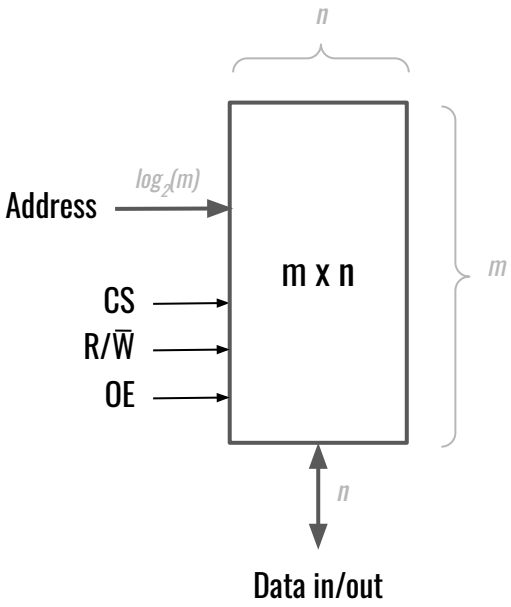
CLASE 2

BUSES
ARQUITECTURA DE VON NEUMANN
EL CICLO DE INSTRUCCIÓN

Bibliografía para las próximas clases

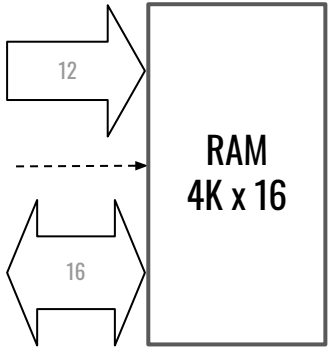
Linda Null - Essentials of Computer Organization and Architecture (1a ed. 2003)
Capítulo 4: MARIE: An Introduction to a Simple Computer
Capítulo 5: A Closer Look at Instruction Set Architectures

IMPORTANTE
Capítulo 1: Introduction
Capítulo 2: Data Representation in Computer Systems
Capítulo 3: Boolean Algebra and Digital Logic
Comprobar que no haya dudas, revisión de conceptos al final de cada capítulo y ejercicios.



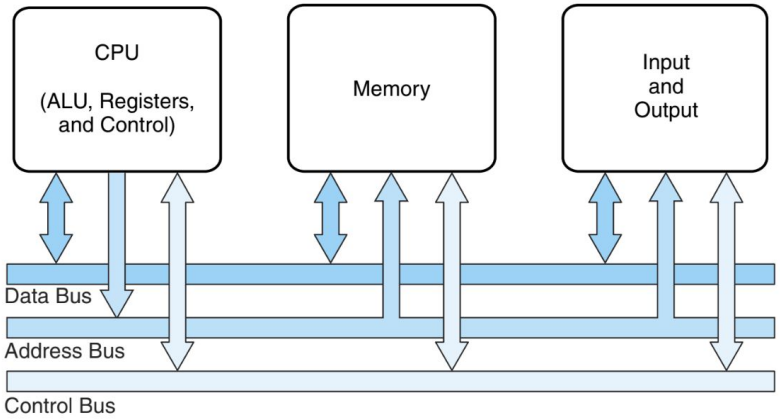
Modelo abstracto de una memoria (m x n)

- Arreglo de **m** celdas consecutivas, cada una capaz de almacenar **n** bits.
- Cada una de las **m** celdas tiene una identificación única **(dirección)**, por medio de la cual puede ser accedida para almacenar o recuperar información de la misma.
- La dirección está codificada en binario, utilizando **$\log_2(m)$** bits.
- Los **n** bits contenidos en cada celda **(datos)** se leen o escriben simultáneamente.
- Sólo una de las **m** direcciones puede ser accedida por turno (para lectura o escritura).

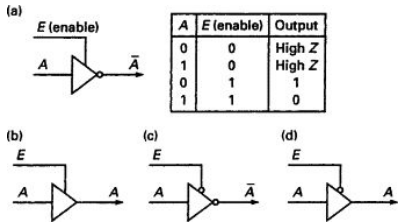


Ejemplo
4 chips TMS4016 (2K x 8)

Buses compartidos

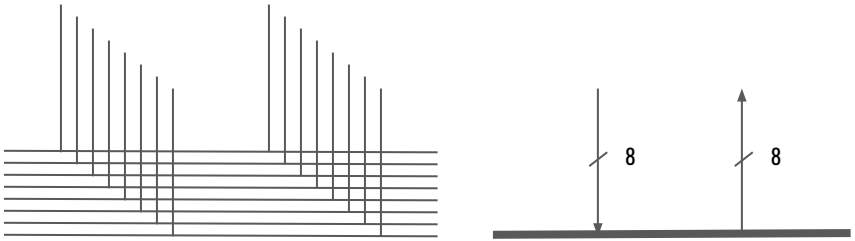


Conjunto de conexiones que sirven como vía compartida para interconectar diferentes subsistemas. Múltiples líneas que permiten mover la información en paralelo.
Opción versátil y de bajo costo que facilita la expansión. Pero representa una restricción (*bottleneck*): sólo una comunicación puede realizarse en cada instante.
La velocidad del bus se ve afectada por la distancia que debe recorrer la señal y por la cantidad de dispositivos conectados. Los dispositivos pueden ser maestro (*master*) o esclavo (*slave*). En nuestro caso, en principio, sólo la CPU es master.
Protocolo: reglas de uso. Bus cycle. Datos, direcciones, control y power.
Procesador-memoria y de entrada-salida (I/O bus). Jerarquía.

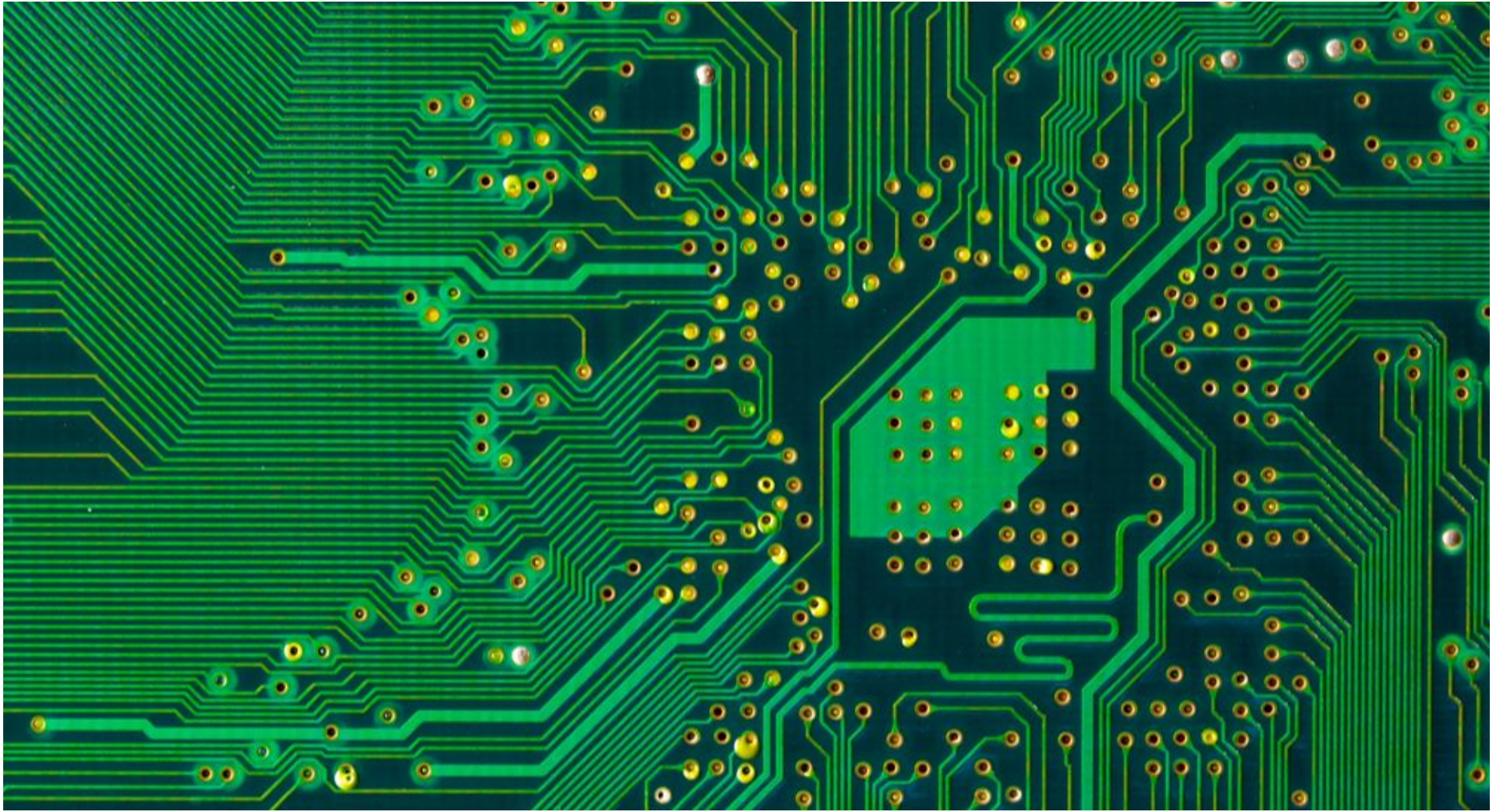


A	E (enable)	Output
0	0	High Z
1	0	High Z
0	1	1
1	1	0

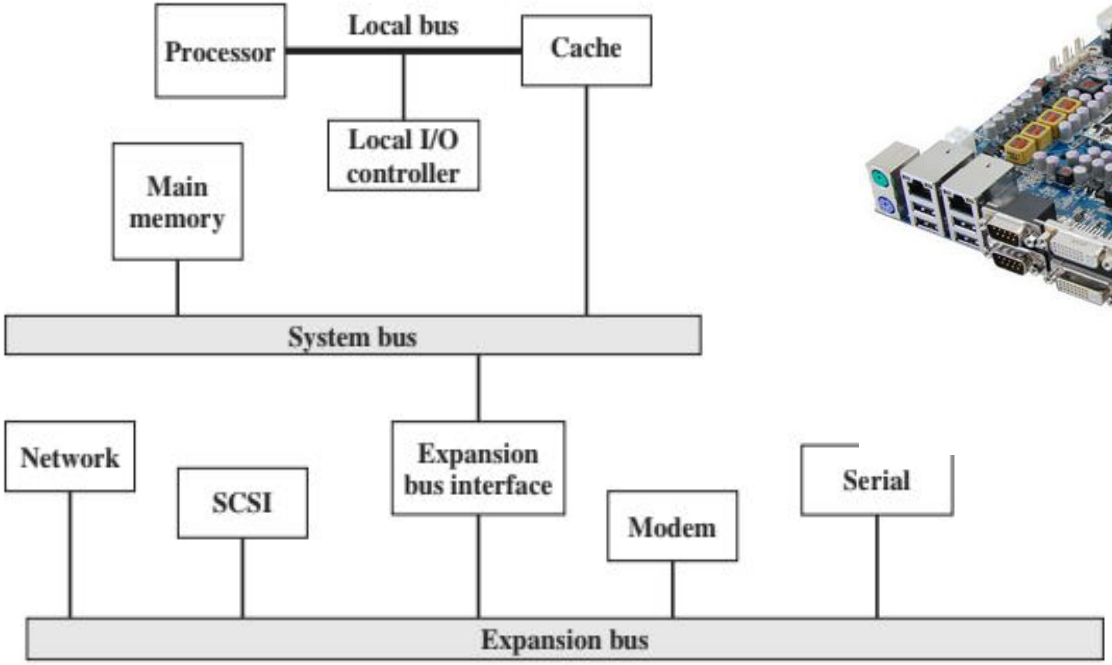
Three-state buffers



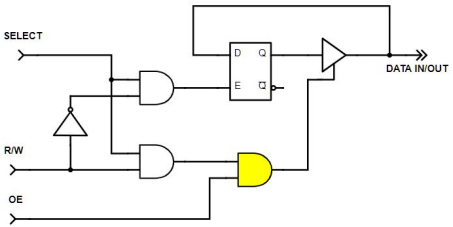
8-bit data bus

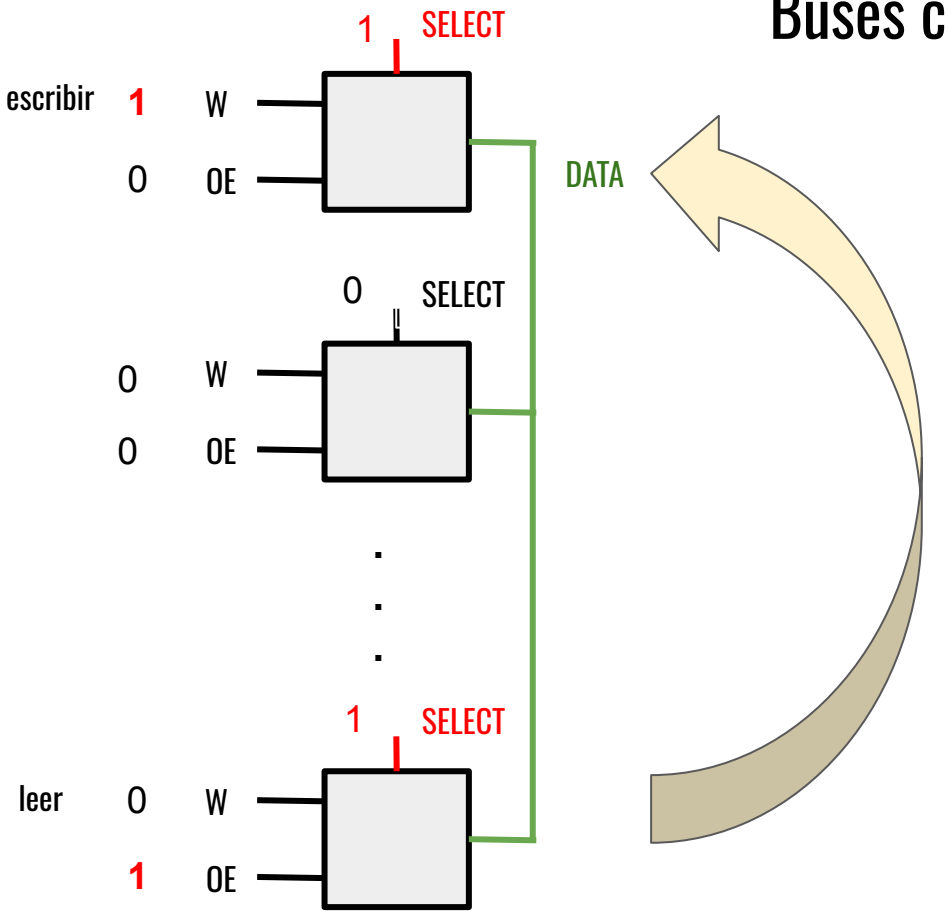


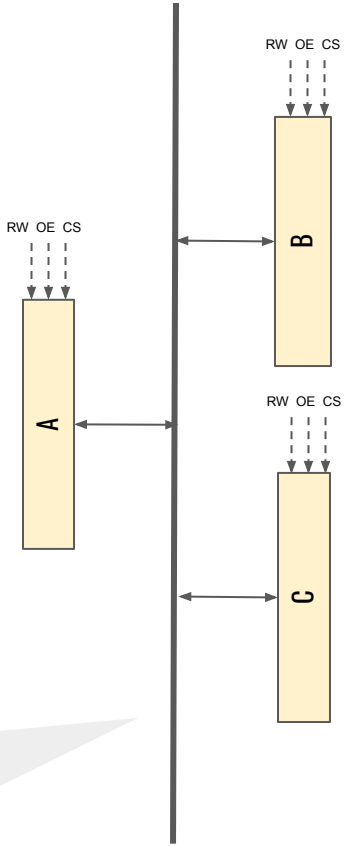
Buses compartidos



Buses compartidos



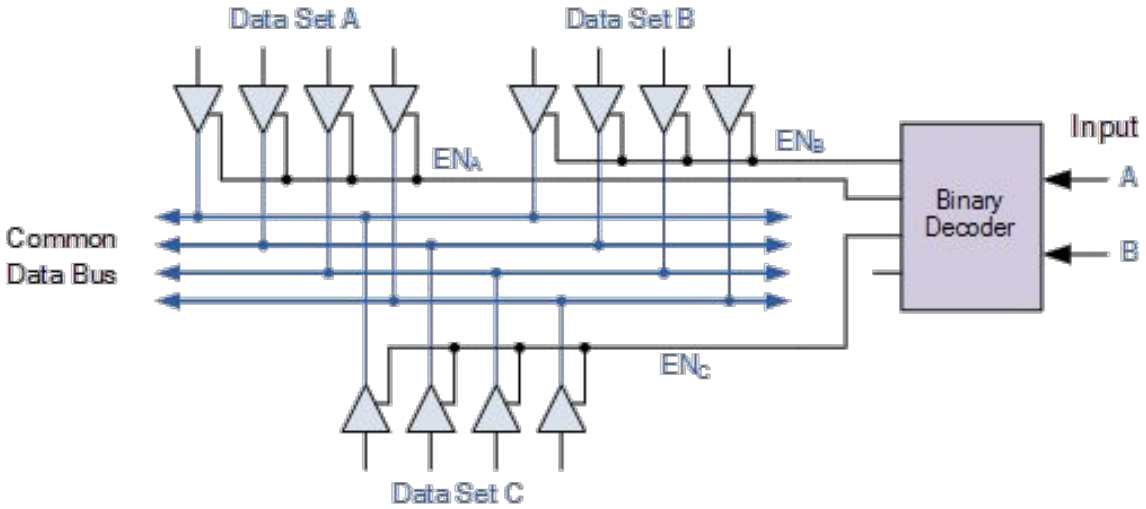




Las líneas de control están conectadas a todos los FF del registro

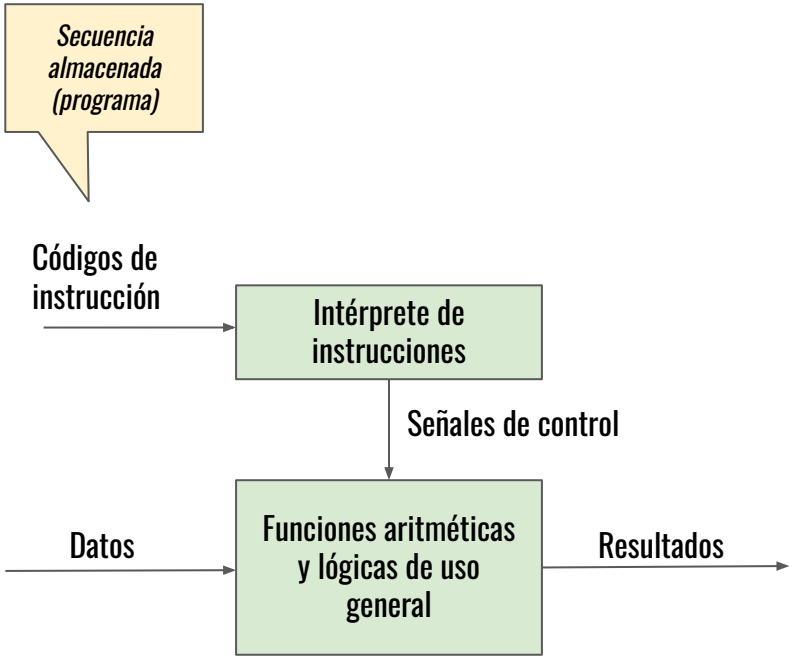
Las líneas de datos están conectadas “en paralelo” entre registros. Se comparte la entrada y la salida.

Buses compartidos



https://www.electronics-tutorials.ws/logic/logic_9.html
Repaso de Fan Out, Tri-state Buffer, Data Bus Control

El modelo de von Neumann

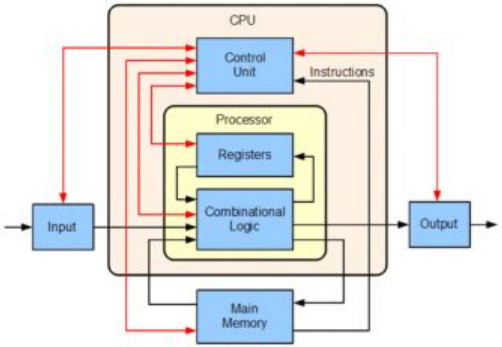


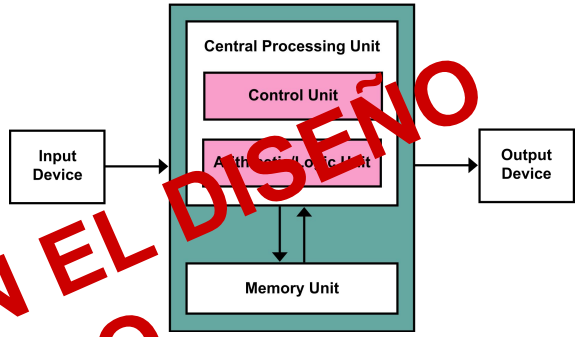
CONCEPTO DE PROGRAMA ALMACENADO

- Las instrucciones se representan en binario, al igual que los argumentos (ver sistemas de representación).
- Los programas (secuencia de instrucciones) se almacenan en memoria, al igual que los datos.

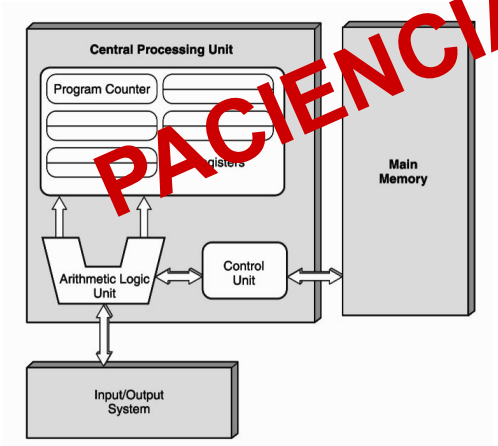
ELEMENTOS PRINCIPALES

- **Unidad central de procesamiento** (CPU):
  - Unidad de control, fetch-decode-execute secuencial.
  - Datapath: ALU (unidad aritmético-lógica) y registros.
- **Unidad de memoria**, almacena las instrucciones y sus argumentos.
- **Unidad de entrada-salida**, interacción con dispositivos externos.





PACIENCIA CON EL DISEÑO GRÁFICO



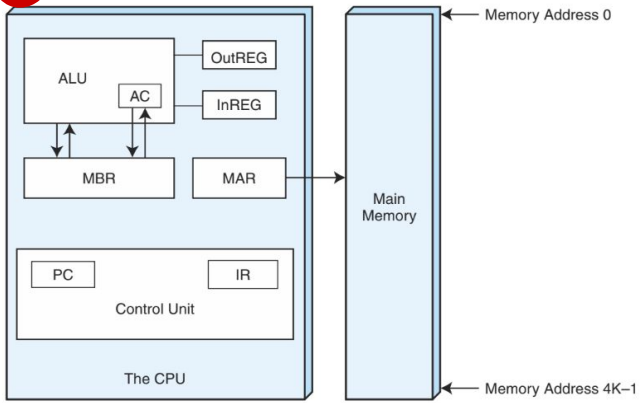


FIGURE 4.8 **MARIE's Architecture**

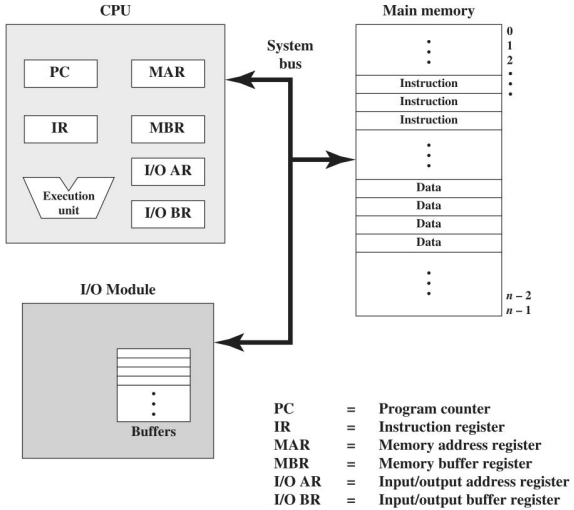


Figure 3.2 Computer Components:Top-Level View

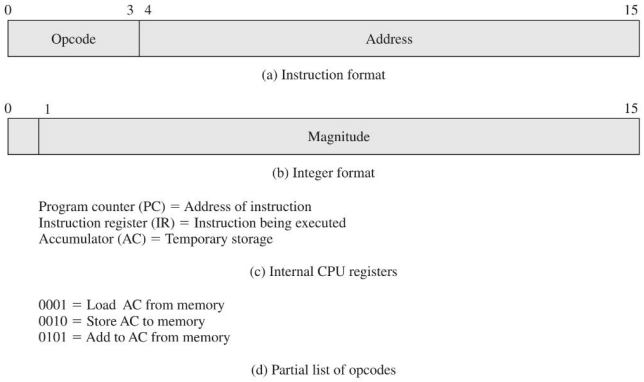


Figure 3.4 Characteristics of a Hypothetical Machine

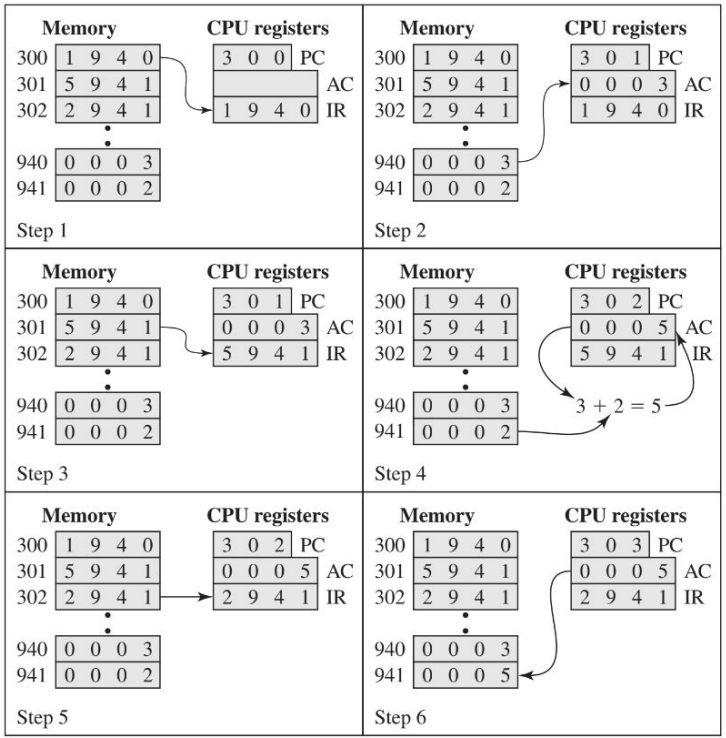
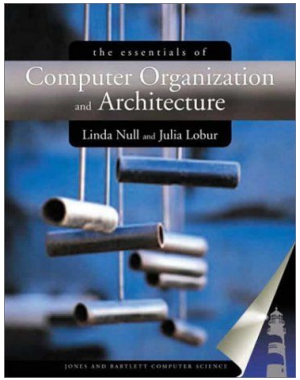


Figure 3.5 Example of Program Execution (contents of memory and registers in hexadecimal)

L. Null

MARIE: A Machine Architecture that is Really Intuitive and Easy

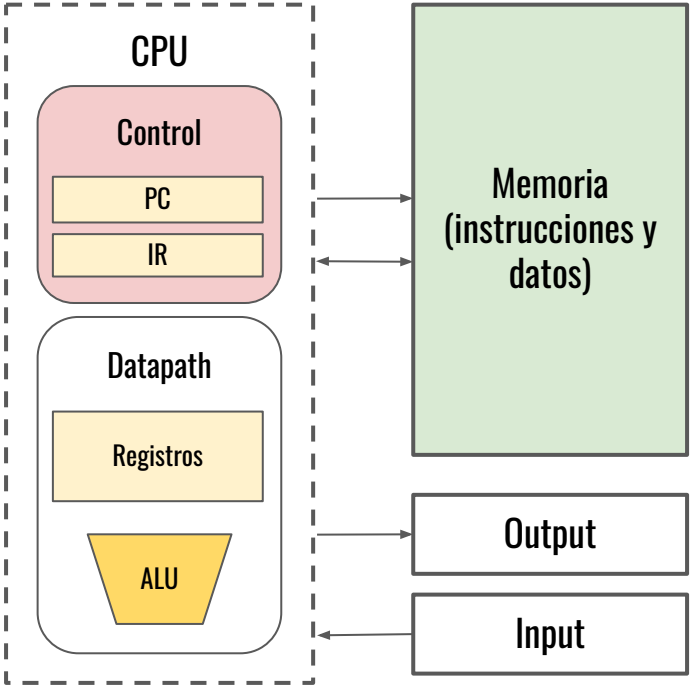


El simulador oficial
http://computerscience.jbpub.com/ecoa/2e/student_resources.cfm

Un simulador online mejorado
https://marie.js.org

Wiki
https://github.com/MARIE-js/MARIE.js/wiki

El modelo de von Neumann



FETCH - DECODE - EXECUTE

- Capacidad de ejecución secuencial de instrucciones.
- Unidad central de procesamiento (CPU, central processing unit).
  - Unidad de control (CU, control unit), lleva a cabo el ciclo de instrucción: captación-decodificación-ejecución (fetch-decode-execute). Incluye dos registros de uso específico, un registro de instrucción (IR, instruction register) y un contador de programa (PC, program counter).
  - Unidad de procesamiento (Datapath) que incluye una unidad aritmética lógica (ALU, arithmetic logic unit) y uno o varios registros de usuario.
- Unidad de memoria única (almacenamiento de instrucciones y de datos).
- Mecanismo de entrada-salida (I/O).

MARIE - CARACTERÍSTICAS PRINCIPALES

Representación binaria de datos codificados en complemento a dos
Datos de 16 bits (las palabras tienen 16 bits)
Programa almacenado, largo de palabra fijo (16 bits)
4K palabras de memoria principal (12 bits por dirección)
Instrucciones de 16 bits, 4 de opcode y 12 de argumento (dirección)
Un registro de usuario, acumulador, de 16 bits (AC)
Un registro de instrucción de 16 bits (IR)

Program counter de 12 bits (PC)
Memory address register de 12 bits (MAR)
Memory buffer register de 16 bits (MBR)

Input register de 8 bits
Output register de 8 bits

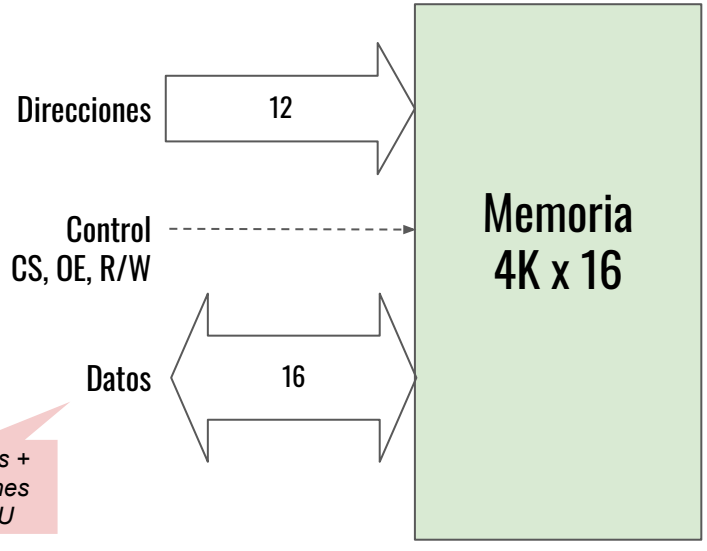
Procesador TIPO(1,1)

```
LOAD  X
ADD  Y
STORE  Z        / z=x+y
```

Clasificación TIPO(m,n)
m: número de operandos en memoria
n: número de operandos explícitos Intel es (1,2), RISC es (0,3)

INSTRUCCION:

OPCODE 15-12	ADDRESS 11-0
2^4 = 16 instr	2^{12} = 4K memoria



Modelo de **von Neumann**: en el mismo espacio lógico de memoria se almacenan **datos** e **instrucciones**.

La memoria es volátil (velocidad), lectura-escritura y de acceso aleatorio: **RAM**

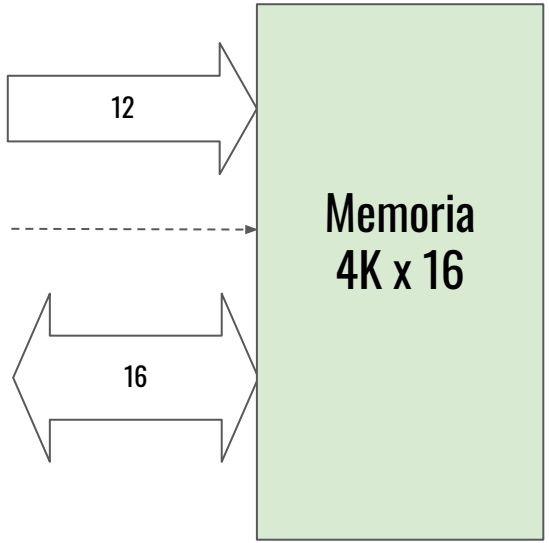
OJO: datos + instrucciones de la CPU

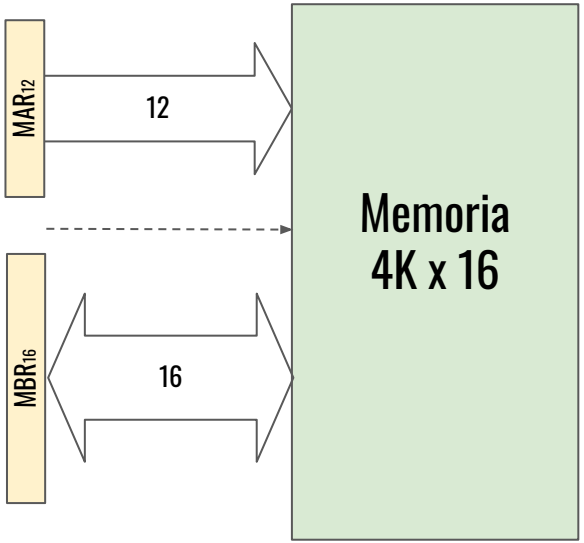
Todas las instrucciones son referidas al unico registro disponible (acumulador)

0001) LOAD X
Cargar el acumulador con el contenido de la posición de memoria X

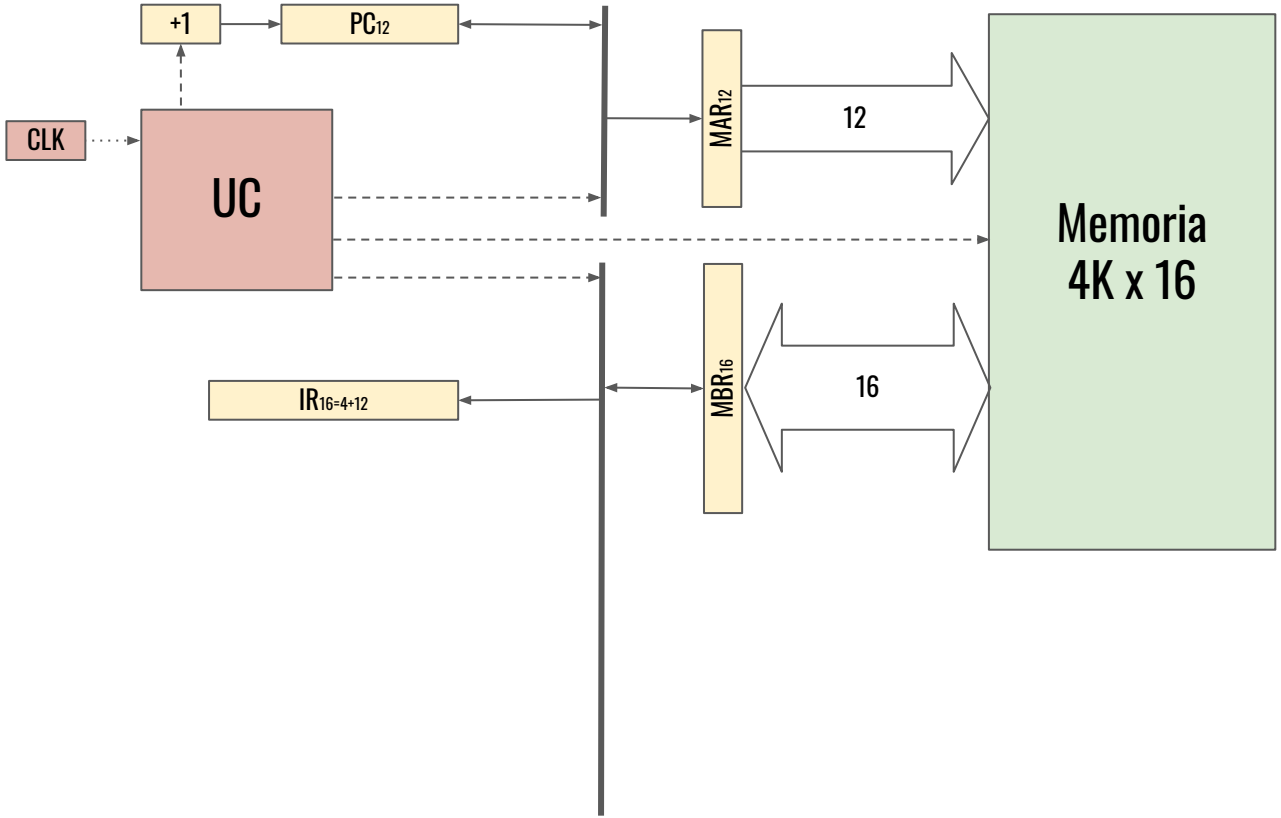
0010) STORE X
Guardar el contenido del acumulador en la posición de memoria X

0011) ADD X
Sumar el contenido del acumulador con el contenido de la posición de memoria X y guardarlo en el acumulador

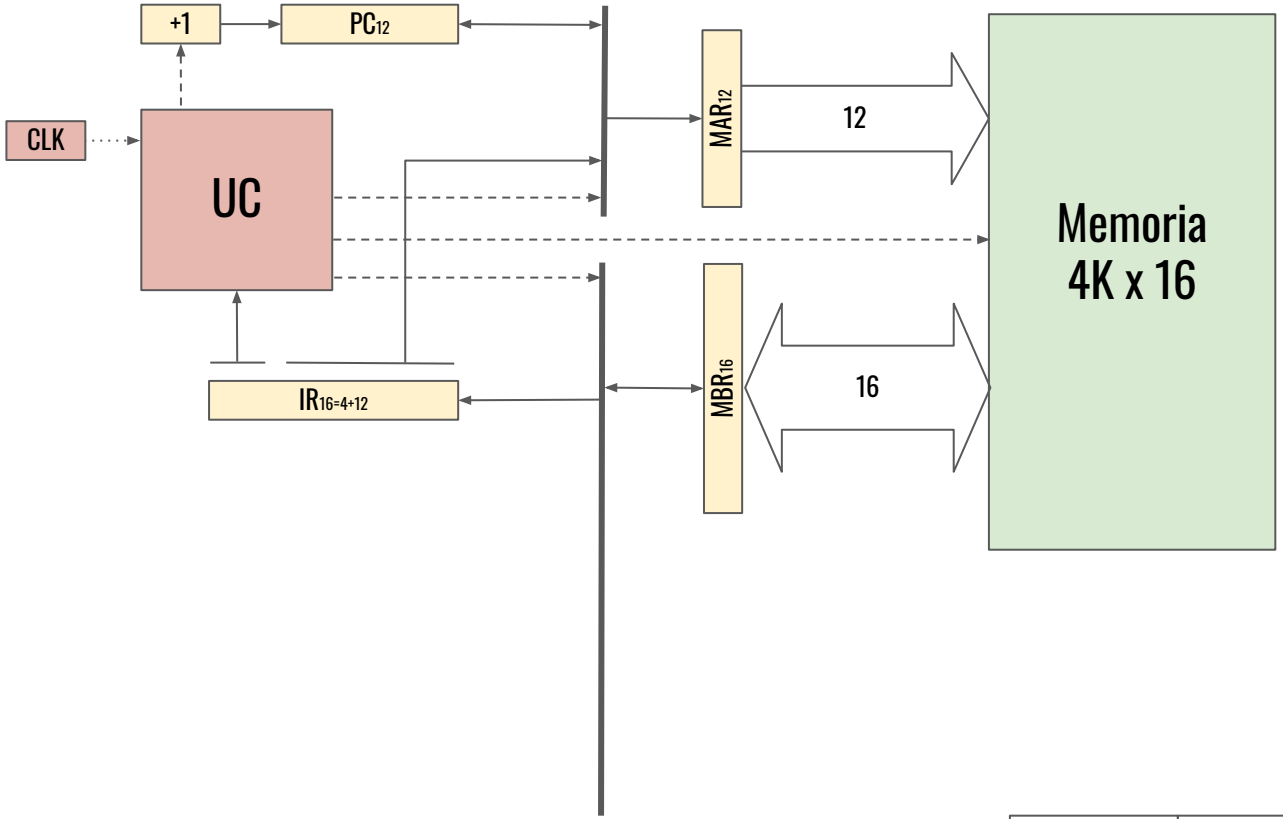

12
Memoria
4K x 16
16


MAR12
12
Memoria
4K x 16
MBR16
16

FETCH

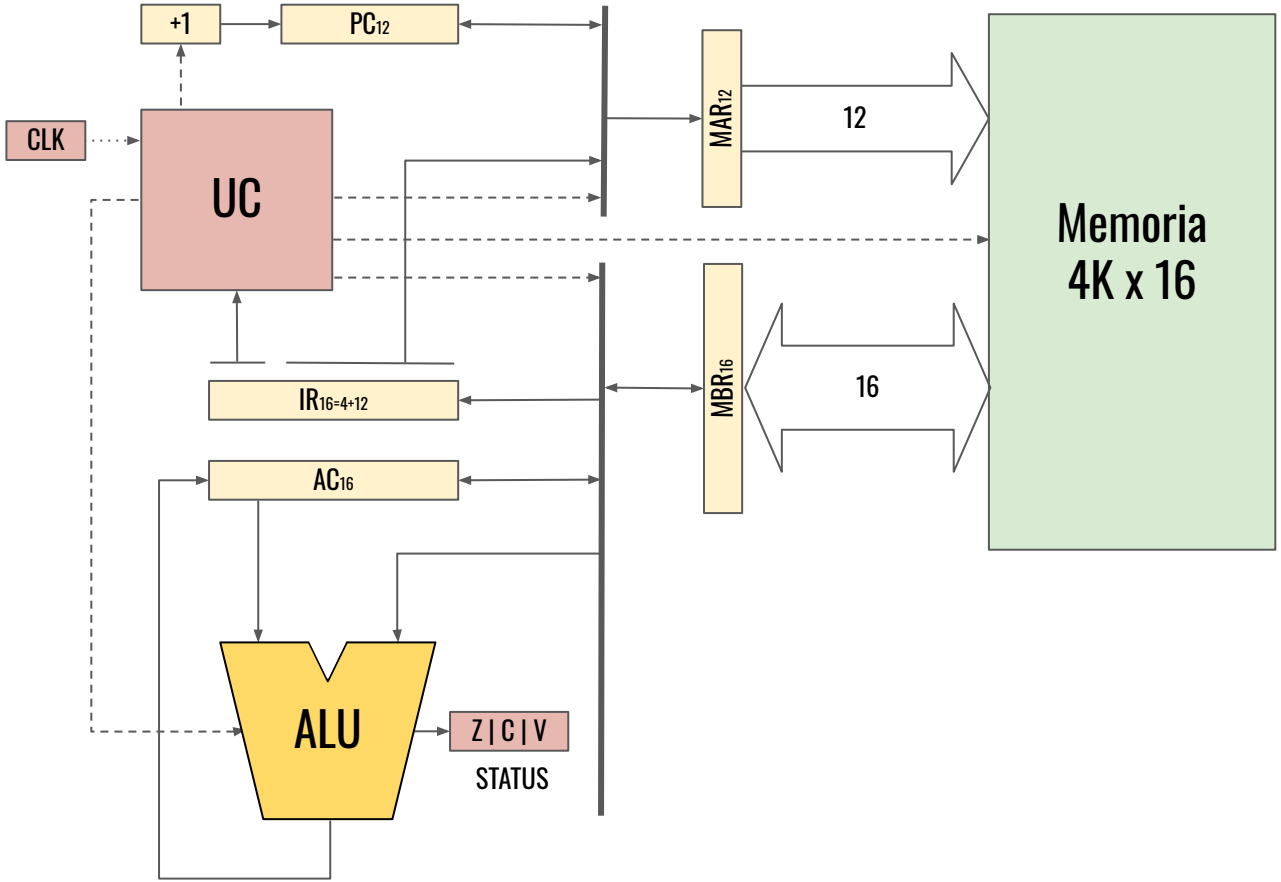


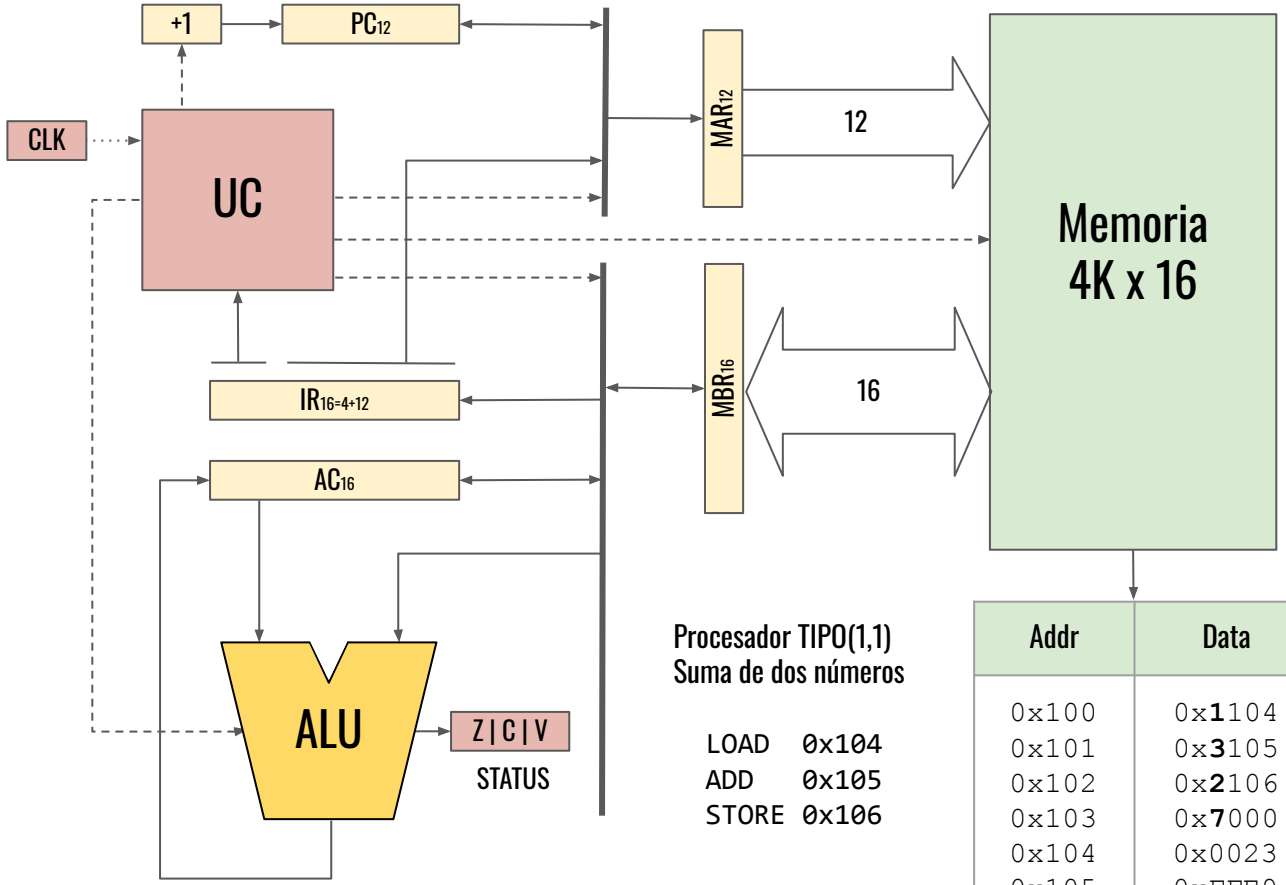
DECODE



OPCODE 15-12	ADDRESS 11-0
2^4 = 16 instr	2^{12} = 4K memoria

EXECUTE


+1
PC12
CLK
UC
MAR12
12
Memoria
4K x 16
MBR16
16
IR16=4+12
AC16
ALU
Z | C | V
STATUS



Procesador TIPO(1,1)
Suma de dos números

LOAD 0x104
ADD 0x105
STORE 0x106

Addr	Data
0x100	0x1104
0x101	0x3105
0x102	0x2106
0x103	0x7000
0x104	0x0023
0x105	0xFFE9
0x106	0x0000

OPCODE 15-12	ADDRESS 11-0
2^4 = 16 instr	2^{12} = 4K memoria

RTL

FETCH:
```
MAR <- PC
IR  <- M[MAR]
```

DECODE:
```
PC  <- PC + 1
Decode IR15-12
MAR <- IR11-0
```

EXECUTE:

0001) LOAD
```
MBR <- M[MAR]
AC  <- MBR
```

0010) STORE
```
MBR <- AC
M[MAR] <- MBR
```

0011) ADD
```
MBR <- M[MAR]
AC  <- AC + MBR
```

0111) HALT

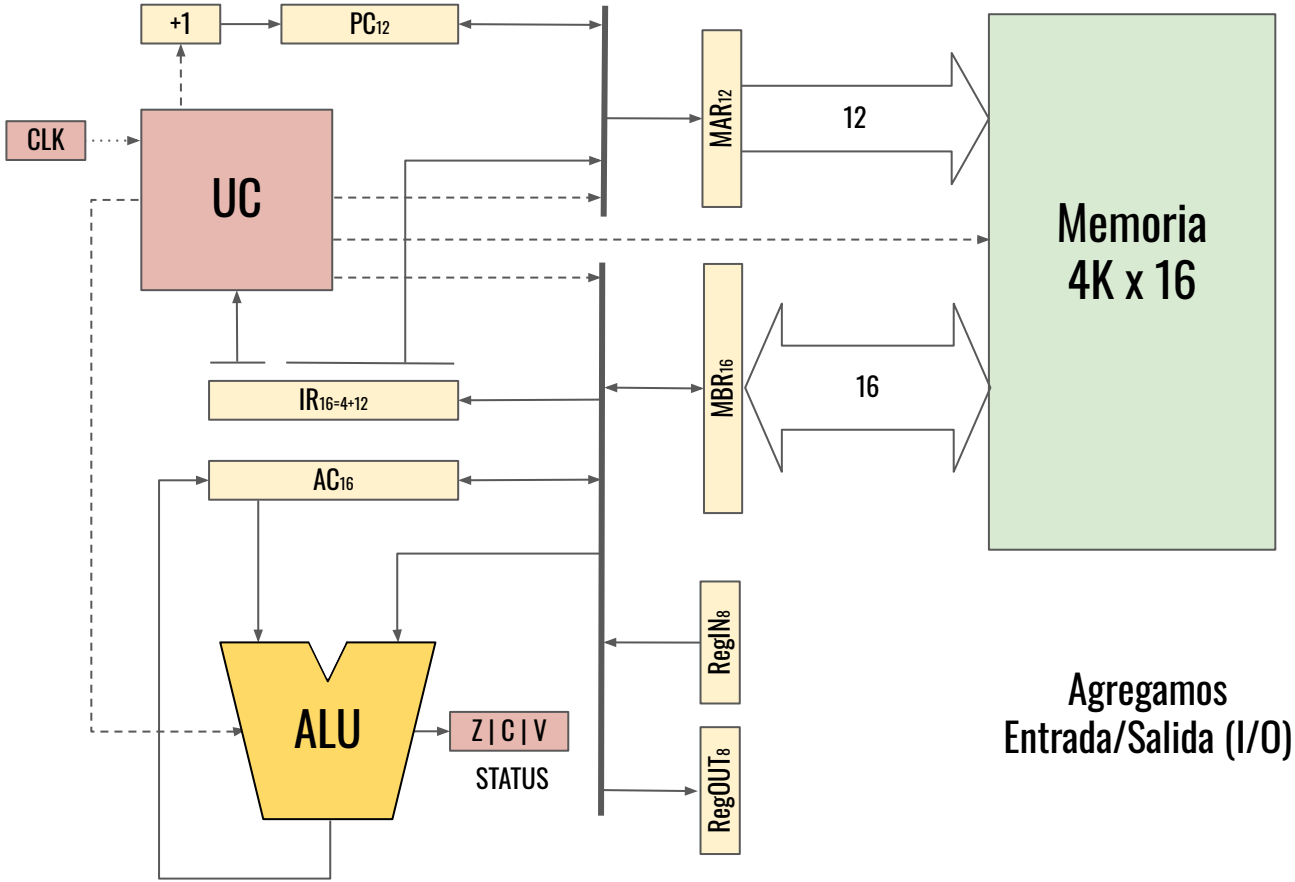
35 + (-23) = 12 = 0x000C

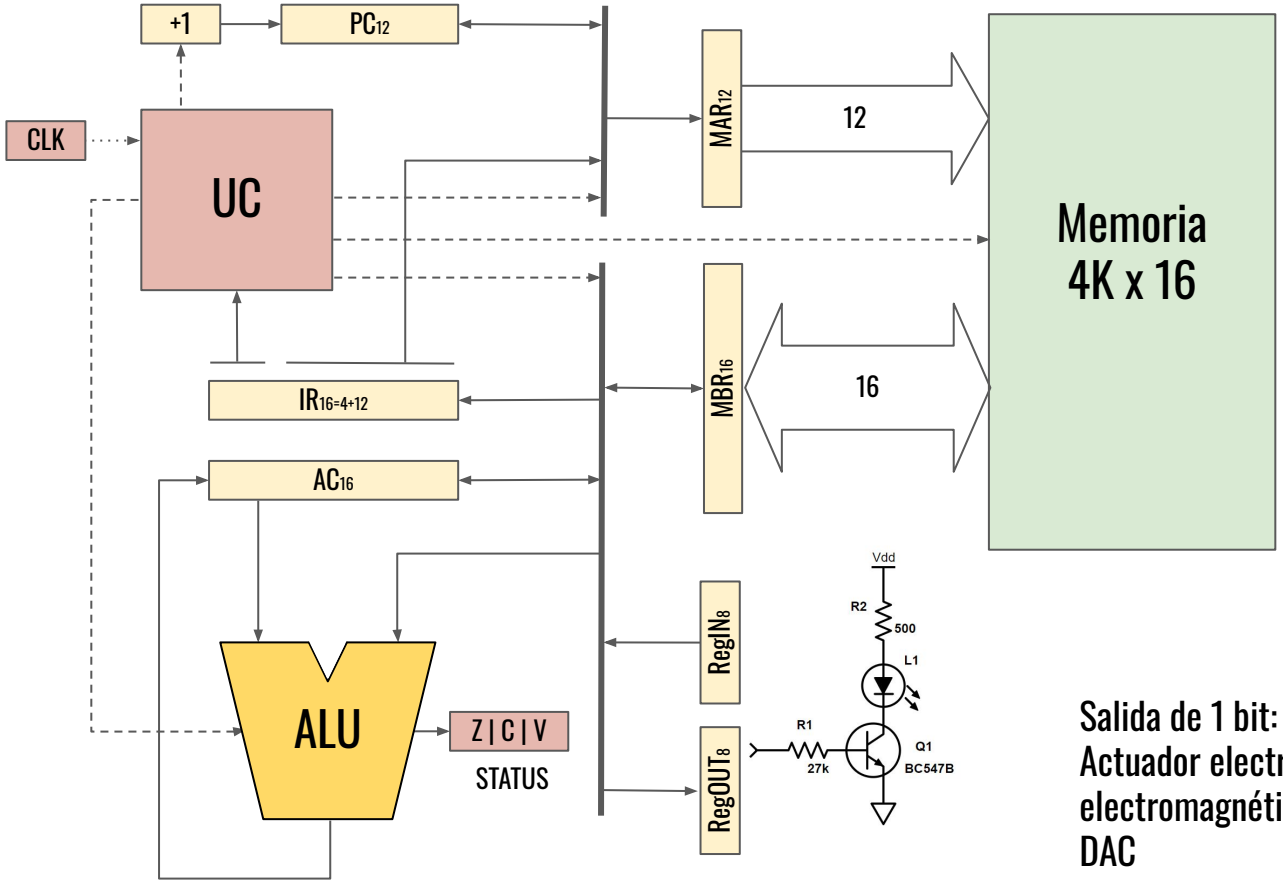
Primer programa RTL

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		100	------	------	------	------
Fetch	MAR ⟵ PC	100	------	100	------	------
	IR ⟵ M[MAR]	100	1104	100	------	------
	PC ⟵ PC + 1	101	1104	100	------	------
Decode	MAR ⟵ IR[11–0]	101	1104	104	------	------
	(Decode IR[15–12])	101	1104	104	------	------
Get operand	MBR ⟵ M[MAR]	101	1104	104	0023	------
Execute	AC ⟵ MBR	101	1104	104	0023	0023

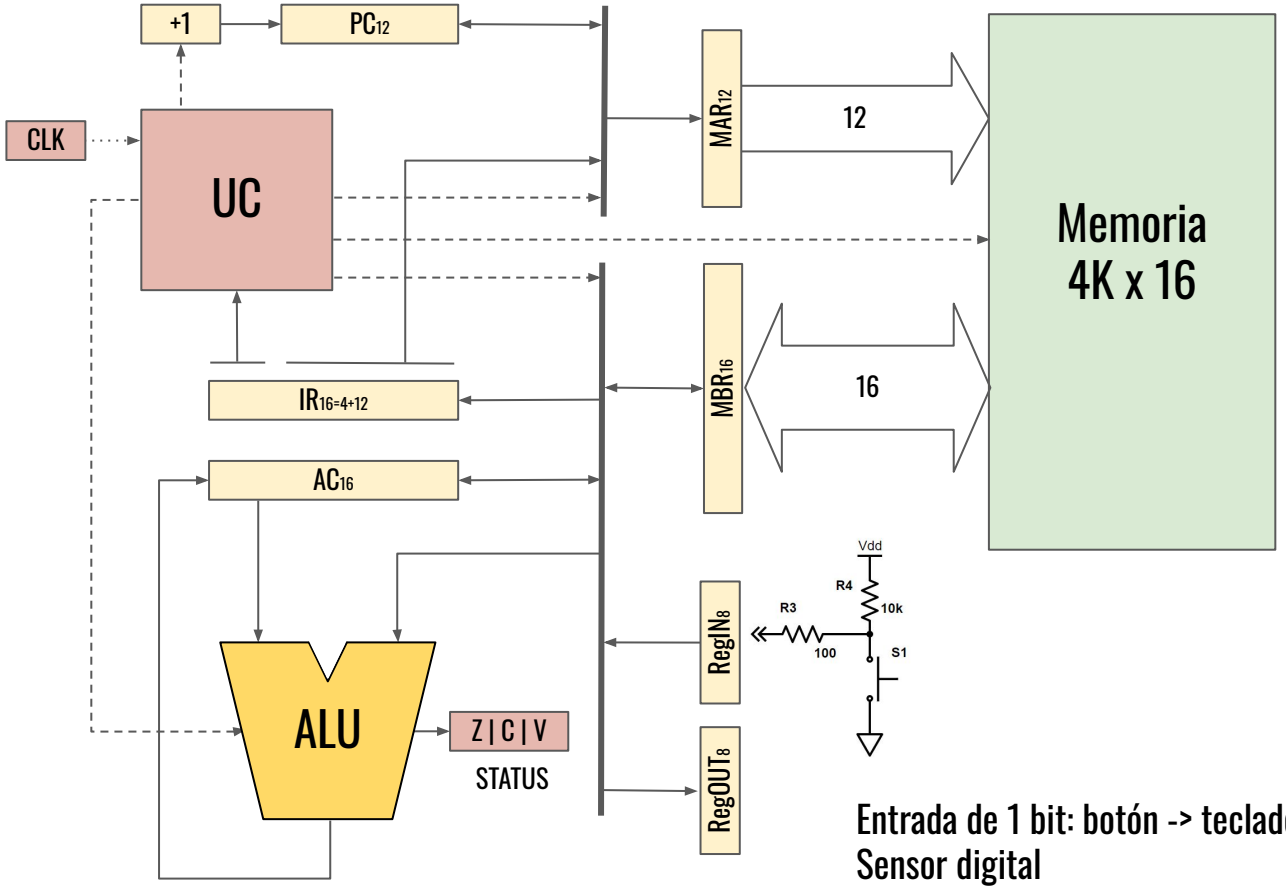
Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		101	1104	104	0023	0023
Fetch	MAR ⟵ PC	101	1104	101	0023	0023
	IR ⟵ M[MAR]	101	3105	101	0023	0023
	PC ⟵ PC + 1	102	3105	101	0023	0023
Decode	MAR ⟵ IR[11–0]	102	3105	105	0023	0023
	(Decode IR[15–12])	102	3105	105	0023	0023
Get operand	MBR ⟵ M[MAR]	102	3105	105	FFE9	0023
Execute	AC ⟵ AC + MBR	102	3105	105	FFE9	000C

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		102	3105	105	FFE9	000C
Fetch	MAR ⟵ PC	102	3105	102	FFE9	000C
	IR ⟵ M[MAR]	102	2106	102	FFE9	000C
	PC ⟵ PC + 1	103	2106	102	FFE9	000C
Decode	MAR ⟵ IR[11–0]	103	2106	106	FFE9	000C
	(Decode IR[15–12])	103	2106	106	FFE9	000C
Get operand	(not necessary)	103	2106	106	FFE9	000C
Execute	MBR ⟵ AC	103	2106	106	000C	000C
	M[MAR] ⟵ MBR	103	2106	106	000C	000C

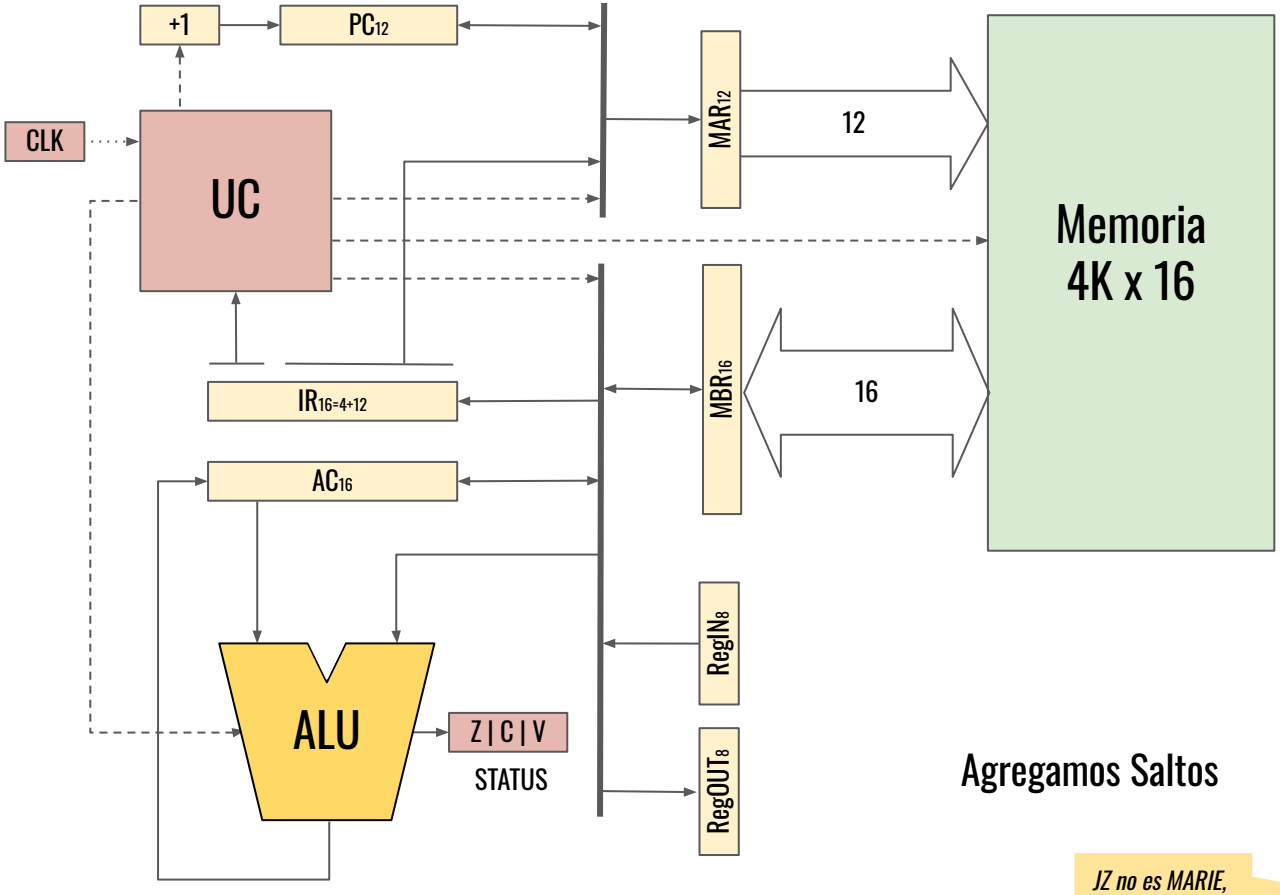

+1
PC12
CLK
UC
MAR12
12
Memoria
4K x 16
MBR16
16
IR16=4+12
AC16
RegIN8
ALU
Z | C | V
STATUS
RegOUT8
Agregamos
Entrada/Salida (I/O)



Salida de 1 bit: LED -> display, monitor
Actuador electromecánico, electromagnético, relé, etc.
DAC



Entrada de 1 bit: botón -> teclado, mouse, touch
Sensor digital
ADC y sensor analógico



Agregamos Saltos

OPCODE 15-12	ADDRESS 11-0
2^4 = 16 instr	2^{12} = 4K memoria

JZ no es MARIE, ver cómo implementa los saltos

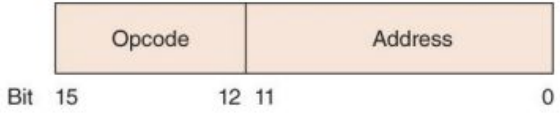
```
FETCH:
        MAR <- PC
        IR  <- M[MAR]
DECODE:
        PC  <- PC + 1

EXECUTE:
0001) LOAD X
        MAR <- X
        MBR <- M[MAR]
        AC  <- MBR
0010) STORE X
        MAR <- X
        MBR <- AC
        M[MAR] <- MBR
0011) ADD X
        MAR <- X
        MBR <- M[MAR]
        AC  <- AC + MBR
0100) SUBT X
        MAR <- X
        MBR <- M[MAR]
        AC  <- AC - MBR
0101) INPUT
        AC <- RegIN
0110) OUTPUT
        RegOUT <- AC
0111) HALT
1000) JZ X
        If AC=0, PC <- X
1001) JUMP X
        PC <- X
```

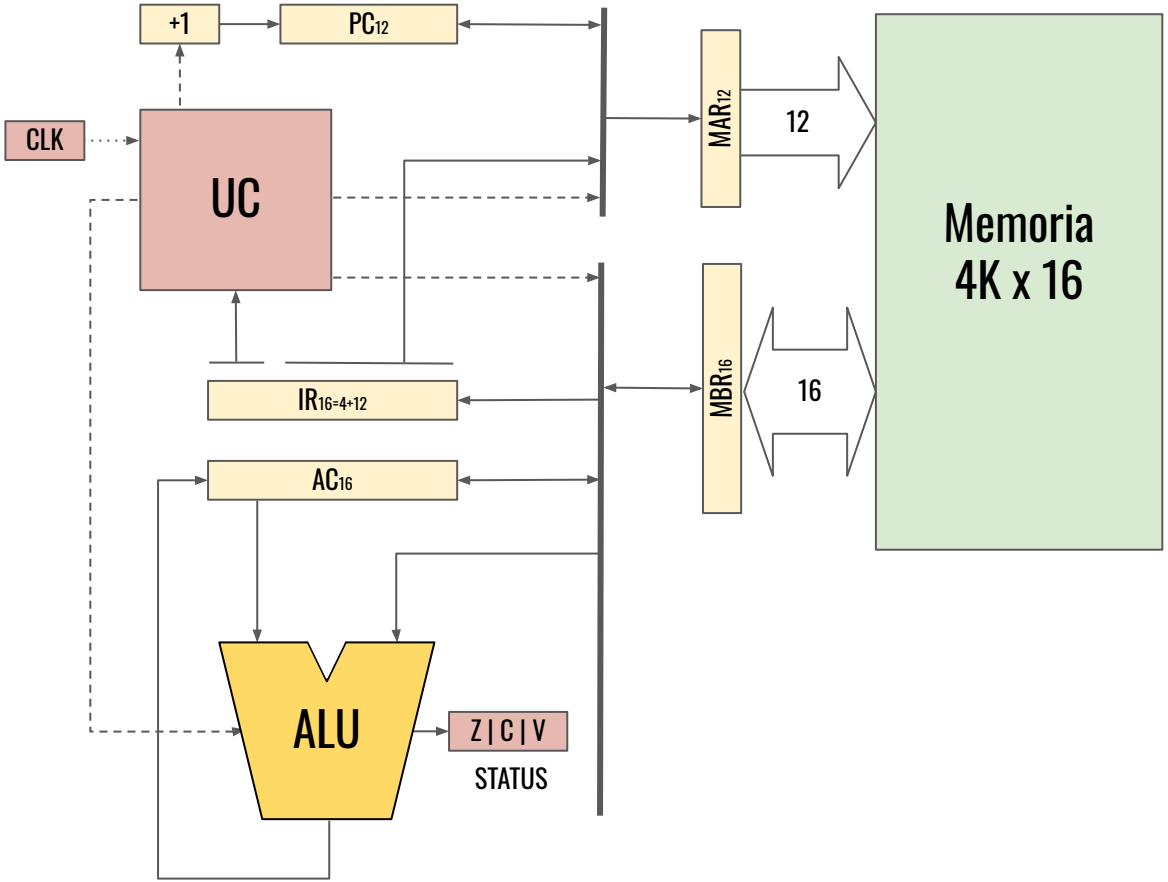
Type	Instruction	Hex Opcode	Summary
Arithmetic	**Add X**	3	Adds value in AC at address X into AC, AC ← AC + X
	Subt X	4	Subtracts value in AC at address X into AC, AC ← AC - X
	Addl X	B	Add Indirect: Use the value at X as the actual address of the data operand to add to AC
	Clear	A	AC ← 0
Data Transfer	**Load X**	1	Loads Contents of Address X into AC
	Store X	2	Stores Contents of AC into Address X
I/O	**Input**	5	Request user to input a value
	Output	6	Prints value from AC
Branch	**Jump X**	9	Jumps to Address X
	Skipcond (C)	8	Skips the next instruction based on C: if (C) = - 000: Skips if AC < 0 - 400: Skips if AC = 0 - 800: Skips if AC > 0
Subroutine	**JnS X**	0	Jumps and Store: Stores value of PC at address X then increments PC to X+1
	Jumpl X	C	Uses the value at X as the address to jump to
Indirect Addressing	**Storel**	D	Stores value in AC at the indirect address. e.g. Storel addresspointer Gets value from addresspointer, stores the AC value into the address
	Loadl	E	Loads value from indirect address into AC e.g. Loadl addresspointer Gets address value from addresspointer, loads value at the address into AC
	Halt	7	End the program

MARIE: Repertorio de instrucciones completo

MARIE ISA

Opcode	Instruction	Description
0	JnS X	Store the PC at address X and jump to X+1
1	Load X	Load contents of address X into AC
2	Store X	Store the contents of AC at address X
3	Add X	Add the contents of address X to AC
4	Subt X	Subtract the contents of address X from AC
5	Input	Input a value from the keyboard into AC
6	Output	Output the value in AC to the display
7	Halt	Terminate program
8	Skipcond X	Skip next instruction on condition (See note below.) The two address bits closest to the opcode field, bits 10 and 11 specify the condition to be tested. If the two address bits are 00, this translates to "skip if the AC is negative". If the two address bits are 01, this translates to "skip if the AC is equal to 0". Finally, if the two address bits are 10 (or 2), this translates to "skip if the AC is greater than 0". Example: the instruction Skipcond 800 will skip the instruction that follows if the AC is greater than 0.
9	Jump X	Load the value of X into PC
A	Clear	Put all zeros in AC
B	AddI X	Add indirect: Use the value at X as the actual address of the data operand to add to AC
C	JumpI X	Use the value at X as the address to jump to
D	LoadI X	Load indirect: Use the value at X as the address of the value to load.
E	StoreI X	Store indirect: Use X the value at X as the address of where to store the value.

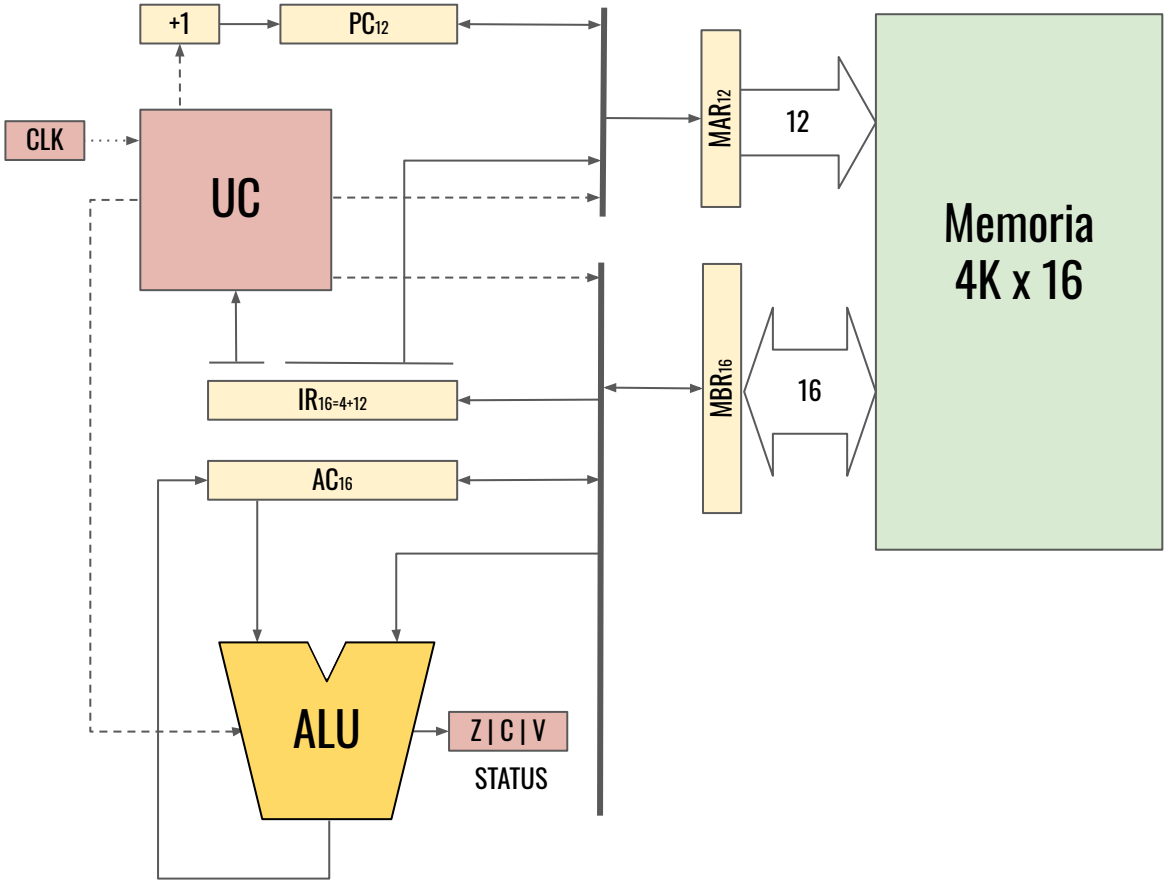




MARIE
REPASO DE CARACTERÍSTICAS

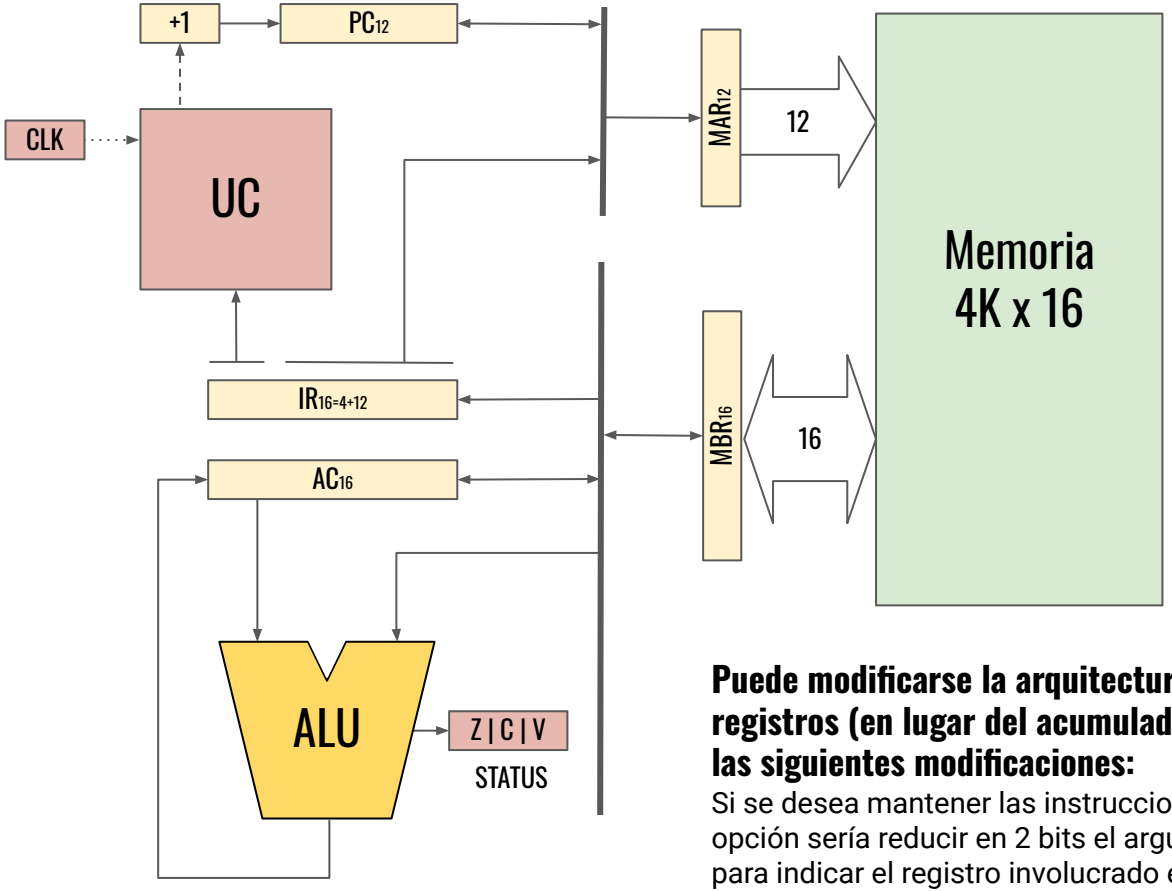
Representación binaria de datos en complemento a dos
Datos de 16 bits (las palabras tienen 16 bits)
Programa almacenado, largo de palabra fijo (16 bits)
4K palabras de memoria principal (12 bits por dirección)
Instrucciones de 16 bits, 4 de opcode y 12 de argumento (dirección)
Un registro de usuario, acumulador, de 16 bits (AC)
Un registro de instrucción de 16 bits (IR)
Program counter de 12 bits (PC)
Memory address register de 12 bits (MAR)
Memory buffer register de 16 bits (MBR)
Input register de 8 bits
Output register de 8 bits
Procesador de 16 bits TIPO(1,1)

https://github.com/MARIE-js/MARIE.js/wiki/



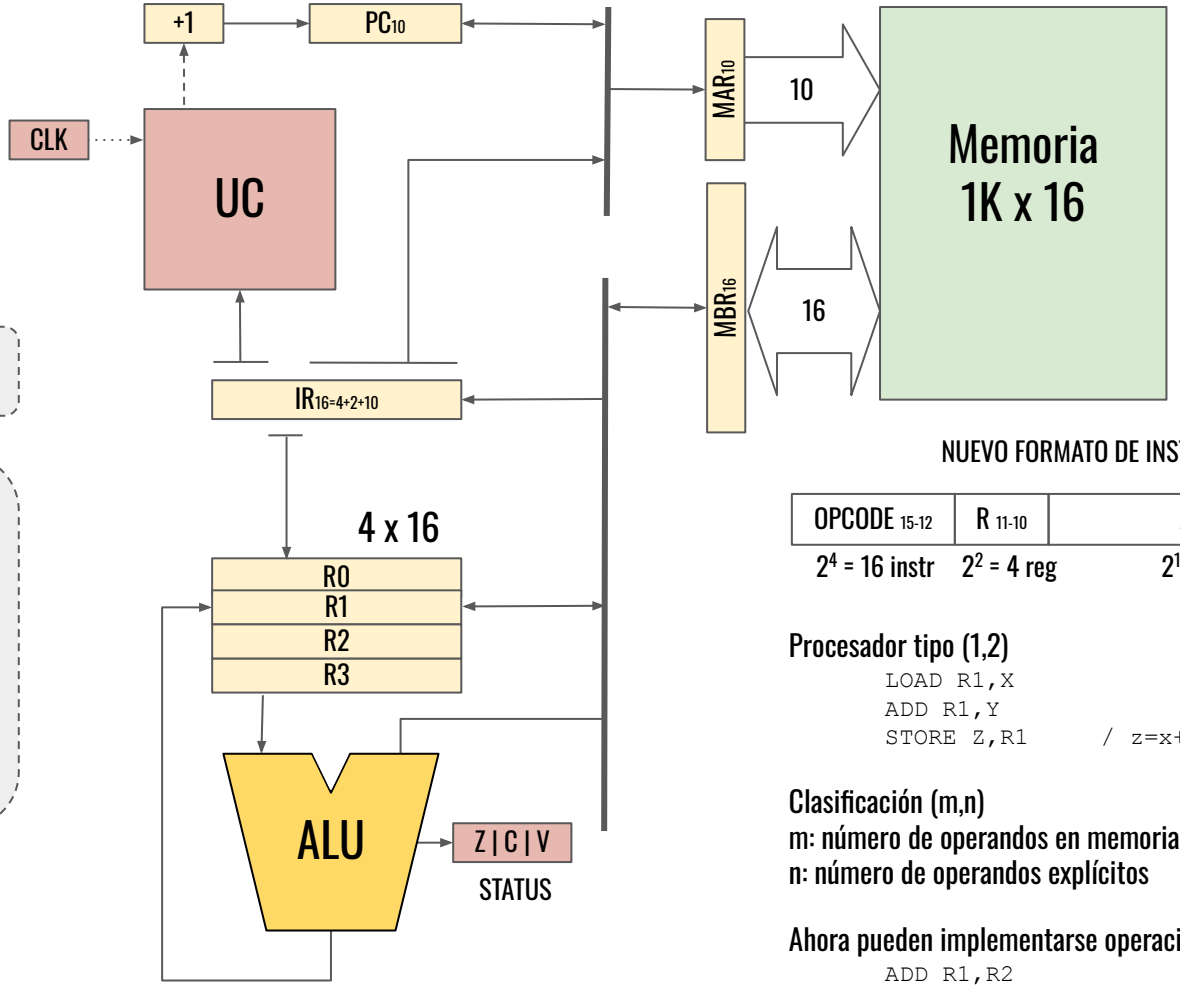
PC	Contador sincrónico de 16 bits con pre establecimiento	Técnicas Digitales P5
CLK	Reloj	Técnicas Digitales P3
Regis-t ros	16 latches D con salida 3-state	Técnicas Digitales P3
ALU	Suma Ca2, full adder 16 bits, multiplicación, etc.	Técnicas Digitales P7
UC	Máquina de estados sincrónica	Técnicas Digitales P8
Bus	Líneas comunes, 3-state	Técnicas Digitales P9

NUESTRA PRACTICA 0



Puede modificarse la arquitectura para incluir 4 registros (en lugar del acumulador único) haciendo las siguientes modificaciones:

Si se desea mantener las instrucciones de 16 bits, una opción sería reducir en 2 bits el argumento y utilizarlos para indicar el registro involucrado en la instrucción. Se reduce la cantidad de memoria accesible desde el argumento (10 bits = 1KB) y pueden direccionarse 4 registros.



OJO! Esto no es MARIE

Se podría mantener el PC de 12 bits para manejar 4K instrucciones, pero sólo el primer 1K podría ser utilizado como memoria de datos.
Eso ya no sería von Neumann.
¿Cómo sería la instrucción de salto?

NUEVO FORMATO DE INSTRUCCIÓN

OPCODE 15-12	R 11-10	ADDRESS 9-0
2^4 = 16 instr	2^2 = 4 reg	2^{10} = 1K memoria

Procesador tipo (1,2)

```
LOAD R1,X
ADD R1,Y
STORE Z,R1      / z=x+y
```

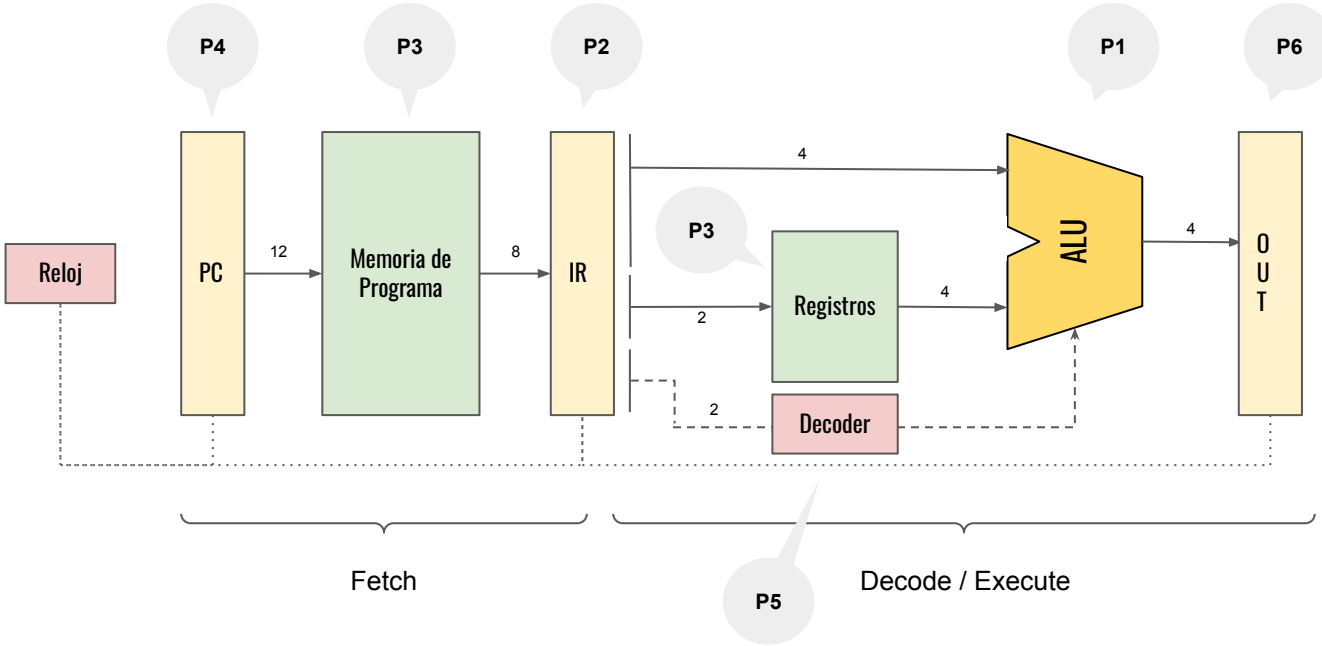
Clasificación (m,n)
m: número de operandos en memoria
n: número de operandos explícitos

Ahora pueden implementarse operaciones entre registros

```
ADD R1,R2
```

Práctica 0: Diseño parcial, sin UC

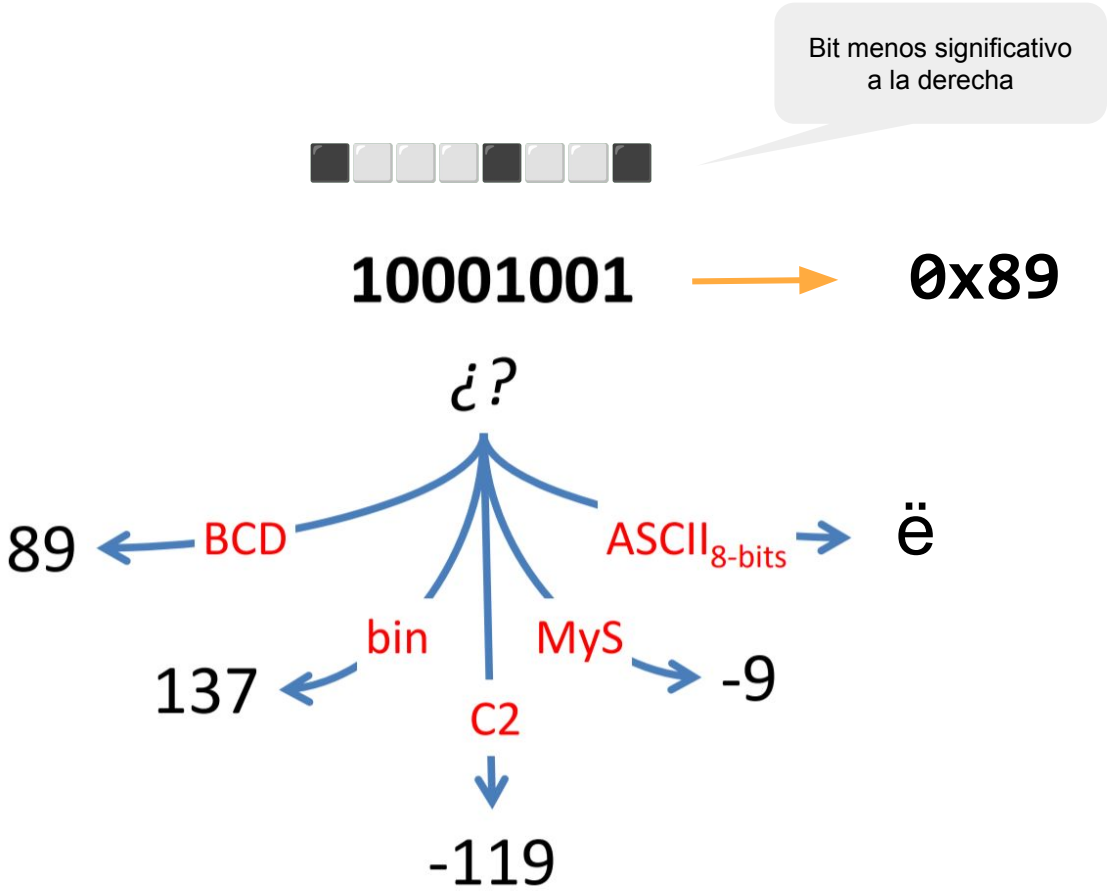
Como no puede accederse dos veces a la memoria, no puede implementarse el modelo de von Neumann.
El diseño con UC como máquina de estados se muestra en la última clase de práctica del módulo (MARIE).



Anexos

Representación de la información en sistemas digitales

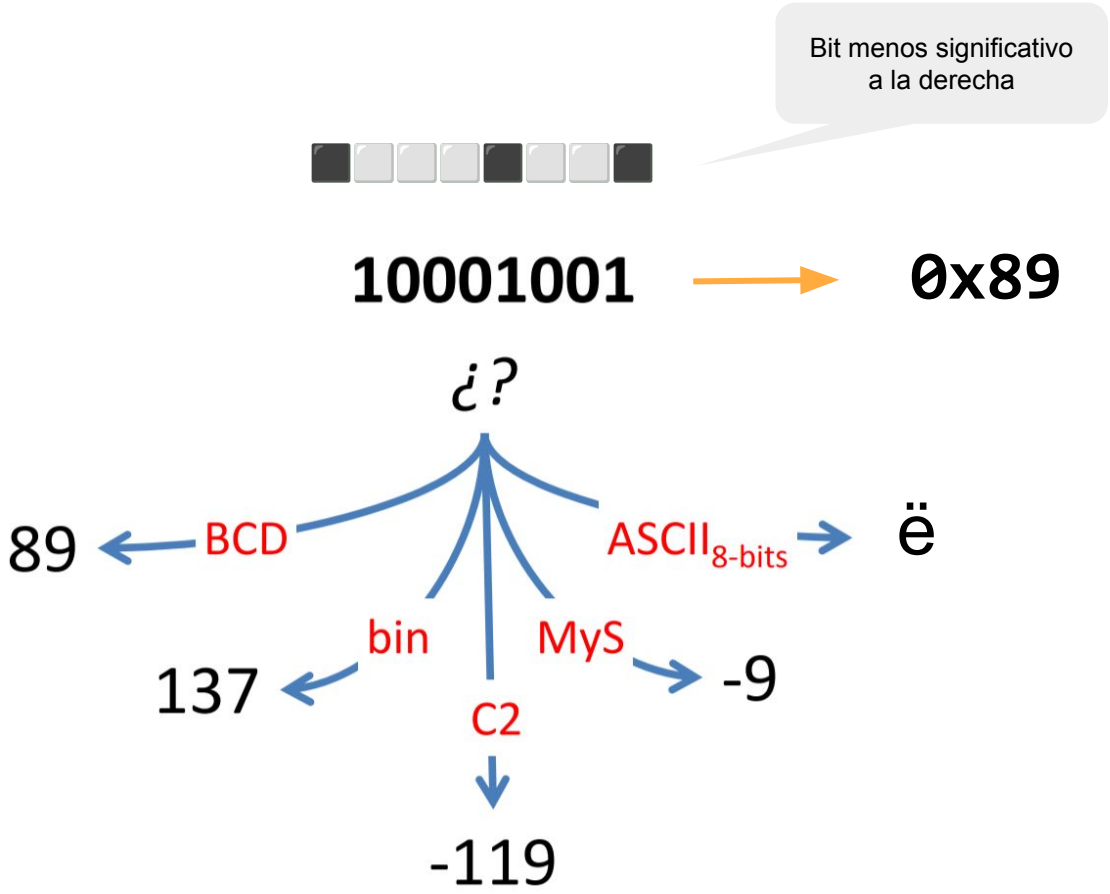
Una cadena de bits por sí misma no significa nada. Es la codificación utilizada lo que le da sentido (interpretación).



Probarlo en
https://hexed.it/

Representación de la información en sistemas digitales

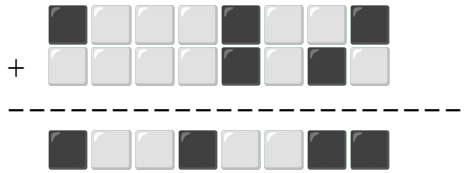
Una cadena de bits por sí misma no significa nada. Es la codificación utilizada lo que le da sentido (interpretación).



Si busco 10001001 en Google...

Bit menos significativo a la derecha → carry a la izquierda

Sumador de 8 bits

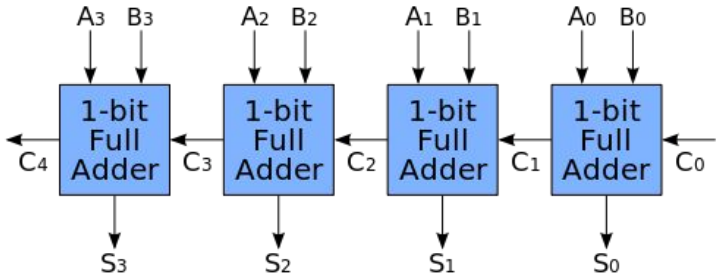


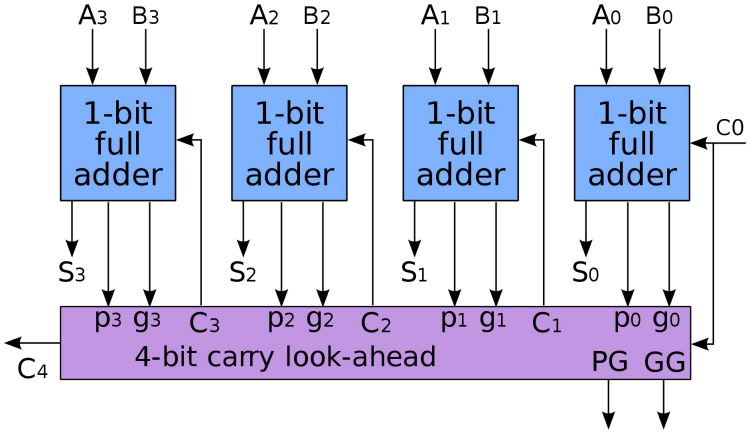
```
BIN: 10001001 + 00001010 = 10010011
HEX: 0x89 + 0x0A = 0x93
```

Interpretación del resultado

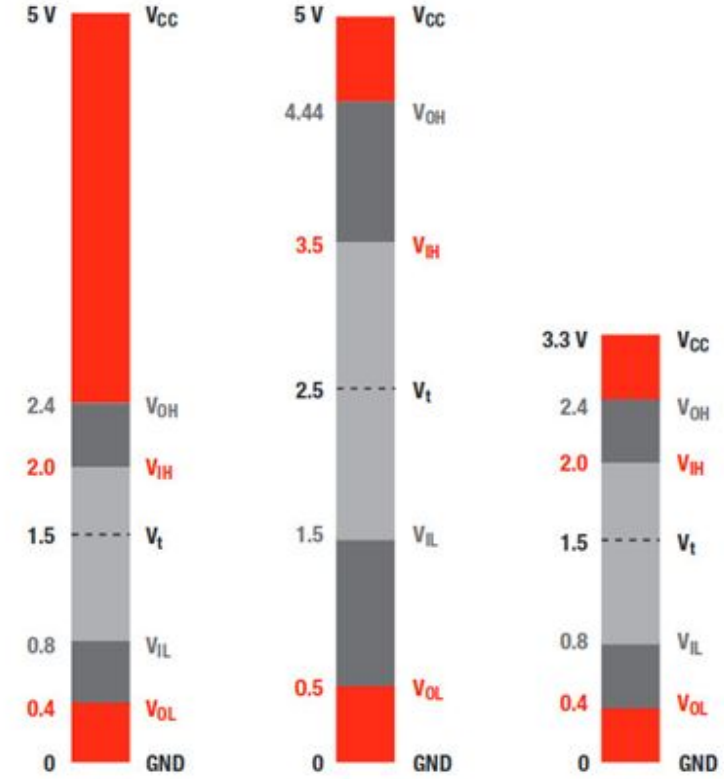
```
BIN:   137 + 10 = 147      ✅
Ca2:   -119 + 10 = -109    ✅
MyS:   -9 + 10 = -19       ✖
BCD:   89 + 0? = 92        ✖
ASCII: ë + LF = ô          ✖
```

Si se representan los enteros en Ca2, en la ALU no hace falta implementar la resta.





https://en.wikipedia.org/wiki/Adder_(electronics)

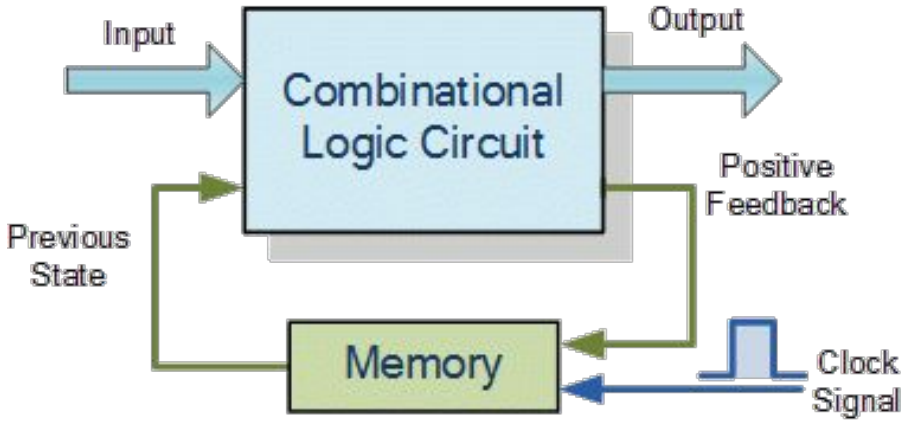


5-V TTL
Standard TTL: ABT, AHCT, HCT, ACT, bipolar, LV1T, LV4T

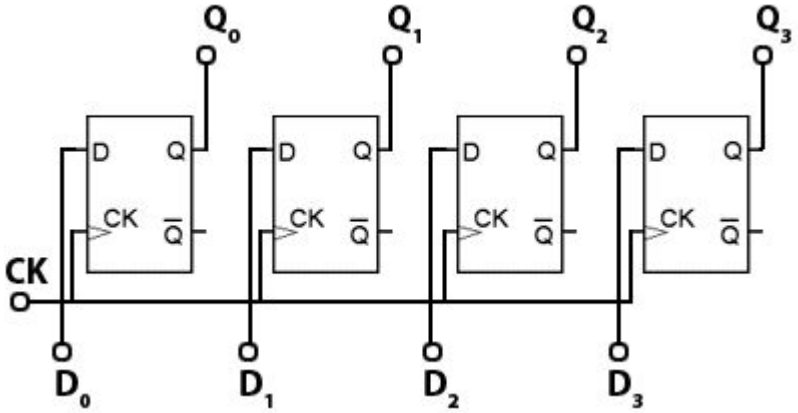
5-V CMOS
Rail-to-Rail 5 V HC, AHC, AC, LV-A, LV1T, LV4T

3.3-V LVTTL
LVT, LV1T, LV4T, LVC, ALVC, AUP, LV-A, ALVT

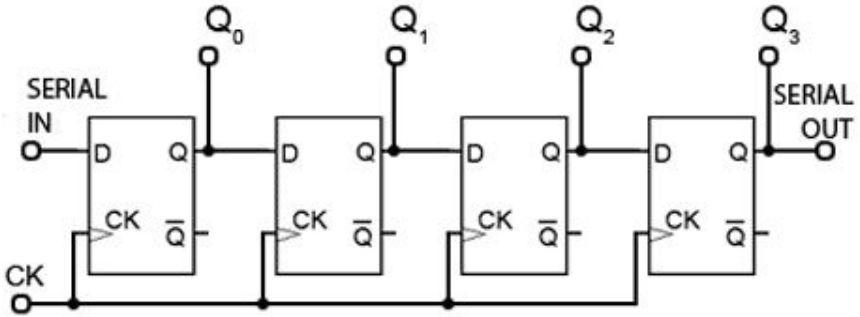
https://www.allaboutcircuits.com/textbook/digital/chpt-3/logic-signal-voltage-levels/


Input
Combinational
Logic Circuit
Output
Positive
Feedback
Previous
State
Memory
Clock
Signal

Notas sobre Registos



4-bit (Data) Register: 4 FF tipo D **en paralelo**

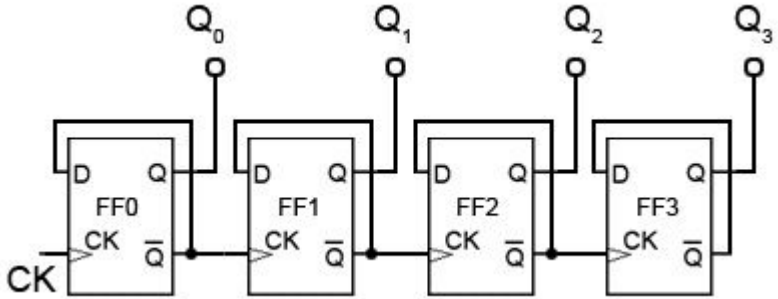


4-bit Shift Register: 4 FF tipo D **en cascada**

Si el registro va a ser conectado a un bus, debe ser diseñado como hicimos con una memoria, en este caso sería una

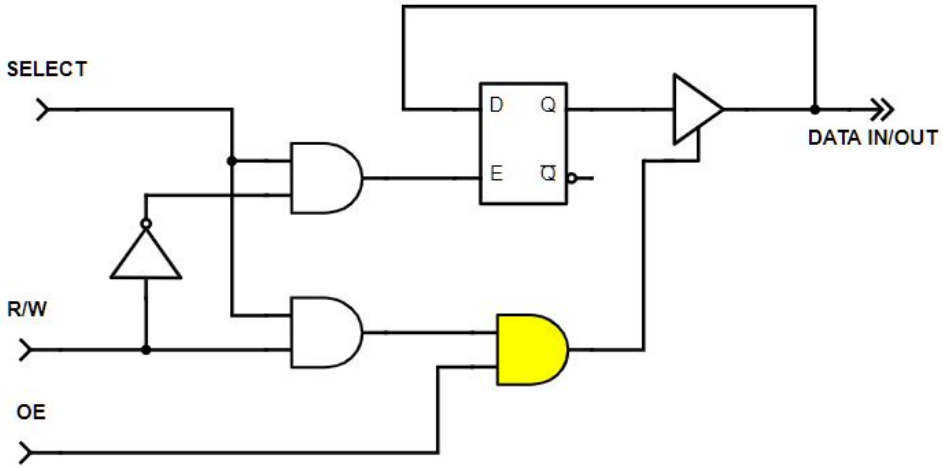
1x4 SRAM

con tristate output y chip select, recordar la clase anterior.



4-bit Ripple Counter: 4 FF tipo T **en cascada**

Elemento de almacenamiento estático

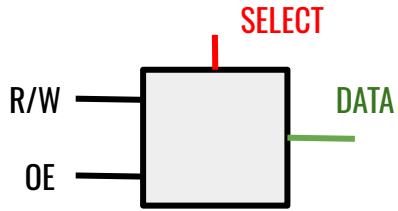


Flip-flop tipo D

Unificación de las líneas de entrada y de salida de datos (o leo o escribo, selección con una línea única que puede ser común para todos los latches).
Una entrada R/~W
Una entrada/salida DATA

Se agrega lógica que permita seleccionar el elemento: SELECT o CS.

Se agrega una línea adicional que habilita la salida OE (Output Enable). Útil para la interconexión con buses.



5. Functional diagram

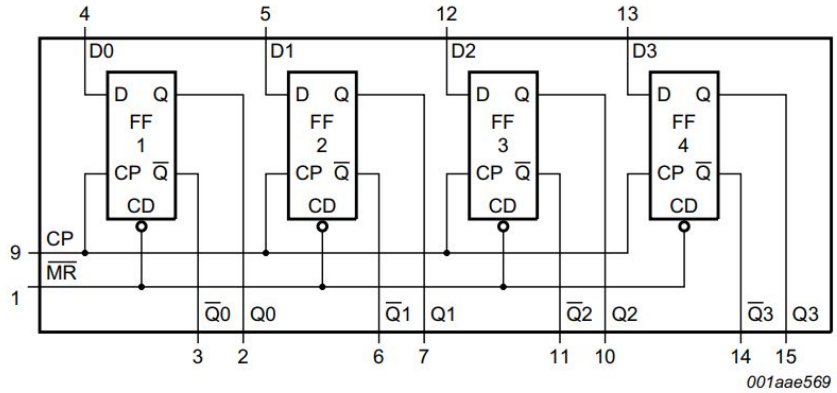
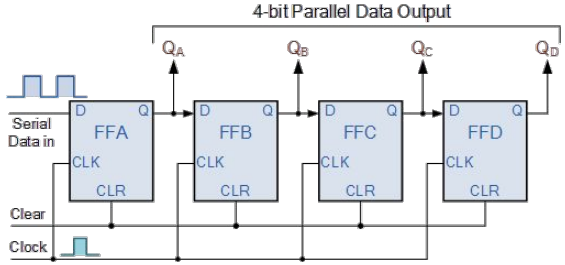


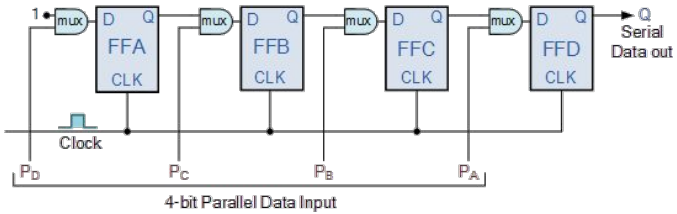
Fig 1. Functional diagram

VHDL

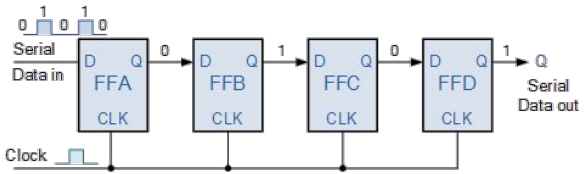
```
ENTITY register32 IS PORT(
    d   : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    ld  : IN STD_LOGIC; -- load/enable.
    clr : IN STD_LOGIC; -- async. clear.
    clk : IN STD_LOGIC; -- clock.
    q   : OUT STD_LOGIC_VECTOR(31 DOWNTO 0) -- output
);
END register32;

ARCHITECTURE description OF register32 IS

BEGIN
    process(clk, clr)
    begin
        if clr = '1' then
            q <= x"00000000";
        elsif rising_edge(clk) then
            if ld = '1' then
                q <= d;
            end if;
        end if;
    end process;
END description;
```



Serial-in to Parallel-out (SIPO) Shift Register

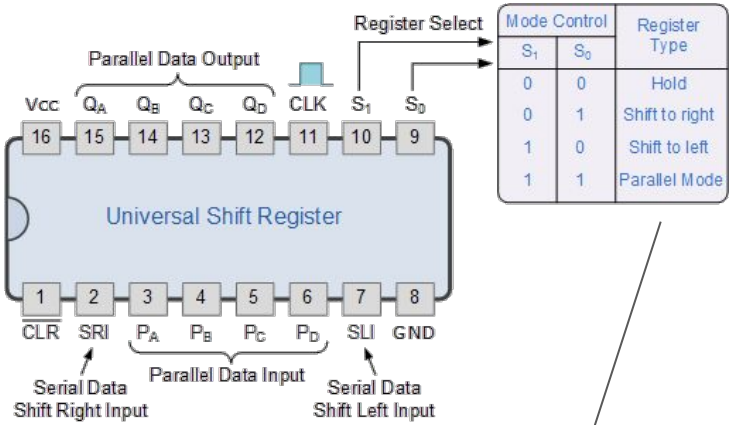


Parallel-in to Serial-out (PISO) Shift Register

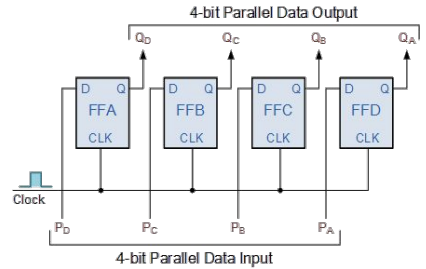


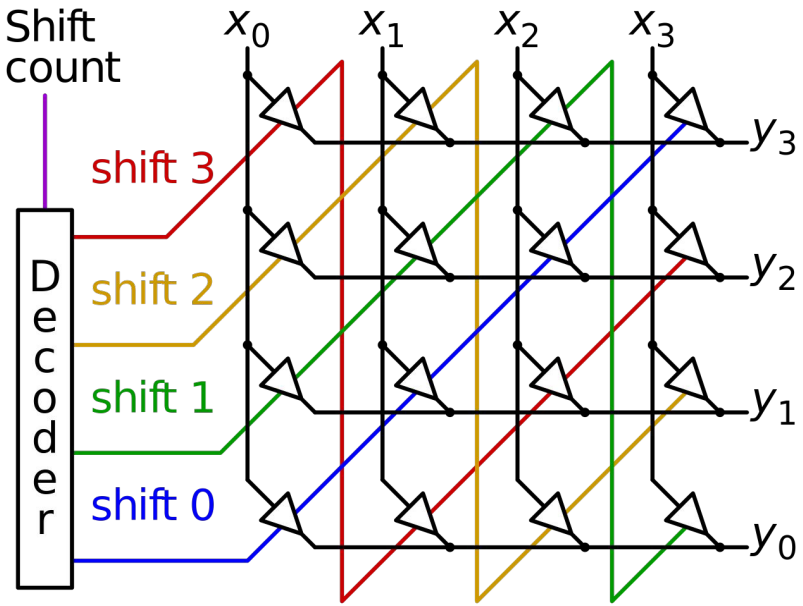
Serial-in to Serial-out (SISO) Shift Register

4-bit Universal Shift Register (74LS194)



Mode Control		Register Type
S_1	S_0	
0	0	Hold
0	1	Shift to right
1	0	Shift to left
1	1	Parallel Mode





NOTA: Barrel Shifter

Circuito puramente combinacional, usualmente construido con multiplexores y decodificadores, que puede rotar una palabra un determinado número de bits