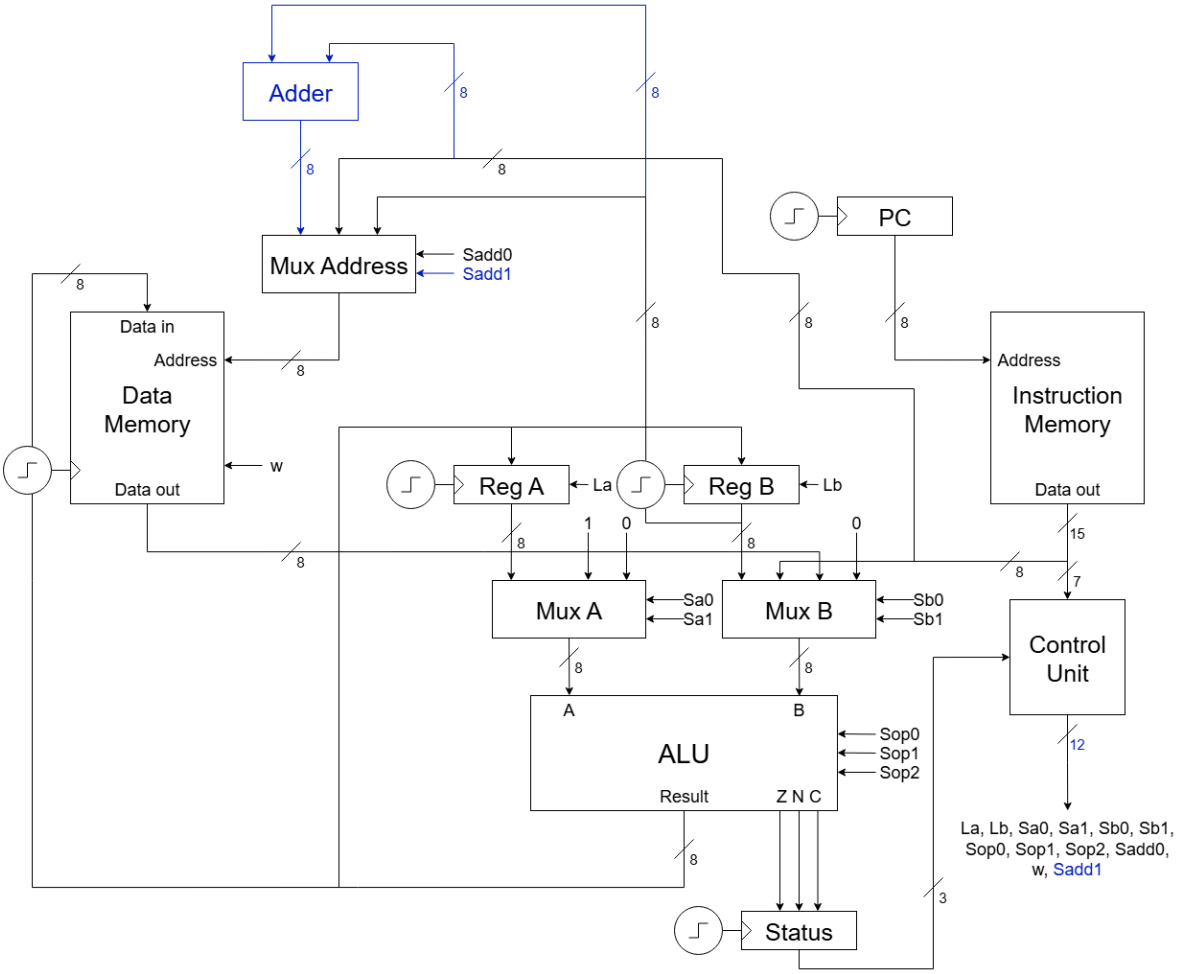

Figura 1: Diagrama Solución a.

(b) De manera similar al item anterior, se agrega una nueva entrada al Mux Address de la siguiente forma (la modificación de este item se muestra con rojo):

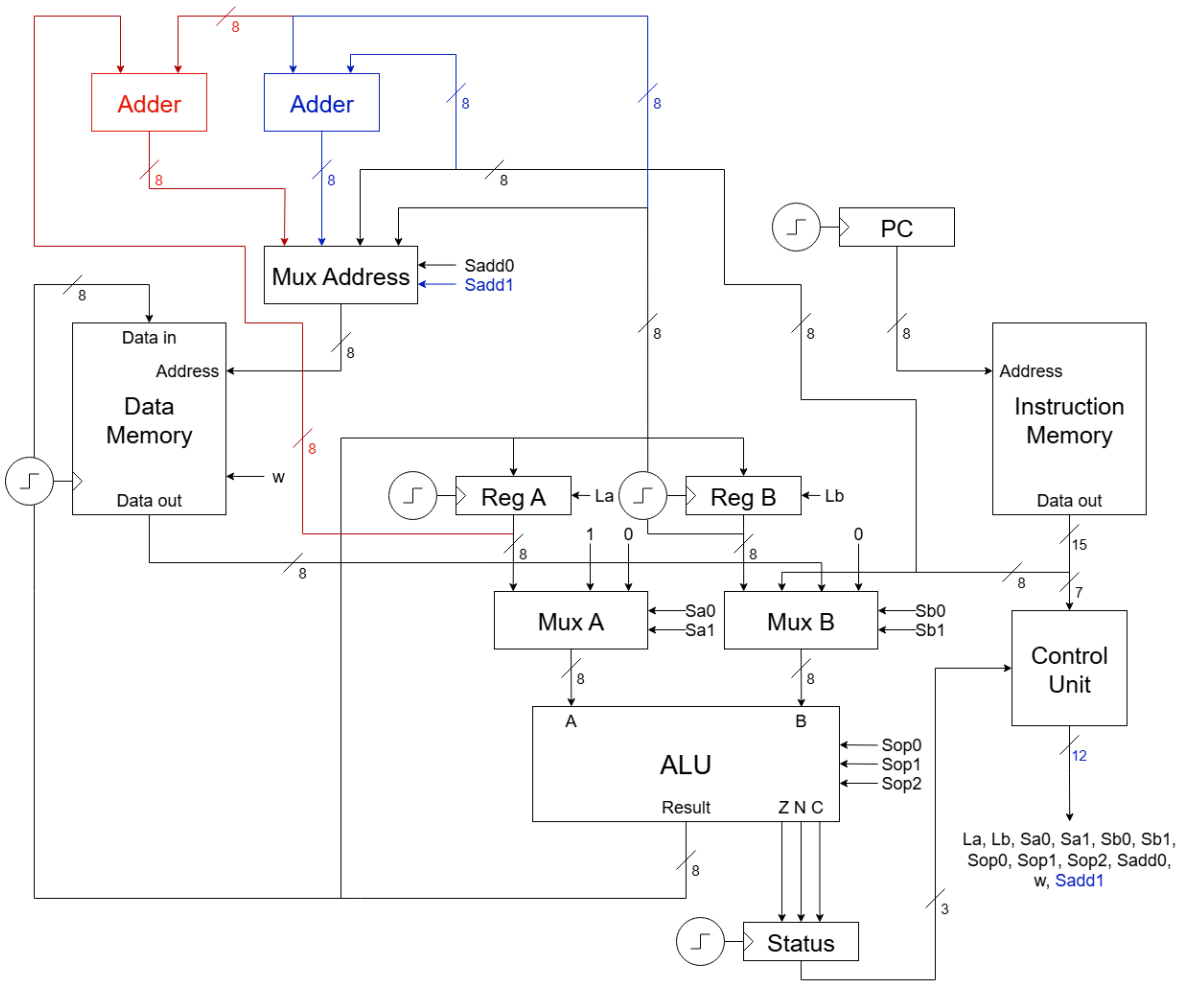

Figura 2: Diagrama Solución b.

(c) Para añadir una nueva operación, como lo es la multiplicación, se tendrá que manejar con un multiplexor externo, ya que las 8 operaciones ya existentes ya cubren todas las posibilidades del selector Sop. A continuación se muestra (la modificación de este item se muestra con verde):

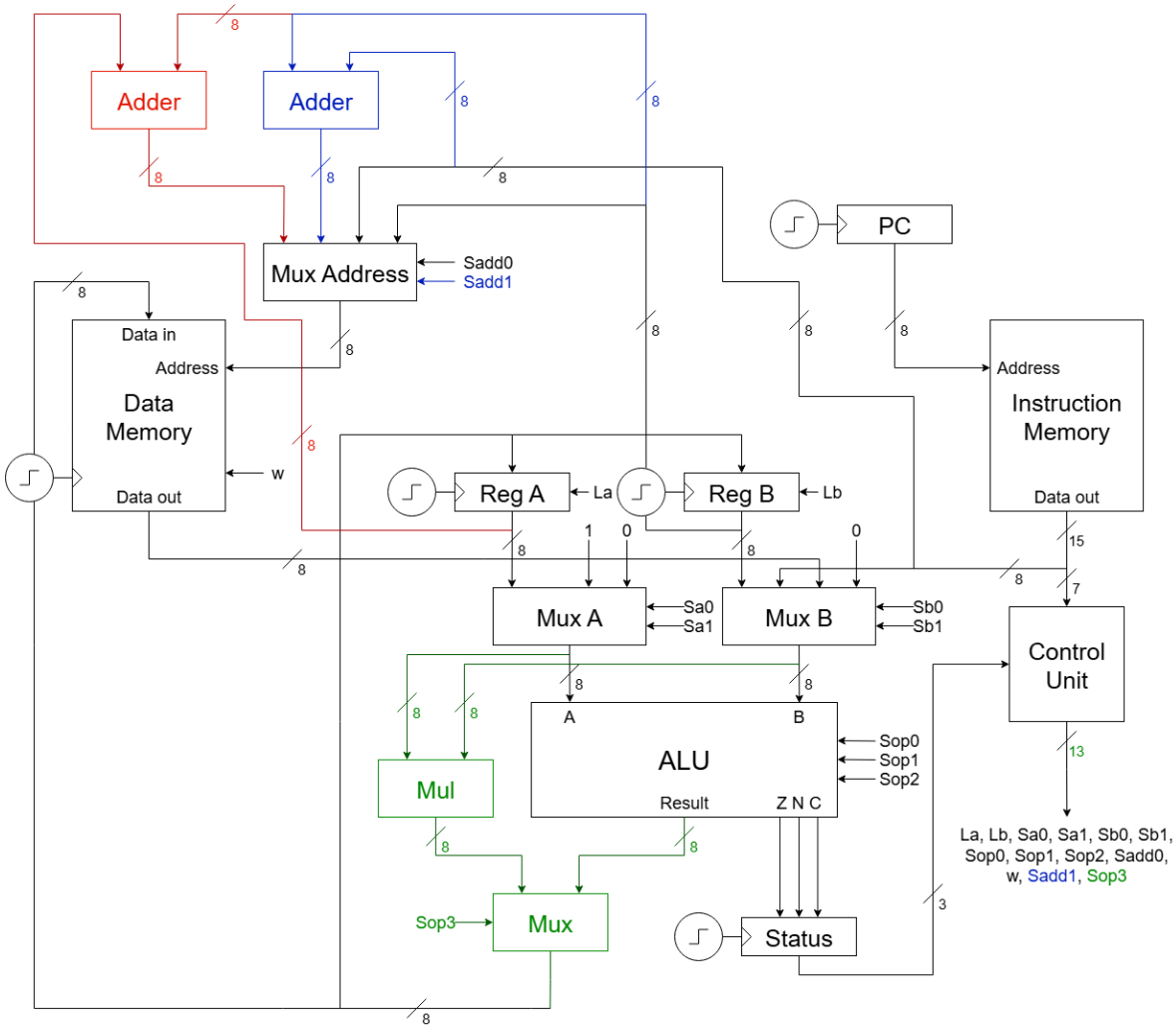

Figura 3: Diagrama Solución c.

(d) Para esta modificación, basta con conectar el bit menos significativo de Result al registro Status. De este modo, si el bit es 1, el resultado será impar; en caso contrario, será par (la modificación de este item se muestra con magenta):

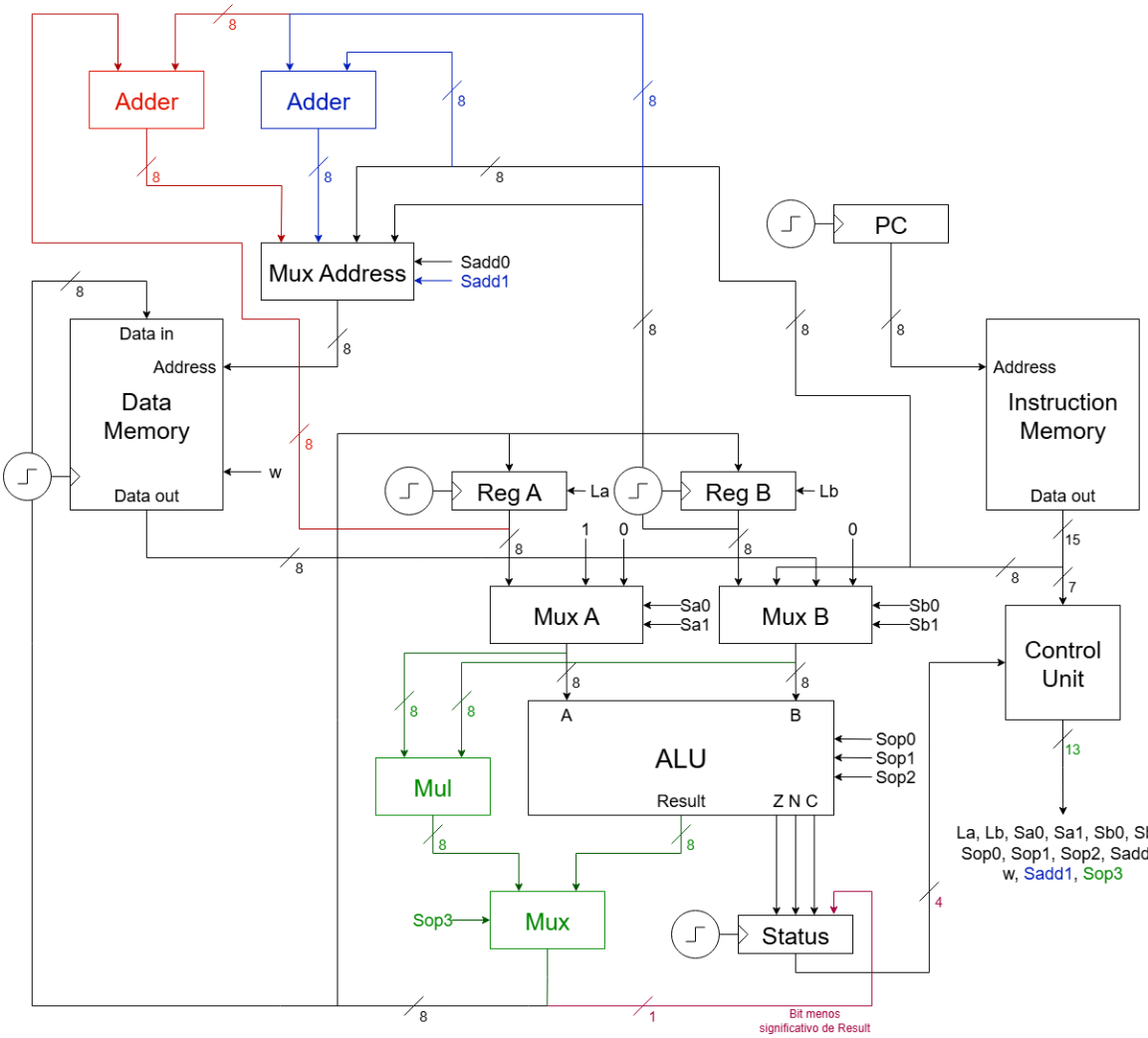

Figura 4: Diagrama Solución d.

Diagrama Computador Básico

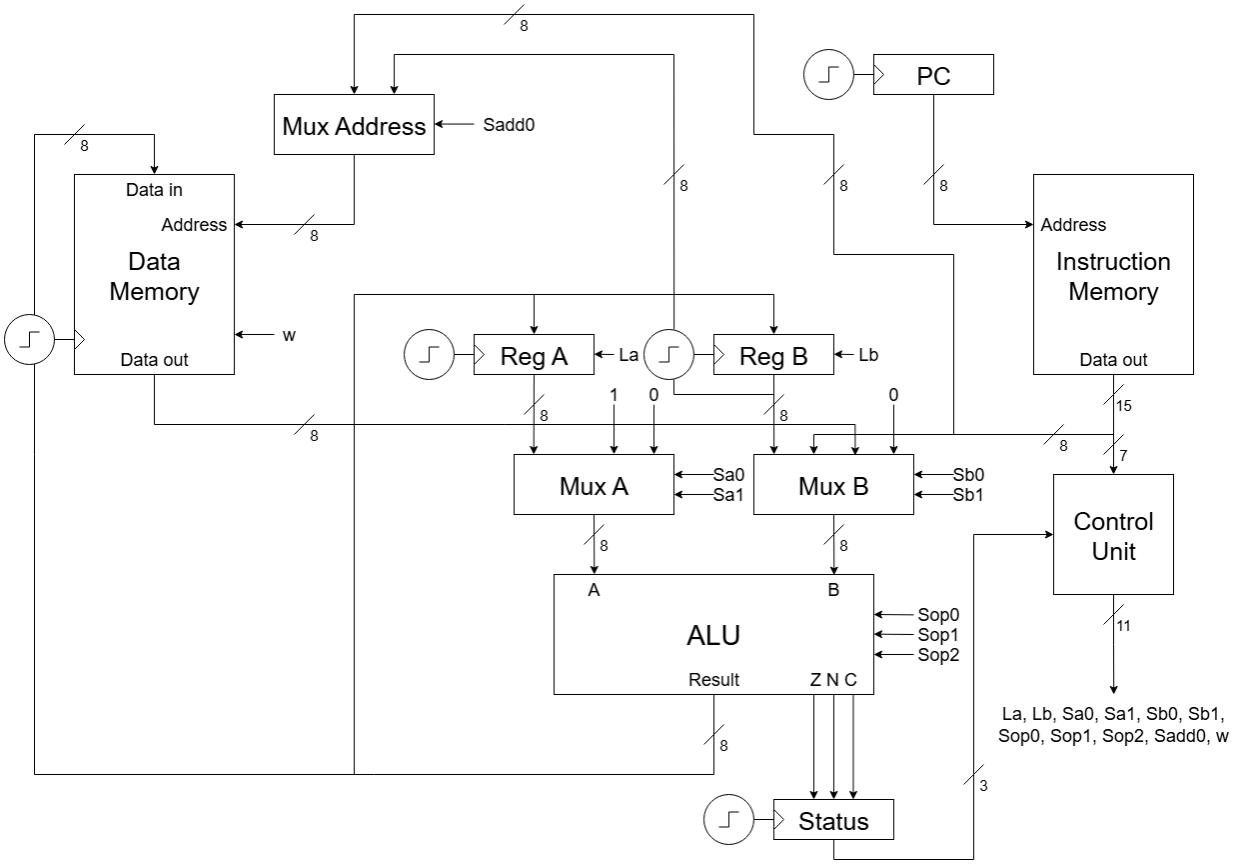

Figura 5: Computador Básico Sin Saltos.

Tabla de instrucciones

Instrucción	Operandos	*Opcode*	Condición	L_A	L_B	S_A0,1	S_B0,1	S_{Op}0,1,2	S_{Add}0,1	W
MOV	A,B	0000000		1	0	ZERO	B	ADD	-	0
	B,A	0000001		0	1	A	ZERO	ADD	-	0
	A,Lit	0000010		1	0	ZERO	LIT	ADD	-	0
	B,Lit	0000011		0	1	ZERO	LIT	ADD	-	0
	A,(Dir)	0000100		1	0	ZERO	DOUT	ADD	LIT	0
	B,(Dir)	0000101		0	1	ZERO	DOUT	ADD	LIT	0
	(Dir),A	0000110		0	0	A	ZERO	ADD	LIT	1
	(Dir),B	0000111		0	0	ZERO	B	ADD	LIT	1
	A,(B)	0001000		1	0	ZERO	DOUT	ADD	B	0
	B,(B)	0001001		0	1	ZERO	DOUT	ADD	B	0
	(B),A	0001010		1	0	A	ZERO	ADD	B	1
ADD	A,B	0001011		1	0	A	B	ADD	-	0
	B,A	0001100		0	1	A	B	ADD	-	0
	A,Lit	0001101		1	0	A	LIT	ADD	-	0
	A,(Dir)	0001110		1	0	A	DOUT	ADD	LIT	0
	A,(B)	0001111		1	0	A	DOUT	ADD	B	0
	(Dir)	0010000		0	0	A	B	ADD	LIT	1
SUB	A,B	0010001		1	0	A	B	SUB	-	0
	B,A	0010010		0	1	A	B	SUB	-	0
	A,Lit	0010011		1	0	A	LIT	SUB	-	0
	A,(Dir)	0010100		1	0	A	DOUT	SUB	LIT	0
	A,(B)	0010101		1	0	A	DOUT	SUB	B	0
	(Dir)	0010110		0	0	A	B	SUB	LIT	1
AND	A,B	0010111		1	0	A	B	AND	-	0
	B,A	0011000		0	1	A	B	AND	-	0
	A,Lit	0011001		1	0	A	LIT	AND	-	0
	A,(Dir)	0011010		1	0	A	DOUT	AND	LIT	0
	A,(B)	0011011		1	0	A	DOUT	AND	B	0
	(Dir)	0011100		0	0	A	B	AND	LIT	1
OR	A,B	0011101		1	0	A	B	OR	-	0
	B,A	0011110		0	1	A	B	OR	-	0
	A,Lit	0011111		1	0	A	LIT	OR	-	0
	A,(Dir)	0100000		1	0	A	DOUT	OR	LIT	0
	A,(B)	0100001		1	0	A	DOUT	OR	B	0
	(Dir)	0100010		0	0	A	B	OR	LIT	1
NOT	A,A	0100011		1	0	A	-	NOT	-	0
	B,A	0100100		0	1	A	-	NOT	-	0
	(Dir)	0100101		0	0	A	B	NOT	LIT	1
XOR	A,B	0100110		1	0	A	B	XOR	-	0
	B,A	0100111		0	1	A	B	XOR	-	0
	A,Lit	0101000		1	0	A	LIT	XOR	-	0
	A,(Dir)	0101001		1	0	A	DOUT	XOR	LIT	0
	A,(B)	0101010		1	0	A	DOUT	XOR	B	0
	(Dir)	0101011		0	0	A	B	XOR	LIT	1
SHL	A,A	0101100		1	0	A	-	SHL	-	0
	B,A	0101101		0	1	A	-	SHL	-	0
	(Dir)	0101110		0	0	A	B	SHL	LIT	1
SHR	A,A	0101111		1	0	A	-	SHR	-	0
	B,A	0110000		0	1	A	-	SHR	-	0
	(Dir)	0110001		0	0	A	B	SHR	LIT	1
INC	B	0110010		0	1	ONE	B	ADD	-	0
	(B)	0110011		0	0	ONE	DOUT	ADD	B	1
	(Dir)	0110100		0	0	ONE	DOUT	ADD	LIT	1

Tabla 1: ISA del computador básico.

Feedback ayudantía

Escanee el QR para entregar feedback sobre la ayudantía.