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

IIC2343 - Arquitectura de Computadores (I/2026)

Ayudantía 4: Computador básico I

Ayudantes: Alberto Maturana (alberto.maturana@uc.cl), Mario Rojas (mario.denzel@estudiante.uc.cl, Nicolás Romo (nroma@uc.cl)

Pregunta 1: Preguntas conceptuales

a. Describa y entregue las señales de una instrucción que es soportada por el computador básico, pero que no se encuentra en el conjunto de instrucciones de este (*i.e.* que no necesita modificaciones de la arquitectura para implementarla).

b. ¿Por qué motivo no está soportada la instrucción `ADD B, LIT` en el computador básico visto en clases? Explique con palabras cómo podría implementarse.

Solución:

a. Las instrucciones `B, LIT` no se puede implementar para cualquier literal debido a que en la arquitectura actual, no existe una conexión entre el `MUX A` y la memoria de instrucciones, esto implica que la `ALU` no puede recibir el valor del registro `B` y de un literal de la memoria de instrucciones a la vez, los dos únicos literales que pueden operar con `B` son `1`y `0` que son las conexiones que están en el `MUX A`. Para poder implementar la instrucción es importante notar que se debe modificar la arquitectura agregando una conexión entre la memoria de instrucciones y el espacio libre del `MUX A`, por lo que, haciendo esto y asignándole la señal de selección libre se podría ejecutar esta instrucción en el computador.

b. Como se puede observar, en la conjunto de instrucciones del computador básico hay un gran número de *opcodes* que no se están utilizando, en el caso del computador básico sin saltos las instrucciones son de `7` *bits*, lo que implica que tenemos un gran número de combinación de señales que no estamos utilizando.

Un ejemplo es una instrucción que suma los valores de los registros de `A` y `LIT`, pero que guarde en ambos registros a la vez, esto implica que las dos señales de carga de los registros deben estar activas, las señales de esta instrucción son las siguiente:

Instrucción	Operandos	Condición	L_A	L_B	S_A0,1	S_B0,1	S_{Op}0,1,2	S_{Add}0,1	W
ADD	A,B		1	1	A	B	ADD	-	0

Pregunta 2: Análisis de Fragmentos de Código

Detalle, basándose en los nombres de las variables y etiquetas, la operación lógica o matemática que realizan los siguientes fragmentos de código desarrollados en el *Assembly* visto en clase:

(a)

```
DATA:
    var1 3
    var2 5
CODE:
    MOV B,120
    MOV A,B
    XOR A,(var2)
    SUB A,1
    SHR B,A
    MOV (var1),B
```

(b)

```
DATA:
    var1 5
    var2 3
    var3 0
CODE:
    MOV A, (var1)
    SHL A, A
    MOV (var3), A
    MOV A, (var2)
    MOV B, (var2)
    ADD A, B
    ADD A, B
    MOV B, (var3)
    ADD A, B
    MOV (var3), A
```

Solución:

(a) Como se puede notar en este código, este es un código secuencial bastante simple, el cual mueve el valor 120 al registro `B` para posteriormente, llevar este valor a `A` y realizar las siguientes operaciones: `XOR` con lo almacenado en `var2`, restarle 1 y hacer *shift right* para divir por dos el resultado.

El resultado de la operación se almacena en el registro `B` y posteriormente se guarda este en la dirección de la variable `var1`. La ejecución se detalla a continuación:

```
DATA:
    var1 3
    var2 5
CODE:
    MOV B,120       // B      = 01111000
    MOV A,B         // A      = 01111000
    XOR A,(var2)    // A      = 01111000 XOR 00000101 = 01111101
    SUB A,1         // A      = 01111100
    SHR B,A         // B      = 00111110
    MOV (var1),B    // (var1) = 00111110
```

(b) El programa calcula el valor `2*var1 + 3*var2` dejándolo guardado en la variable `var3`. Para ello, el programa calcula el valor de `2*var1` y lo guarda en `var3`, luego calcula `3*var2` y lo deja guardado en el registro `A`, por último se trae el valor guardado en `var3` al registro `B`, los suma y lo guarda en `var3`. La ejecución se detalla a continuación:

```
DATA:
    var1 5
    var2 3
    var3 0
CODE:
    // Calcular 2*var1
    MOV A, (var1)      // A      = 00000101 =  5
    SHL A, A           // A      = 00001010 = 10
    MOV (var3), A      // (var3) = 00001010 = 10
    // Calcular 3*var2
    MOV A, (var2)      // A      = 00000011 =  3
    MOV B, (var2)      // B      = 00000101 =  3
    ADD A, B           // A      = 00000110 =  6
    ADD A, B           // A      = 00001001 =  9
    // Sumar ambos y guardar en var3
    MOV B, (var3)      // B      = 00001010 = 10
    ADD A, B           // A      = 00010011 = 19
    MOV (var3), A      // (var3) = 00010011 = 19
```

Pregunta 3: Depuración de Cálculo Geométrico

El siguiente programa debe calcular el **perímetro** de un rectángulo alineado a los ejes, dados sus puntos inferior-izquierdo $(p1x, p1y)$ y superior-derecho $(p2x, p2y)$. El perímetro se define como $P = 2 \times ((x_2 - x_1) + (y_2 - y_1))$.

Sin embargo, el código contiene errores que impiden su correcto funcionamiento. Identifique estos y proponga la solución corregida.

```
DATA:
    p1x 2
    p1y 3
    p2x 8
    p2y 7
    peri 0
CODE:
    MOV A, p2x
    MOV B, p1x
    SUB A, B
    MOV (peri), A
    MOV A, p2y
    MOV B, p1y
    SUB A, B
    MOV B, (peri)
    ADD A, B
    SHR A, A
    MOV (peri), A
```

Solución:

El código presenta tres errores principales:

1. **Direccionamiento:** Se intenta mover la dirección de la etiqueta (ej. `p2x`) al registro

en lugar de su contenido `(p2x)`.

2. **Operación Incorrecta:** Para calcular el perímetro se requiere multiplicar la suma de los lados por 2. El código realiza un `SHR` (división por 2) en lugar de un `SHL`.

3. **Consistencia:** Se sobreescribe la variable `peri` antes de terminar el cálculo completo.

Código Corregido:

```
DATA:
    p1x 2
    p1y 3
    p2x 8
    p2y 7
    peri 0
CODE:
    MOV A, (p2x)
    MOV B, (p1x)
    SUB A, B
    MOV (peri), A    ; Guarda la
      base
    MOV A, (p2y)
    MOV B, (p1y)
    SUB A, B         ; Calcula
     altura en A
    MOV B, (peri)
    ADD A, B         ; A = base
     + altura
    SHL A, A         ; A = 2 * (
     base + altura)
    MOV (peri), A
```

Pregunta 4: Modificaciones al computador básico

A partir del diagrama del computador básico visto en clases, realice las modificaciones necesarias a la arquitectura para ejecutar las siguientes instrucciones e indique la combinación de señales que deben tener para ser ejecutadas en un ciclo de reloj:

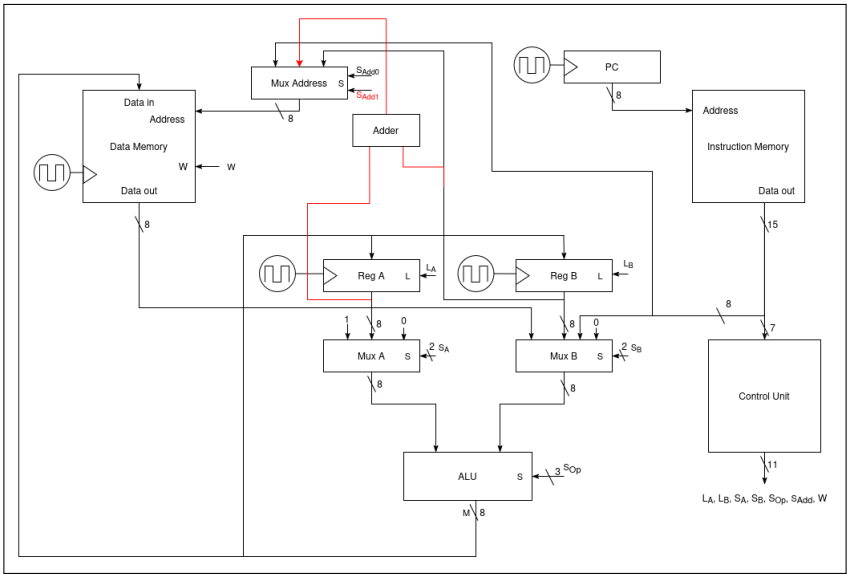
a) `MOV (A+B),B`: Direccionamiento indirecto por medio de registro base + registro índice.

b) (**I1 2025-2**) `SWAP A,B`: Almacena en el registro `A` el valor del registro `B`; y almacena en el registro `B` el valor del registro `A`, i.e. intercambia sus valores.

Solución:

a) En este caso se pide direccionamiento indirecto con registro base y registro índice. Para esto se debe conectar las salidas de los registros A y `B` a un `ADDER`, el cual a su vez se conecta con el `Mux Address`, note que el `Mux Address` aumenta en uno el tamaño de su señal de selección.

El diagrama que indica las modificaciones a realizar es es siguiente.



Instrucción	Operandos	Condición	L_A	L_B	S_A0,1	S_B0,1	S_{Op}0,1,2	S_{Add}0,1	W
MOV	(A+B),B	-	0	1	ZERO	B	ADD	A+B	1

b) Se añade el multiplexor `Mux A In` y su señal de selección `SAIn` para seleccionar el valor de entrada del registro `A`. Para la instrucción `SWAP A,B`, se seleccionará el valor del registro `B`; en cualquier otro caso, se seleccionará el valor proveniente de la **ALU**. Cabe destacar que se pudo haber implementado un cambio similar para el registro `B`, pero no es necesario para implementar la instrucción.

Instrucción	Operandos	Condición	L_A	L_B	S_A0,1	S_B0,1	S_{Op}0,1,2	S_{Add}0,1	W	SAIn
SWAP	A,B	-	1	1	A	ZERO	ADD	-	0	B

Computador básico sin saltos

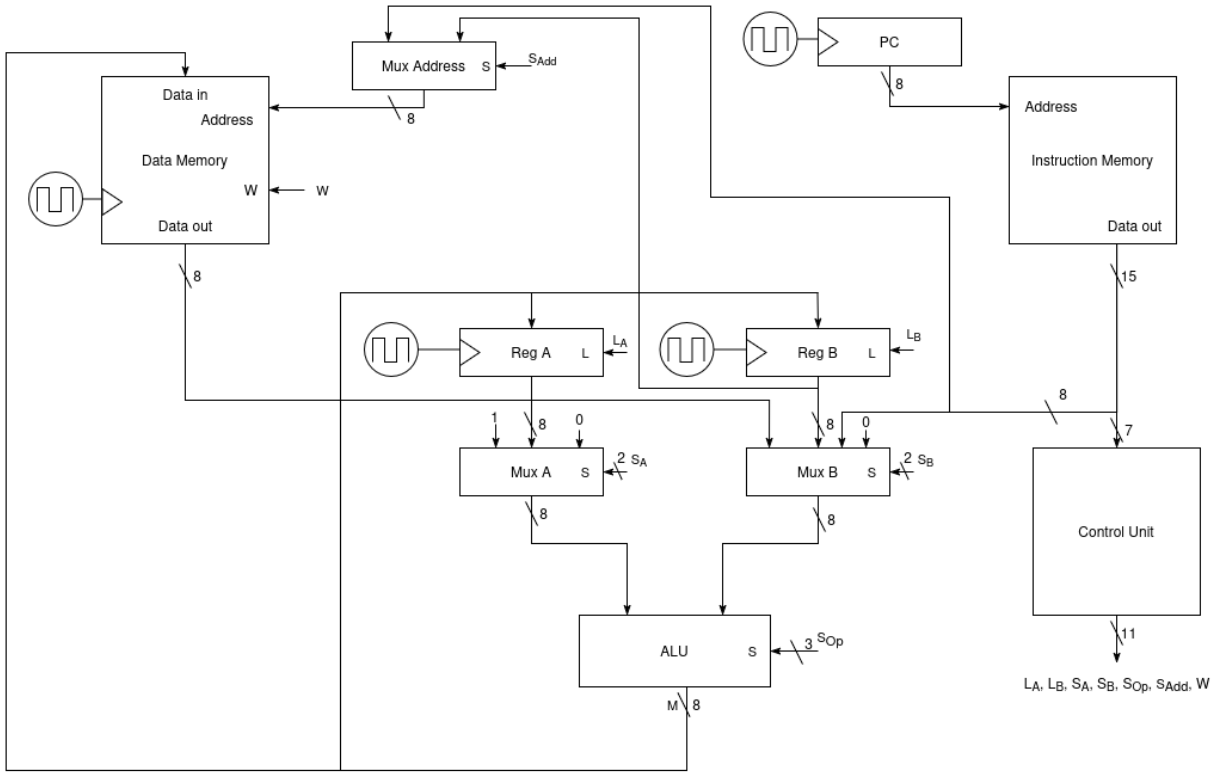


Tabla de instrucciones

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.

