Clase 16 - Programabilidad de circuitos digitales

IIC1001 - Algoritmos y Sistemas Computacionales

Cristian Ruz – cruz@uc.cl

Martes 05-Mayo-2026

Departamento de Ciencia de la Computación
Escuela de Ingeniería
Pontificia Universidad Católica de Chile

Contenidos

Contacto

Temas

Programabilidad

Clocks

Literales

Unidad de control

Contenidos

Contacto

Temas

Programabilidad

Clocks

Literales

Unidad de control

Contacto

🐐

vicente.cabra@uc.cl

Vicente Cabra

Ayudante jefe



🐸

alejandro.tapia@uc.cl

Alejandro Tapia

Ayudante jefe



🛼

gabriel.sandoval@uc.cl

Gabriel Sandoval

Ayudante jefe



Contenidos

Contacto

Temas

Programabilidad

Clocks

Literales

Unidad de control

Temas del curso

Sistemas computacionales

- Representación datos, números y codificación
- Funcionamiento hardware, procesadores y memoria.
- Funcionamiento de sistemas operativos
- Funcionamiento de Internet
- Herramientas computacionales: github + latex

Algoritmos

- Algoritmos y resolución de problemas
- Eficiencia algorítmica
- Estructuras secuenciales y ordenamiento
- Grafos y árboles

Próximos eventos

Domingo 3-Mayo: Cierre ETC (Evaluación Temprana)

- ¡Llegamos al 54%!
- Bonus general: +0.1 en midterm

Jueves 7-Mayo: Actividad de práctica (sin nota)

- Individual
- Programabilidad y assembler

Martes 26-Mayo: Interrogación 2

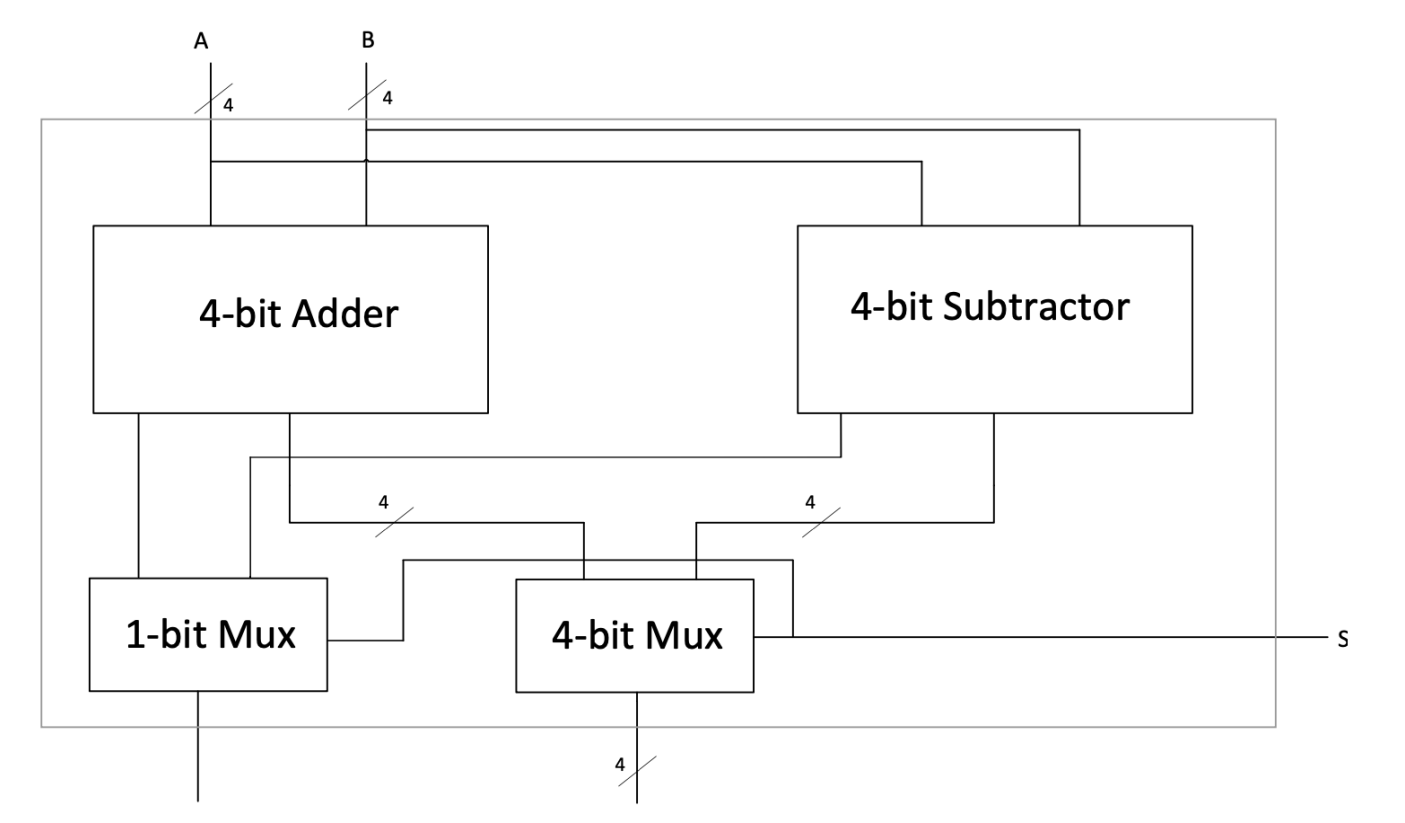
- 17:30

Unidad de selección

Se pueden combinar operaciones y seleccionar usando multiplexores.

Es una primera aproximación a una **unidad aritmético-lógica** (ALU)

$S = 0$: Suma, $S = 1$:, Resta



Sumador-restador de 4 bit de datos

Contenidos

Contacto

Temas

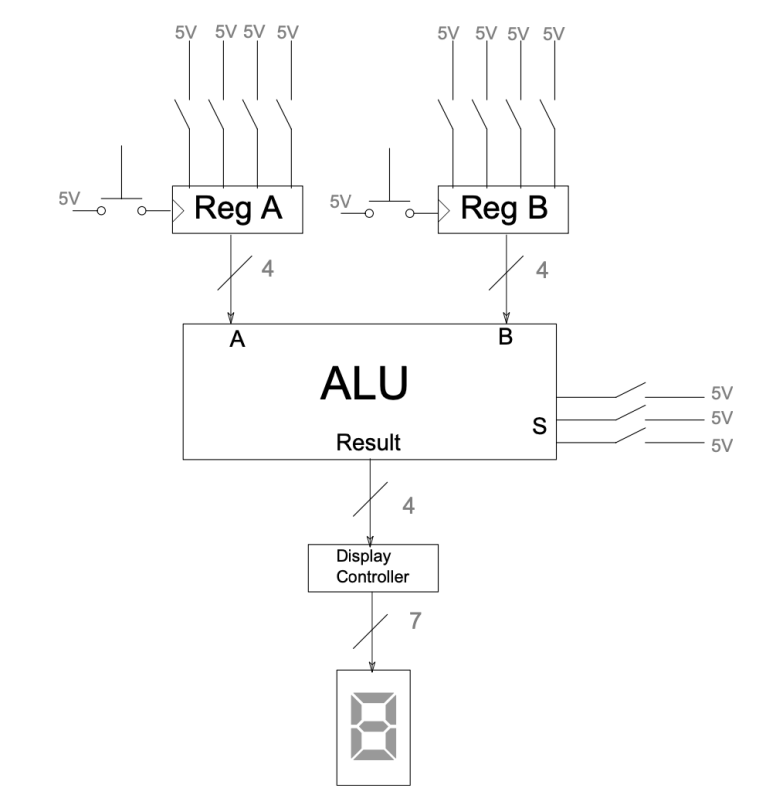
Programabilidad

Clocks

Literales

Unidad de control

Acumulación de operaciones

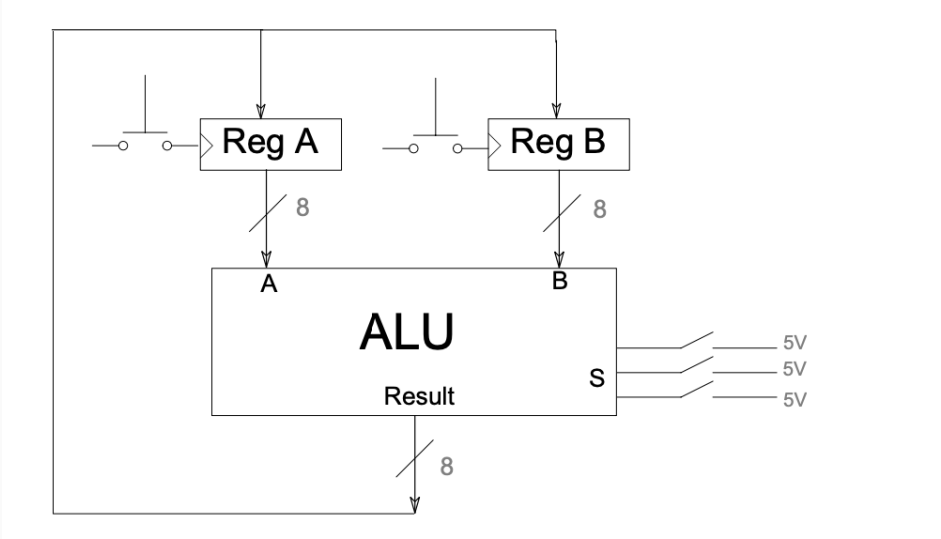


select	s2	s1	s0	operación
0	0	0	0	Suma
1	0	0	1	Resta
2	0	1	0	And
3	0	1	1	Or
4	1	0	0	Not A
5	1	0	1	Xor
6	1	1	0	Shift Left A
7	1	1	1	Shift Right A

Calculador de 4 bits con ingreso de usuario y display

- RegA y RegB son **registros**. Almacenan bits.
- ALU es *Arithmetic Logic Unit*. Ejecuta operaciones entre A y B de acuerdo a S.
- La tabla indica el **set de instrucciones** que acepta este "computador".
- Los *display controller* muestran el resultado usando una representación visual.

Acumulación de operaciones

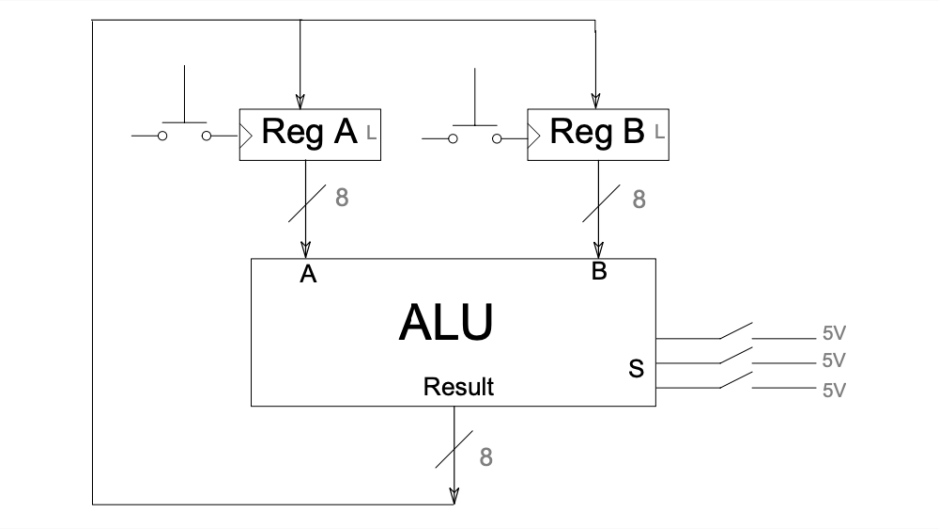


select	s2	s1	s0	operación
0	0	0	0	Suma
1	0	0	1	Resta
2	0	1	0	And
3	0	1	1	Or
4	1	0	0	Not A
5	1	0	1	Xor
6	1	1	0	Shift Left A
7	1	1	1	Shift Right A

Calculadora de 8 bits con **registros acumuladores**

- El resultado se almacena, al final de cada ejecución, en RegA y RegB.
- La siguiente ejecución usa los resultados actualizados de RegA y RegB.
- RegA y RegB son **registros acumuladores**.
- La tabla indica el **set de instrucciones** que acepta este “computador”.

Acumulación de operaciones

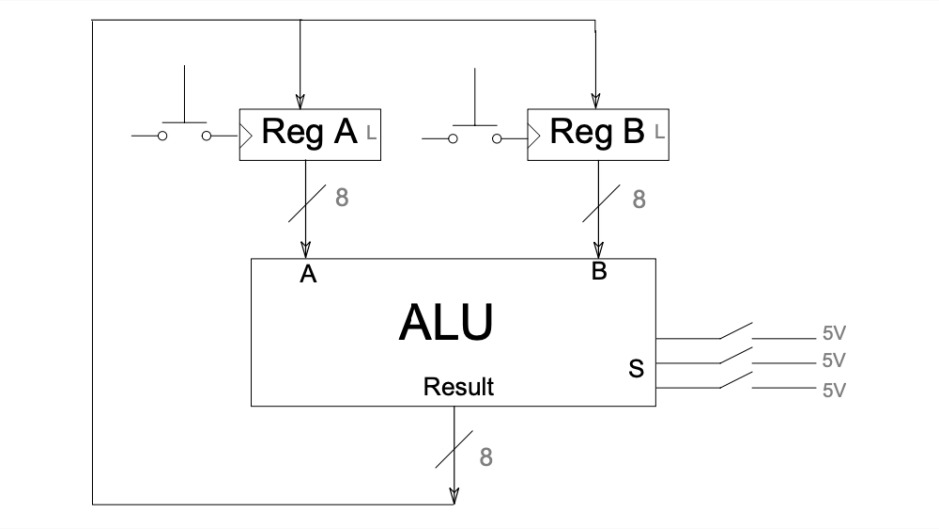


la	lb	s2	s1	s0	operación
1	0	0	0	0	A=A+B
0	1	0	0	0	B=A+B
1	0	0	0	1	A=A-B
0	1	0	0	1	B=A-B
1	0	0	1	0	A=A and B
0	1	0	1	0	B=A and B
1	0	0	1	1	A=A or B
0	1	0	1	1	B=A or B
1	0	1	0	0	A=notA
0	1	1	0	0	B=notA
1	0	1	0	1	A=A xor B
0	1	1	0	1	B=A xor B
1	0	1	1	0	A=shift left A
0	1	1	1	0	B=shift left A
1	0	1	1	1	A=shift right A
0	1	1	1	1	B=shift right A

Calculadora de 8 bits con **registros acumuladores** y señales de carga

- Los bits l_a y l_b son **señales de carga**
- Si están en 1, el registro correspondiente se actualiza. Si no, queda igual.
- El resultado se almacena en RegA o en RegB, dependiendo de l_a y l_b
- La siguiente ejecución usa los resultados actualizados de RegA y RegB.
- El **set de instrucciones** acepta más combinaciones ahora.

Acumulación de operaciones

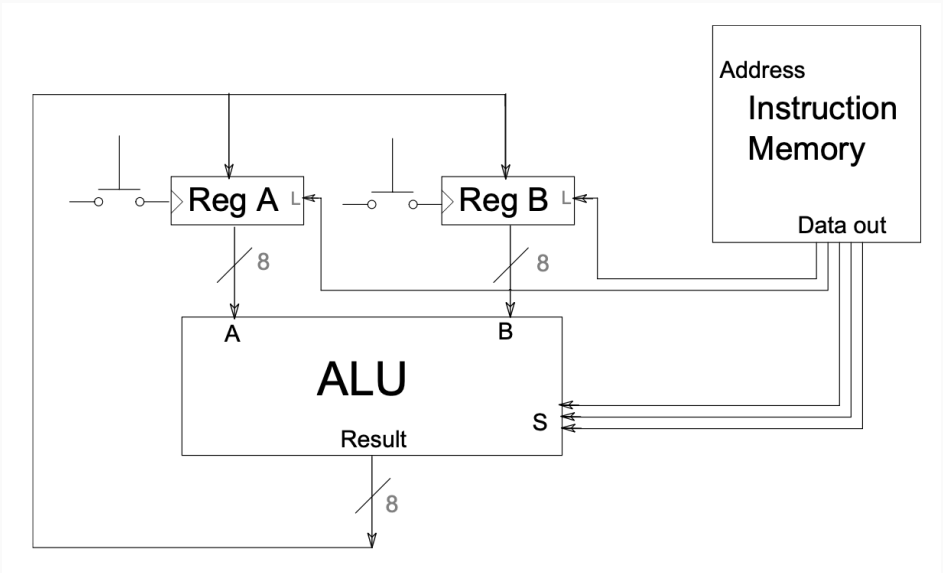


la	lb	s2	s1	s0	operación
1	0	0	0	0	A=A+B
0	1	0	0	0	B=A+B
1	0	0	0	1	A=A-B
0	1	0	0	1	B=A-B
1	0	0	1	0	A=A and B
0	1	0	1	0	B=A and B
1	0	0	1	1	A=A or B
0	1	0	1	1	B=A or B
1	0	1	0	0	A=notA
0	1	1	0	0	B=notA
1	0	1	0	1	A=A xor B
0	1	1	0	1	B=A xor B
1	0	1	1	0	A=shift left A
0	1	1	1	0	B=shift left A
1	0	1	1	1	A=shift right A
0	1	1	1	1	B=shift right A

la	lb	s2	s1	s0	operación	A	B
0	0	-	-	-	-	**0**	1
1	0	0	0	0	A=A+B	**1**	1
0	1	0	0	0	B=A+B	1	**2**
1	0	0	0	0	A=A+B	**3**	2
0	1	0	0	0	B=A+B	3	**5**
1	0	0	0	0	A=A+B	**8**	5
0	1	0	0	0	B=A+B	8	**13**

- Podemos especificar un **programa** como una secuencia de instrucciones, cada una compuesta por 5 bits: S, l_a y l_b.

Almacenamiento de instrucciones

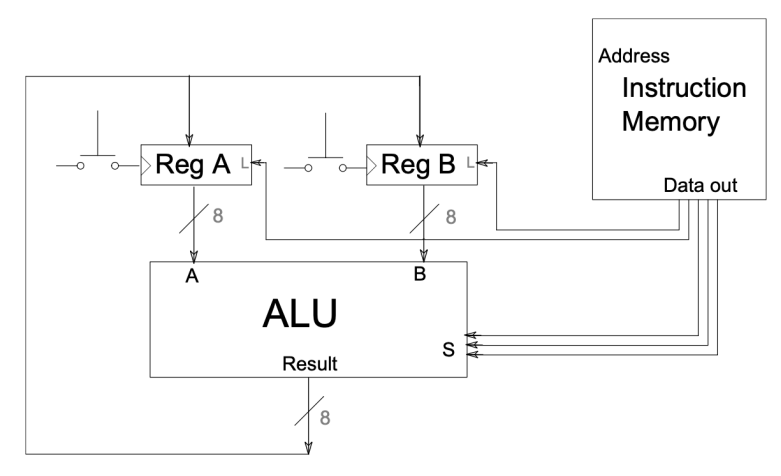


la	lb	s2	s1	s0	operación
1	0	0	0	0	A=A+B
0	1	0	0	0	B=A+B
1	0	0	0	1	A=A-B
0	1	0	0	1	B=A-B
1	0	0	1	0	A=A and B
0	1	0	1	0	B=A and B
1	0	0	1	1	A=A or B
0	1	0	1	1	B=A or B
1	0	1	0	0	A=notA
0	1	1	0	0	B=notA
1	0	1	0	1	A=A xor B
0	1	1	0	1	B=A xor B
1	0	1	1	0	A=shift left A
0	1	1	1	0	B=shift left A
1	0	1	1	1	A=shift right A
0	1	1	1	1	B=shift right A

Un **programa** se almacena usando una **memoria de instrucciones**

- En cada ciclo de ejecución, se leen los 5 bit de cada instrucción del programa desde la **memoria de instrucciones** y se ejecuta de acuerdo al **set de instrucciones**

Almacenamiento de instrucciones



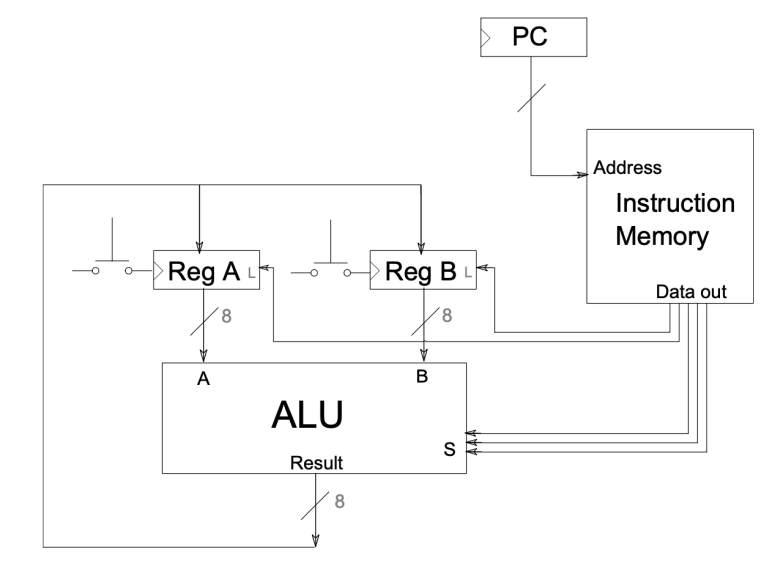
la	lb	s2	s1	s0	operación
1	0	0	0	0	A=A+B
0	1	0	0	0	B=A+B
1	0	0	0	1	A=A-B
0	1	0	0	1	B=A-B
1	0	0	1	0	A=A and B
0	1	0	1	0	B=A and B
1	0	0	1	1	A=A or B
0	1	0	1	1	B=A or B
1	0	1	0	0	A=notA
0	1	1	0	0	B=notA
1	0	1	0	1	A=A xor B
0	1	1	0	1	B=A xor B
1	0	1	1	0	A=shift left A
0	1	1	1	0	B=shift left A
1	0	1	1	1	A=shift right A
0	1	1	1	1	B=shift right A

la	lb	s2	s1	s0	operación	A	B
0	0	-	-	-	-	**0**	1
1	0	0	0	0	A=A+B	**1**	1
0	1	0	0	0	B=A+B	1	**2**
1	0	0	0	0	A=A+B	**3**	2
0	1	0	0	0	B=A+B	3	**5**
1	0	0	0	0	A=A+B	**8**	5
0	1	0	0	0	B=A+B	8	**13**

dirección	instrucción
0000	10000
0001	01000
0010	10000
0011	01000
0100	10000
0101	01000

- Cada línea de la **memoria de instrucciones** tiene un número secuencial. Esto se llama **dirección de memoria**.
- En este caso una dirección de memoria tiene 4 bit. Podría haber hasta $2^4 = 16$ instrucciones.
- En cada **dirección de memoria** hay 5 bit que contienen la instrucción correspondiente del programa.

Almacenamiento de instrucciones



la	lb	s2	s1	s0	operación
1	0	0	0	0	A=A+B
0	1	0	0	0	B=A+B
1	0	0	0	1	A=A-B
0	1	0	0	1	B=A-B
1	0	0	1	0	A=A and B
0	1	0	1	0	B=A and B
1	0	0	1	1	A=A or B
0	1	0	1	1	B=A or B
1	0	1	0	0	A=notA
0	1	1	0	0	B=notA
1	0	1	0	1	A=A xor B
0	1	1	0	1	B=A xor B
1	0	1	1	0	A=shift left A
0	1	1	1	0	B=shift left A
1	0	1	1	1	A=shift right A
0	1	1	1	1	B=shift right A

program counter	instrucción	operación
0000	10000	A=A+B
0001	01000	B=A+B
0010	10000	A=A+B
0011	01000	B=A+B
0100	10000	A=A+B
0101	01000	B=A+B

dirección	instrucción
0000	10000
0001	01000
0010	10000
0011	01000
0100	10000
0101	01000

Registro **Programa Counter (PC)** indica en la instrucción a ejecutar

- El PC se inicializa en 0000. La primera instrucción que se ejecuta será 10000.
- Al final de cada ciclo, el PC se incrementa en 1.

Contenidos

Contacto

Temas

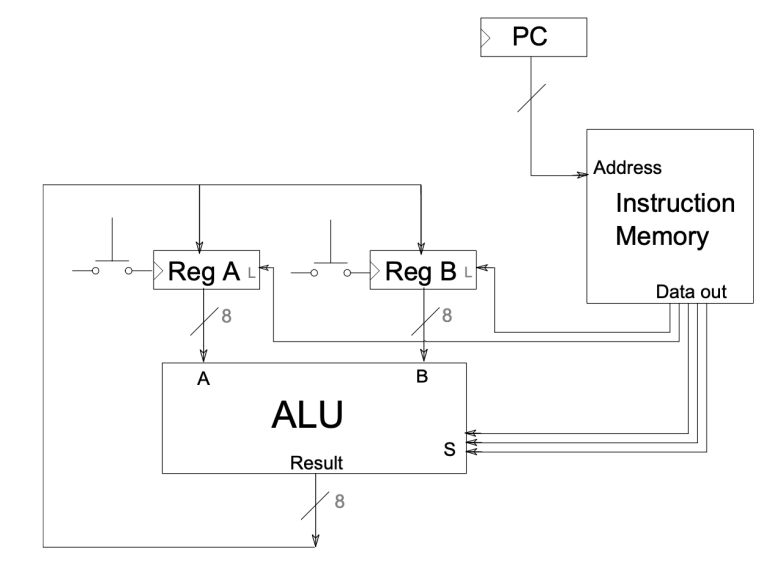
Programabilidad

Clocks

Literales

Unidad de control

Ejecución automática



la	lb	s2	s1	s0	operación
1	0	0	0	0	A=A+B
0	1	0	0	0	B=A+B
1	0	0	0	1	A=A-B
0	1	0	0	1	B=A-B
1	0	0	1	0	A=A and B
0	1	0	1	0	B=A and B
1	0	0	1	1	A=A or B
0	1	0	1	1	B=A or B
1	0	1	0	0	A=notA
0	1	1	0	0	B=notA
1	0	1	0	1	A=A xor B
0	1	1	0	1	B=A xor B
1	0	1	1	0	A=shift left A
0	1	1	1	0	B=shift left A
1	0	1	1	1	A=shift right A
0	1	1	1	1	B=shift right A

program counter	instrucción	operación
0000	10000	A=A+B
0001	01000	B=A+B
0010	10000	A=A+B
0011	01000	B=A+B
0100	10000	A=A+B
0101	01000	B=A+B

dirección	instrucción
0000	10000
0001	01000
0010	10000
0011	01000
0100	10000
0101	01000

Un **clock** emite pulso a intervalos muy regulares.

- Cada pulso gatilla la ejecución de un ciclo.
- En cada ciclo se ejecuta 1 instrucción.
- Si el *clock* genera $1000000000 = 10^9 = 1G$ pulsos por segundo, entonces en cada segundo se pueden ejecutar 10^9 instrucciones: $1GHz$.

Contenidos

Contacto

Temas

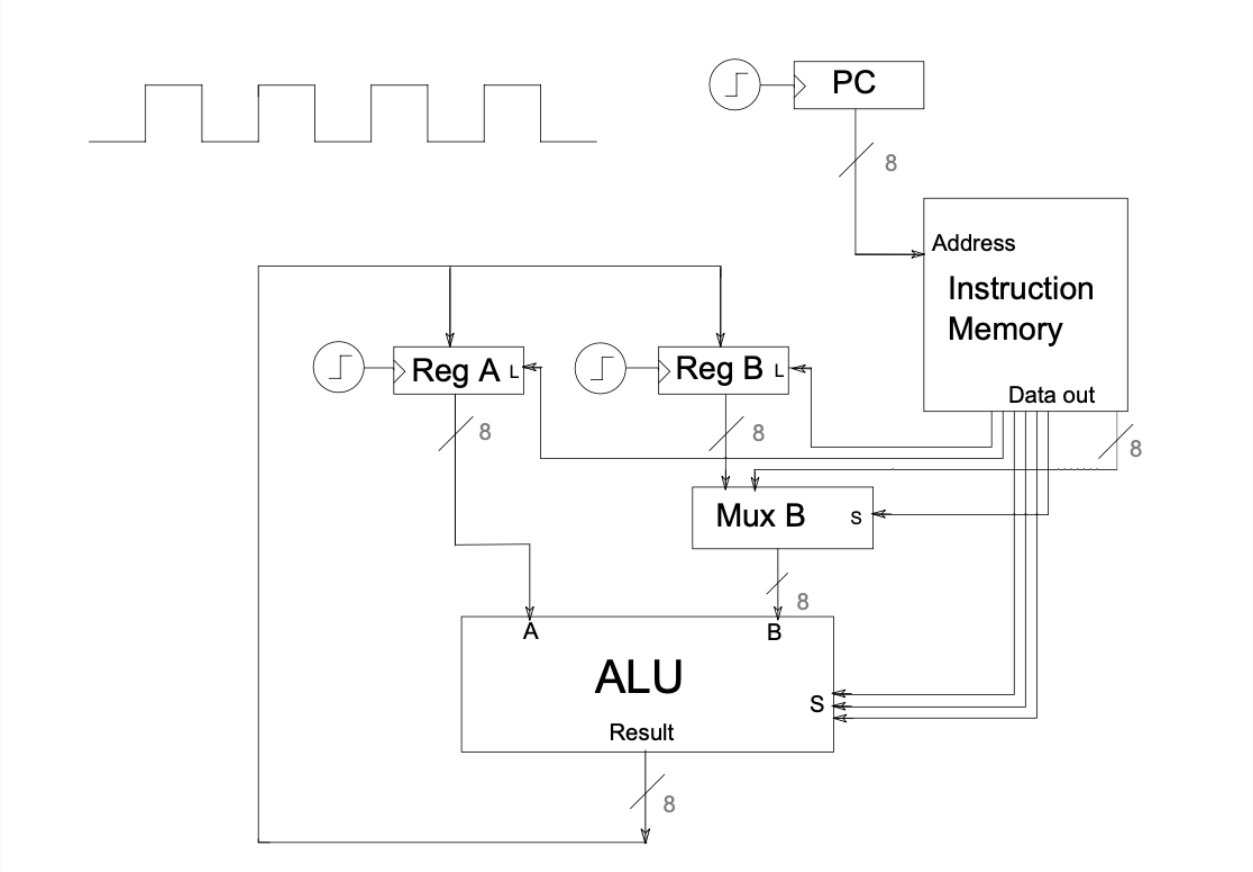
Programabilidad

Clocks

Literales

Unidad de control

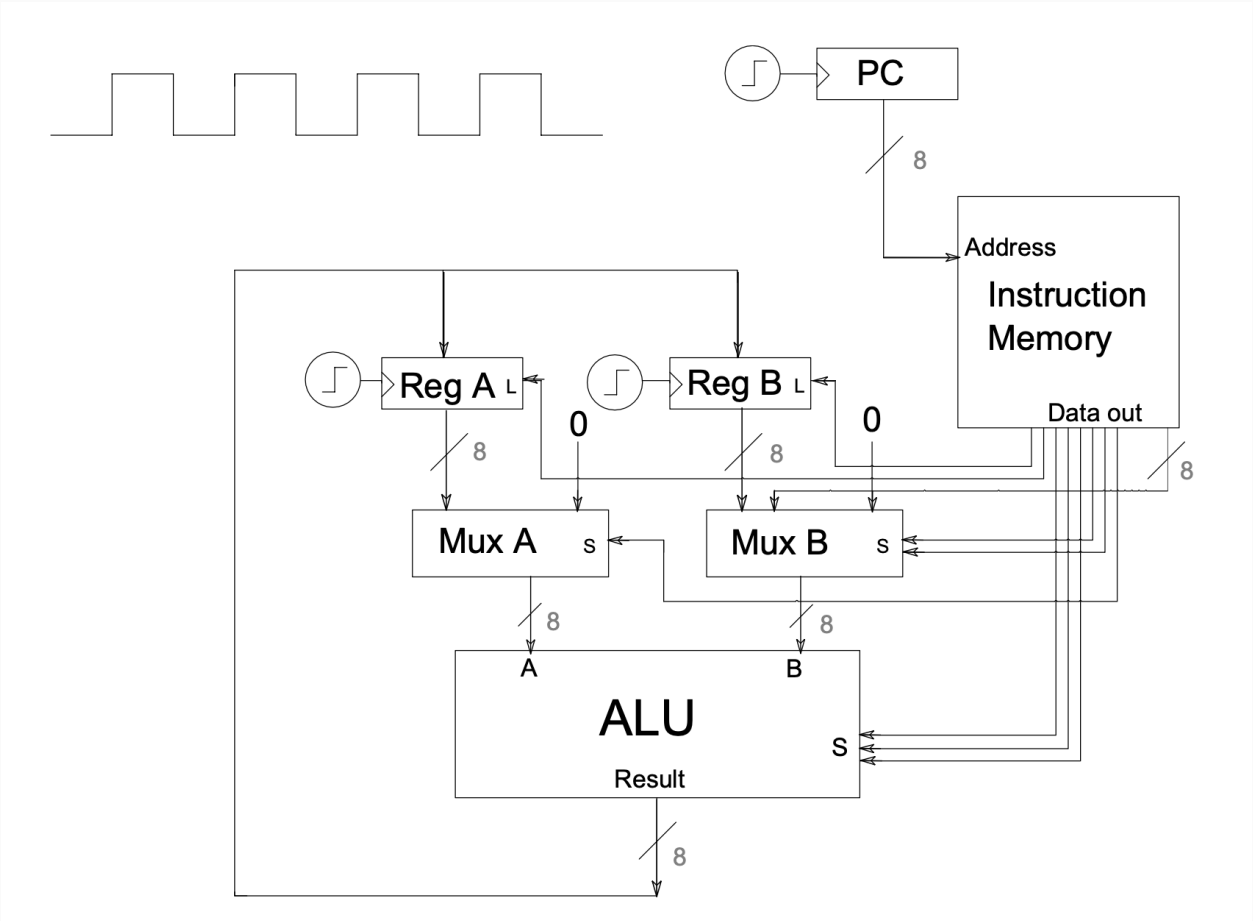
Operaciones con literales



Computador simple con carga de literales

- Literales son constantes: valores fijos.
- Operaciones: $A = A + literal$.

Carga de literales



Computador simple con carga de literales

- Literales son constantes: valores fijos.
- Operaciones: $A = A + \texttt{0}$.

Contenidos

Contacto

Temas

Programabilidad

Clocks

Literales

Unidad de control

Unidad control

La	Lb	Sa0	Sb0	Sb1	Sop2	Sop1	Sop0	Operación
1	0	1	0	0	0	0	0	A=B
0	1	0	1	1	0	0	0	B=A
1	0	1	0	1	0	0	0	A=Lit
0	1	1	0	1	0	0	0	B=Lit
1	0	0	0	0	0	0	0	A=A+B
0	1	0	0	0	0	0	0	B=A+B
1	0	0	0	1	0	0	0	A=A+Lit
1	0	0	0	0	0	0	1	A=A-B
0	1	0	0	0	0	0	1	B=A-B
1	0	0	0	1	0	0	1	A=A-Lit
1	0	0	0	0	0	1	0	A=A and B
0	1	0	0	0	0	1	0	B=A and B
1	0	0	0	1	0	1	0	A=A and Lit
1	0	0	0	0	0	1	1	A=A or B
0	1	0	0	0	0	1	1	B=A or B
1	0	0	0	1	0	1	1	A=A or Lit
1	0	0	0	0	1	0	0	A=notA
0	1	0	0	0	1	0	0	B=notA
1	0	0	0	1	1	0	0	A=notLit
1	0	0	0	0	1	0	1	A=A xor B
0	1	0	0	0	1	0	1	B=A xor B
1	0	0	0	1	1	0	1	A=A xor Lit
1	0	0	0	0	1	1	0	A=shift left A
0	1	0	0	0	1	1	0	B=shift left A
1	0	0	0	0	1	1	1	A=shift right A
0	1	0	0	0	1	1	1	B=shift right A

Unidad de control

- **Unidad de control:** contiene todas las instrucciones posibles, y señales de

Unidad control

Opcode	La	Lb	Sa0	Sb0	Sb1	Sop2	Sop1	Sop0	Operación
000000	1	0	1	0	0	0	0	0	A=B
000001	0	1	0	1	1	0	0	0	B=A
000010	1	0	1	0	1	0	0	0	A=Lit
000011	0	1	1	0	1	0	0	0	B=Lit
000100	1	0	0	0	0	0	0	0	A=A+B
000101	0	1	0	0	0	0	0	0	B=A+B
000110	1	0	0	0	1	0	0	0	A=A+Lit
000111	1	0	0	0	0	0	0	1	A=A-B
001000	0	1	0	0	0	0	0	1	B=A-B
001001	1	0	0	0	1	0	0	1	A=A-Lit
001010	1	0	0	0	0	0	1	0	A=A and B
001011	0	1	0	0	0	0	1	0	B=A and B
001100	1	0	0	0	1	0	1	0	A=A and Lit
001101	1	0	0	0	0	0	1	1	A=A or B
001110	0	1	0	0	0	0	1	1	B=A or B
001111	1	0	0	0	1	0	1	1	A=A or Lit
010000	1	0	0	0	0	1	0	0	A=notA
010001	0	1	0	0	0	1	0	0	B=notA
010010	1	0	0	0	0	1	0	1	A=A xor B
010011	0	1	0	0	0	1	0	1	B=A xor B
010100	1	0	0	0	1	1	0	1	A=A xor Lit
010101	1	0	0	0	0	1	1	0	A=shift left A
010110	0	1	0	0	0	1	1	0	B=shift left A
010111	1	0	0	0	0	1	1	1	A=shift right A
011000	0	1	0	0	0	1	1	1	B=shift right A

OPCODE

- Número que permite indexar (identificar) instrucciones

Unidad control

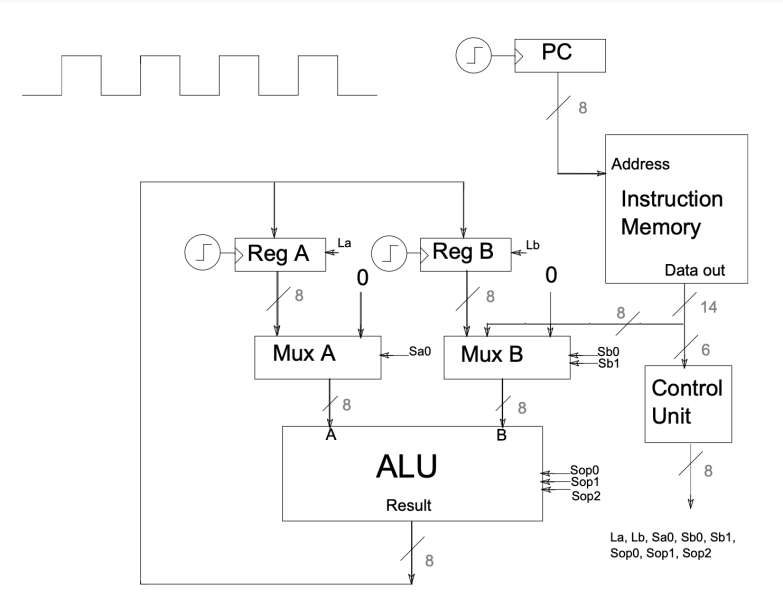
Instrucción	Opcode	La	Lb	Sa0	Sb0	Sb1	Sop2	Sop1	Sop0	Operación
MOVAB	000000	1	0	1	0	0	0	0	0	A=B
MOVBA	000001	0	1	0	1	1	0	0	0	B=A
MOVAL	000010	1	0	1	0	1	0	0	0	A=Lit
MOVBL	000011	0	1	1	0	1	0	0	0	B=Lit
ADDA	000100	1	0	0	0	0	0	0	0	A=A+B
ADDB	000101	0	1	0	0	0	0	0	0	B=A+B
ADDL	000110	1	0	0	0	1	0	0	0	A=A+Lit
SUBA	000111	1	0	0	0	0	0	0	1	A=A-B
SUBB	001000	0	1	0	0	0	0	0	1	B=A-B
SUBL	001001	1	0	0	0	1	0	0	1	A=A-Lit
ANDA	001010	1	0	0	0	0	0	1	0	A=A and B
ANDB	001011	0	1	0	0	0	0	1	0	B=A and B
ANDL	001100	1	0	0	0	1	0	1	0	A=A and Lit
ORA	001101	1	0	0	0	0	0	1	1	A=A or B
ORB	001110	0	1	0	0	0	0	1	1	B=A or B
ORL	001111	1	0	0	0	1	0	1	1	A=A or Lit
NOTA	010000	1	0	0	0	0	1	0	0	A=notA
NOTB	010001	0	1	0	0	0	1	0	0	B=notA
XORA	010010	1	0	0	0	0	1	0	1	A=A xor B
XORB	010011	0	1	0	0	0	1	0	1	B=A xor B
XORL	010100	1	0	0	0	1	1	0	1	A=A xor Lit
SHLA	010101	1	0	0	0	0	1	1	0	A=shift left A
SHLB	010110	0	1	0	0	0	1	1	0	B=shift left A
SHRA	010111	1	0	0	0	0	1	1	1	A=shift right A
SHRB	011000	0	1	0	0	0	1	1	1	B=shift right A

ASSEMBLER

- Para el programar es difícil recordar números de instrucción.
- Se usan nombres simbólicos (*assembler*) para cada instrucción.
- Instrucción *assembler* se traduce a OPCODE y luego a señales.

Unidad control

Instrucción	Opcode	La	Lb	Sa0	Sb0	Sb1	Sop2	Sop1	Sop0	Operación
MOVAB	000000	1	0	1	0	0	0	0	0	A=B
MOVBA	000001	0	1	0	1	1	0	0	0	B=A
MOVAL	000010	1	0	1	0	1	0	0	0	A=Lit
MOVBL	000011	0	1	1	0	1	0	0	0	B=Lit
ADDA	000100	1	0	0	0	0	0	0	0	A=A+B
ADDB	000101	0	1	0	0	0	0	0	0	B=A+B
ADDL	000110	1	0	0	0	1	0	0	0	A=A+Lit
SUBA	000111	1	0	0	0	0	0	0	1	A=A-B
SUBB	001000	0	1	0	0	0	0	0	1	B=A-B
SUBL	001001	1	0	0	0	1	0	0	1	A=A-Lit
ANDA	001010	1	0	0	0	0	0	1	0	A=A and B
ANDB	001011	0	1	0	0	0	0	1	0	B=A and B
ANDL	001100	1	0	0	0	1	0	1	0	A=A and Lit
ORA	001101	1	0	0	0	0	0	1	1	A=A or B
ORB	001110	0	1	0	0	0	0	1	1	B=A or B
ORL	001111	1	0	0	0	1	0	1	1	A=A or Lit
NOTA	010000	1	0	0	0	0	1	0	0	A=notA
NOTB	010001	0	1	0	0	0	1	0	0	B=notA
XORA	010010	1	0	0	0	0	1	0	1	A=A xor B
XORB	010011	0	1	0	0	0	1	0	1	B=A xor B
XORL	010100	1	0	0	0	1	1	0	1	A=A xor Lit
SHLA	010101	1	0	0	0	0	1	1	0	A=shift left A
SHLB	010110	0	1	0	0	0	1	1	0	B=shift left A
SHRA	010111	1	0	0	0	0	1	1	1	A=shift right A
SHRB	011000	0	1	0	0	0	1	1	1	B=shift right A



Computador básico utiliza unidad de control