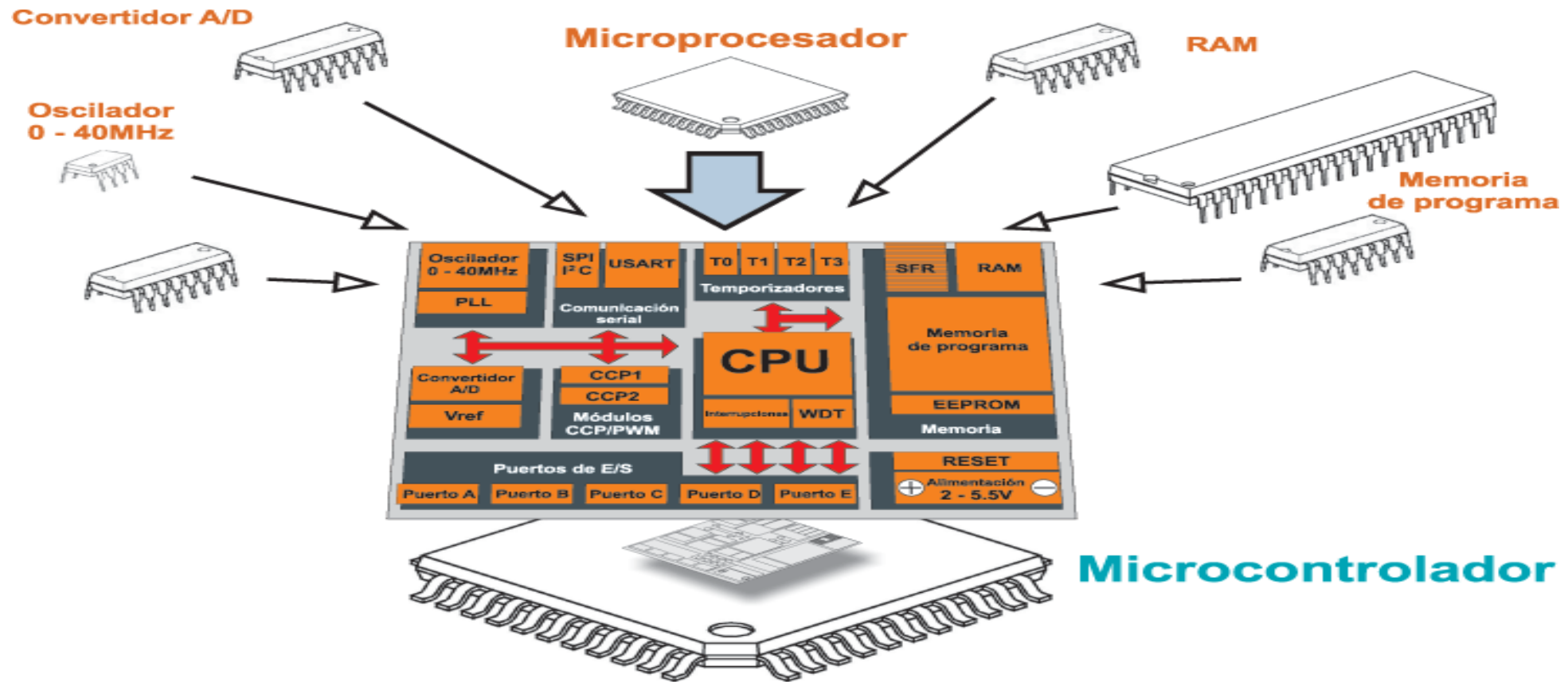


ARQUITECTURA CPU E INTRODUCCION ANSI C



Se basa en una CPU :

El microcomputador, que es una versión reducida de las CPU convencionales (actuales), está conformado por sus periféricos de **hardware** : CPU , **RAM**, ROM, EEPROM, Timers, Conversor A/D, GPIO y el **software** que viene a ser elementos de texto conformado por instrucciones lógicas secuenciales a fin de que la CPU pueda hacer algo (o de sus periféricos) a modo de **tratamiento de información**

HARDWARE

- Es aquello que da soporte a las operaciones lógico-matemáticas que se van a ejecutar(sin intervención humana).
- Suele llevar un “software” preconstruido llamado firmware, y este es almacenado en la memoria **ROM**.
- Lo compone : CPU, **RAM**, Periféricos (E/S), Bus del sistema

TRATAMIENTO DE LA INFORMACION

El tratamiento de la información que pueden realizar los ordenadores(actuales) y/o microcomputadores consiste :

- Entrada de datos(del exterior, medio físico(**sensores**) o ingresado por el usuario)
- **Almacenamiento** de los **datos**
- **Procesamiento** Aritmético/lógico de los datos
- Almacenamiento de **resultados**(entregables)
- **Salida** de datos(hacia el exterior, medio físico(ex. Actuadores, pantallas, Red, consola)

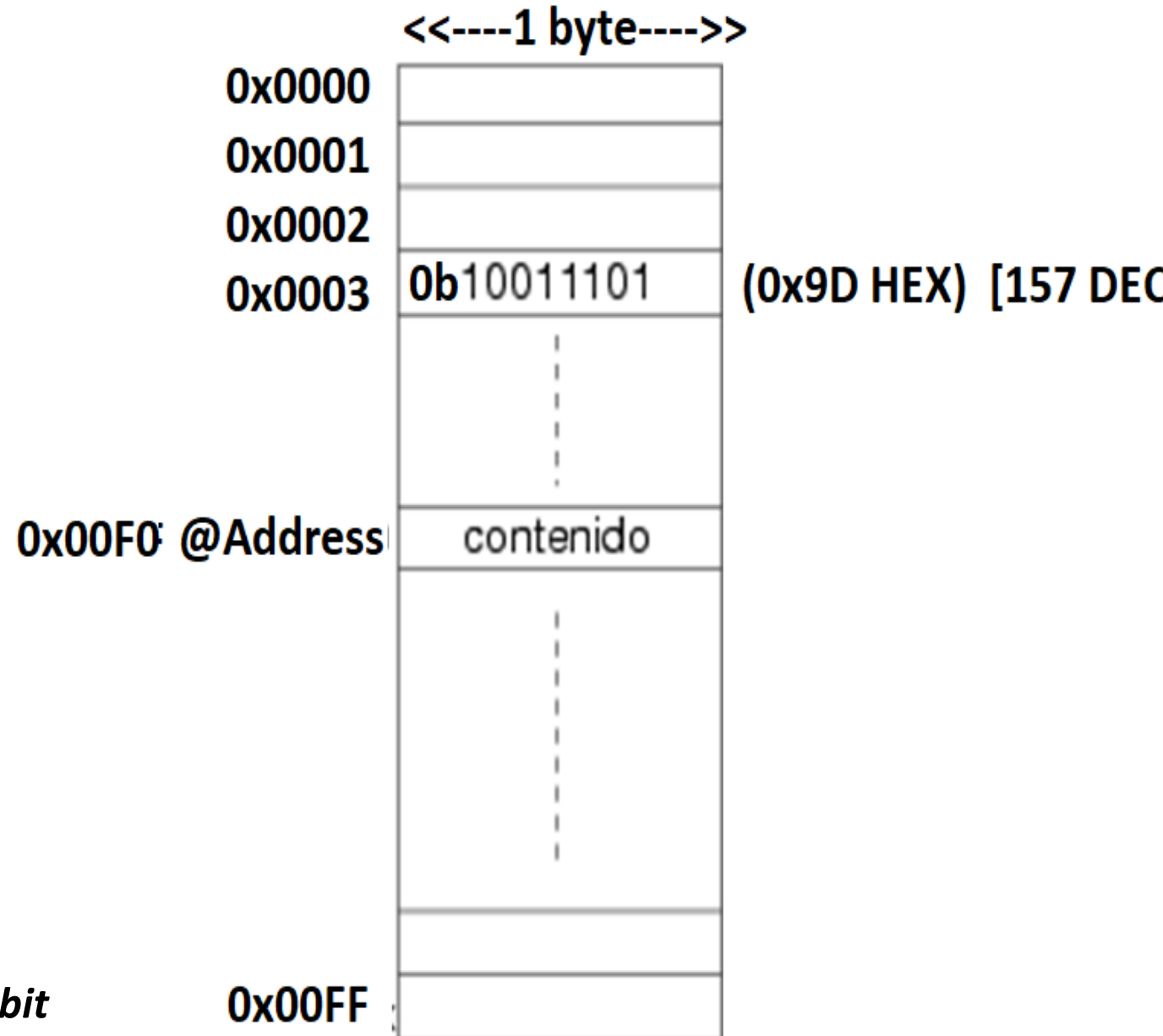
MEMORIA RAM (Random Access Memoery)

- Es donde se cargan los datos variables(o constantes) en el tiempo y las instrucciones que se van a ejecutar.
- Es volátil, su contenido se pierde en cada RESET o al ser apagado.
- Se compone de celdas consecutivas o posiciones idénticas numeradas por una **dirección de memoria @Address**, mediante la que se accede contenido o valor de cada celda.
- La **longitud de la palabra** de dirección puede ser de 8 bit, 16 bit, 32 bits, 64 bits; dependiendo de la arquitectura de la CPU

MEMORIA RAM

El contenido de una RAM, en cada celda puede ser accedido (**leído R**) o **Escrito W**

Ejemplo Memoria 1kB x 8bit
1kB, 1Mb, 1Gb x 8 bit

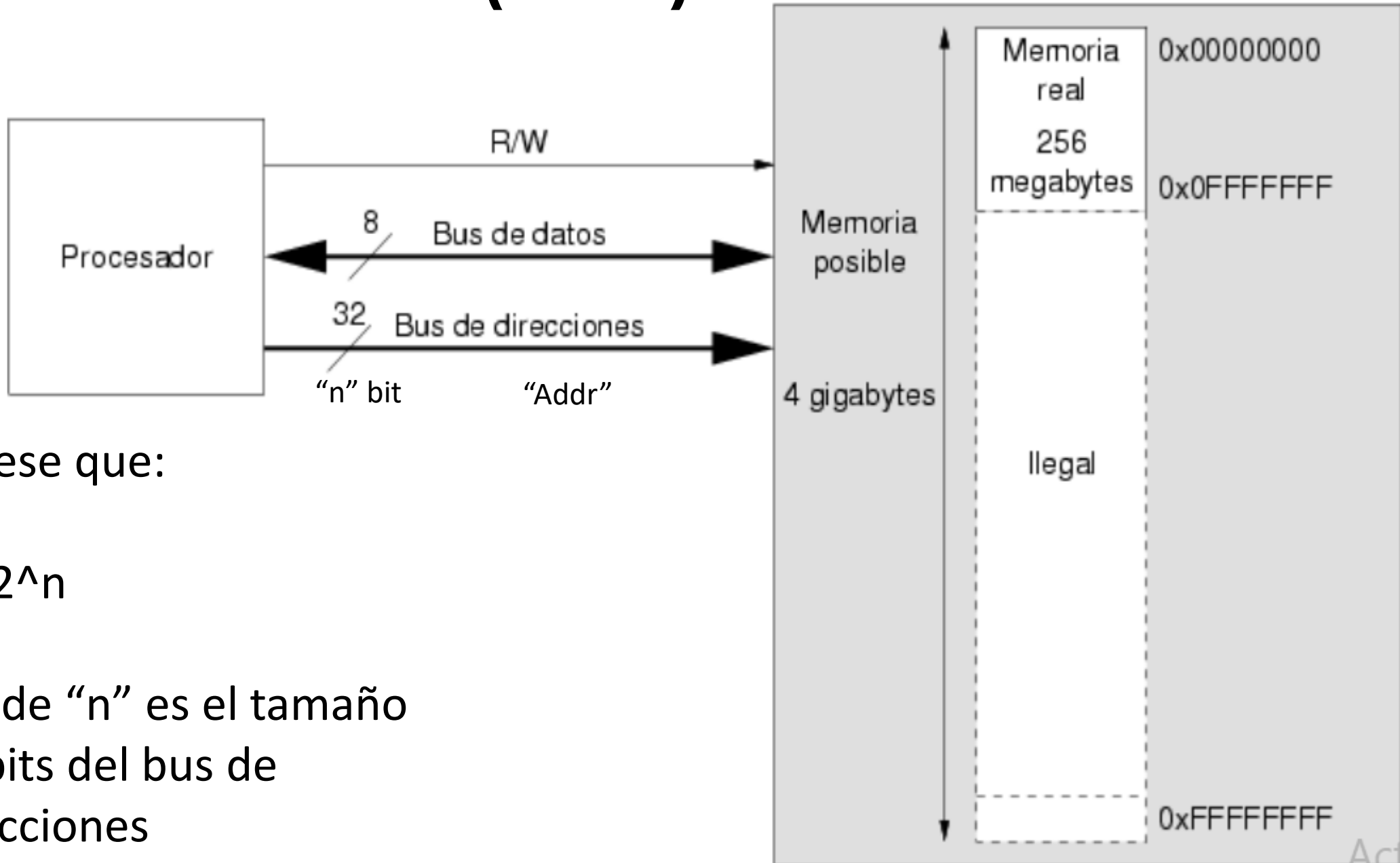


MEMORIA RAM

- La unidad de tamaño se suele expresar como **1 byte (8 bits)**, pero para expresar se suele usar los múltiplos en potencias de 2, por decir 2^{10} bytes equivale a 1024 bytes y esto se conoce como 1kB de RAM

| Potencia de 2 | Valor | Nombre/Abreviatura |
|---------------|----------------------|--------------------|
| 2^{10} | 1.024 | Kilobyte/kB |
| 2^{20} | 1.048.576 | Megabyte/MB |
| 2^{30} | 1.073.741.824 | Gigabyte/GB |
| 2^{40} | 1.099. 511. 627. 776 | Terabyte/TB |

CPU vs MEMORIA RAM (32 bit)

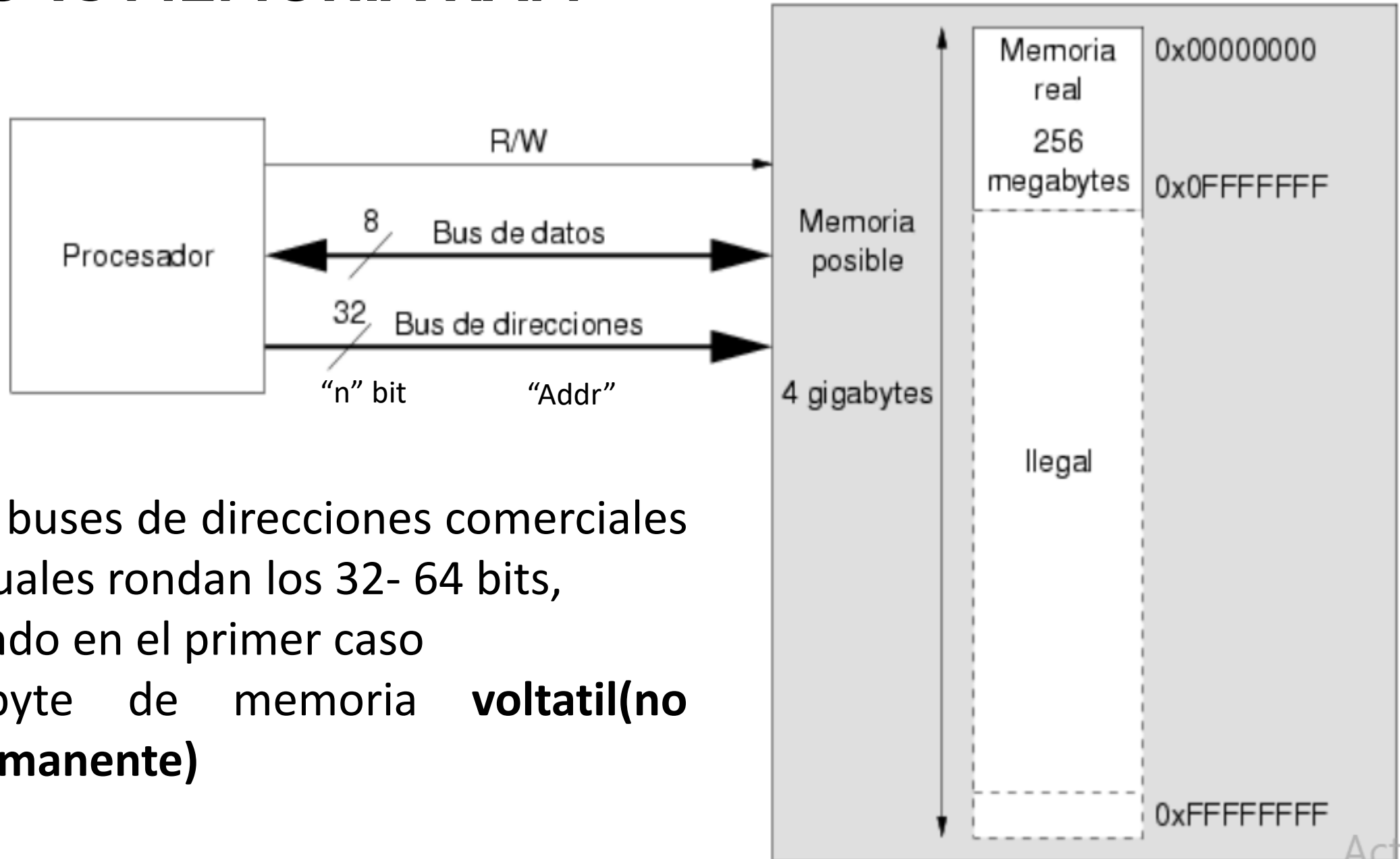


Nótese que:

$$T = 2^n$$

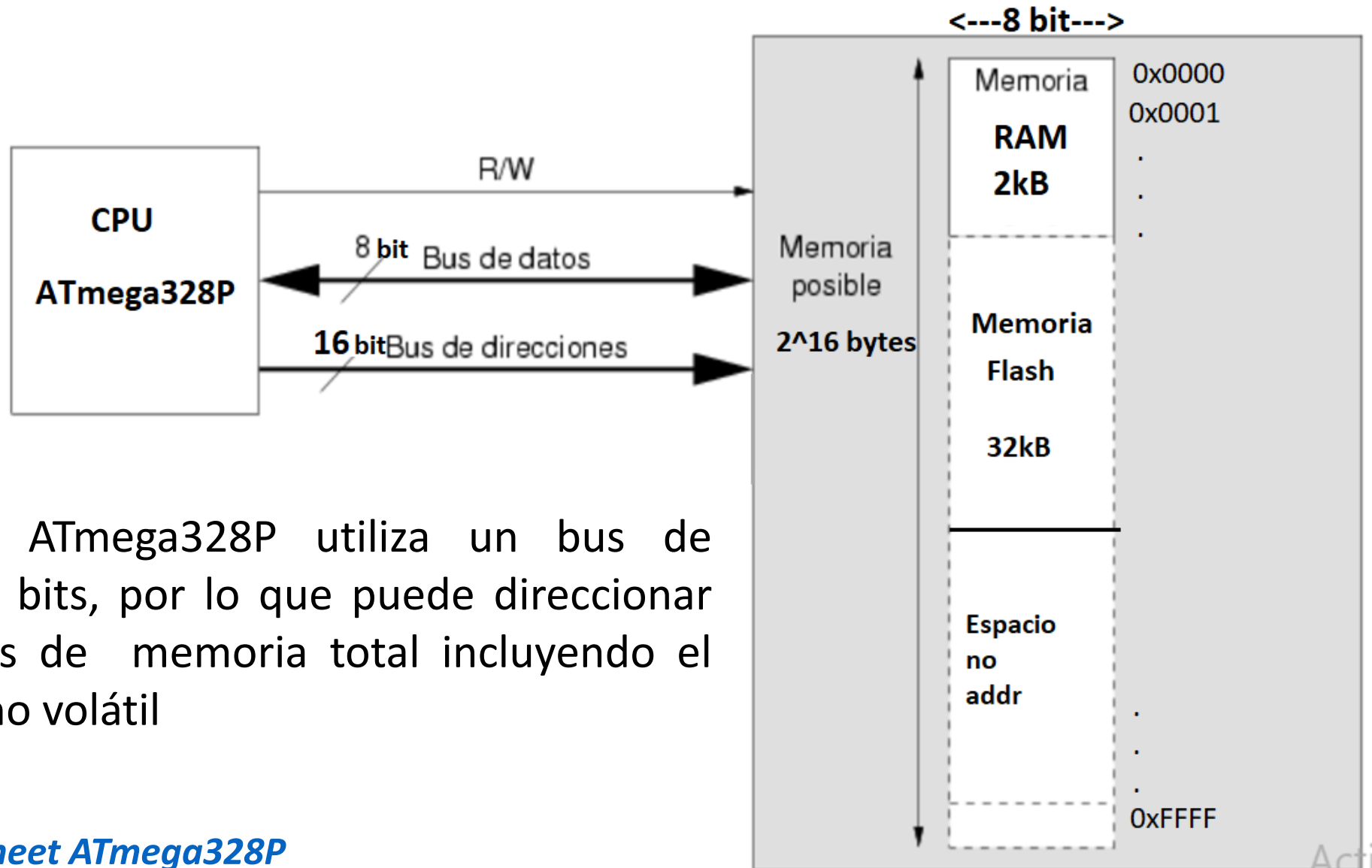
Donde "n" es el tamaño en bits del bus de direcciones

CPU vs MEMORIA RAM



Los buses de direcciones comerciales actuales rondan los 32- 64 bits,
Dando en el primer caso
4Gbyte de memoria **voltatil(no permanente)**

CPU vs MEMORIA RAM(uC)



La CPU del uC ATmega328P utiliza un bus de direcciones de 16 bits, por lo que puede direccionar hasta 65536 bytes de memoria total incluyendo el almacenamiento no volátil

Fuente: [Datasheet ATmega328P](#)

HARDWARE (resumen)

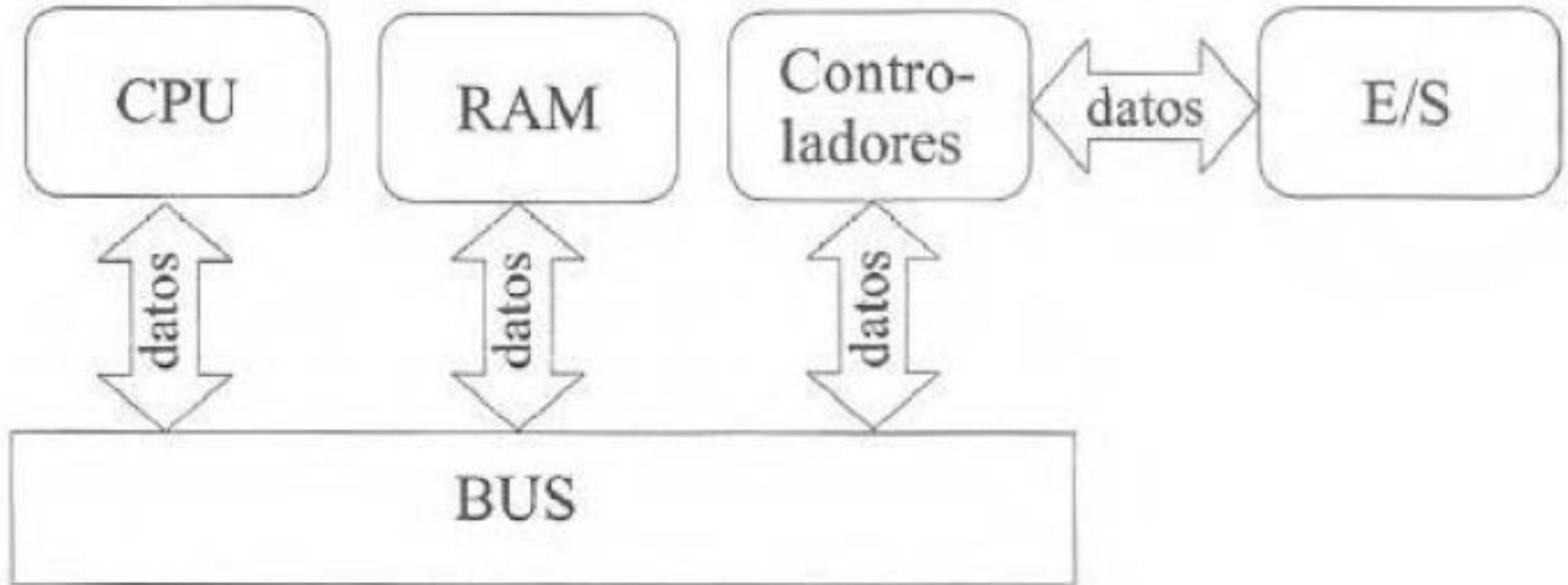


Figura 1.- Interacción entre componentes de un ordenador/microcomputador

SOFTWARE

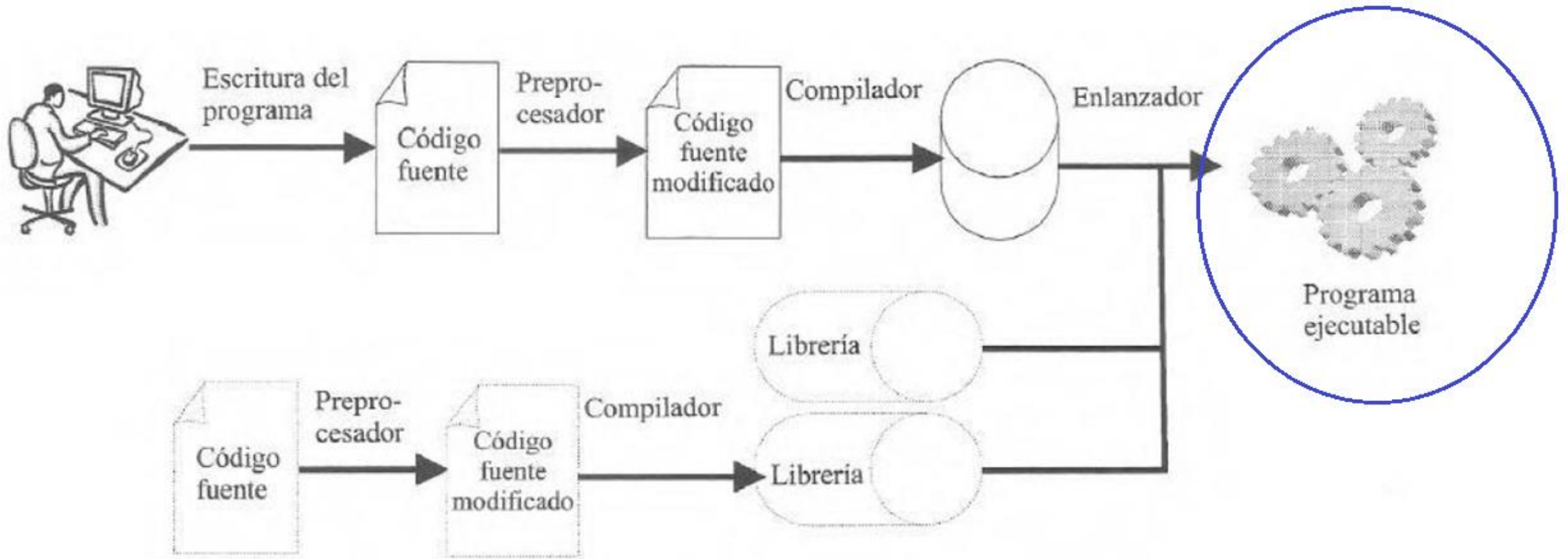
- Son el conjunto de elementos lógicos a fin que el hw haga tareas , dotándolo de capacidad de prestar servicios utiles
- Consta de 1 programa que a us vez esta compuesto por instrucciones, siendo esta ultima una “orden” predefinida dada en bits.
- Se suele preferir la **programación de alto nivel(C/C++, Java, C#, Android,..)**, motivos de productividad y portabilidad.
- Es almacenado de forma permanente(ROM, Flash)

SOFTWARE

| Lenguaje de alto nivel | Lenguaje ensamblador | Lenguaje máquina |
|---|----------------------|---------------------|
| <pre>suma=0; for (i=0; i<10; i++){ suma=suma+i;} printf ("%d",suma);</pre> | CAR SUM, 0 | 1111 0101 0000 0000 |
| | CAR i,0 | 1111 0111 0000 0000 |
| | BUCLE: ADD SUM,i | 1110 0101 0000 0000 |
| | ADD i,1 | 1110 0111 0000 0001 |
| | CAR AUX,i | 1111 0111 0000 0101 |
| | RES AUX,9 | 0111 1111 0000 1001 |
| | STZ SIGUE | 0001 1111 1010 1111 |
| | STI BUCLE | 0010 1111 1010 0011 |
| | SIGUE SAL i | 0110 0111 0000 0000 |

Los lenguajes de alto nivel facilitan la redacción de código , partiendo de una idea/concepto, y de forma independiente a una CPU

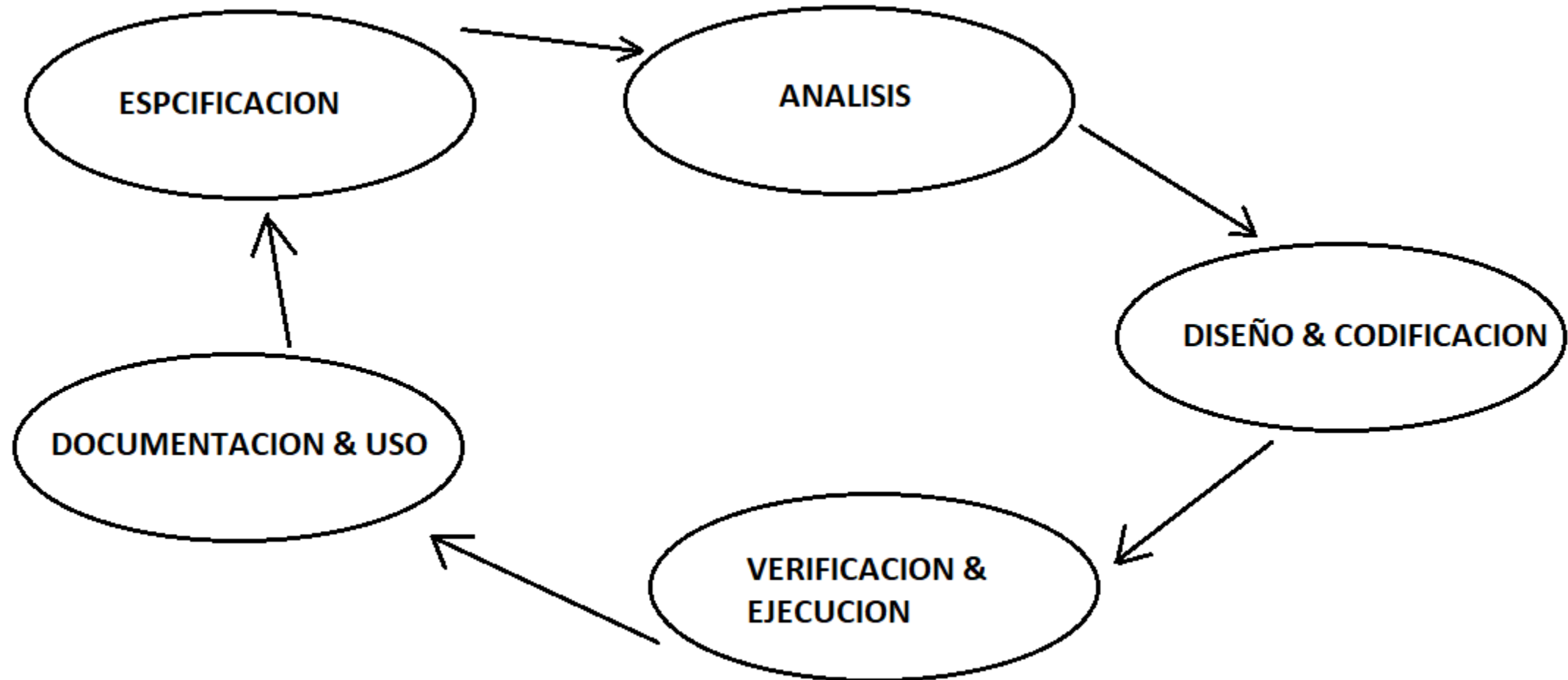
SOFTWARE (Flujo)



OBS: Las CPUs solo entienden lenguaje o código maquina 0....10001...

FASES DE CREACION DE UN SOFTWARE

Para crear un programa/software....




Para crear un programa/software....

- **Especificación.**- Alcances y limites del problema
- **Análisis .**- Problema → Tareas mas pequeñas y realizables
- **Diseño y Codificación .**- Algoritmo, lenguaje ?
- **Ejecución/verificación.**- errores? , salidas correctas?
- **Explotación.**- Uso ininterrumpido.....errores(con el tiempo: **BUGS**)?

TIPOS Y REPRESENTACION DE DATOS ANSI C99

Tipos de datos

- Los datos , a nivel de software, se representan como byte o bytes($N \times 8$ bit), y pueden tener una interpretación diferente(orientado al contexto)
- En general, los tipos de dato básicos (Lenguaje C) son ***char, int, float, double y void***
 - **char**: Almacena un único carácter, como letras, números o símbolos.
 - **int**: Almacena números enteros sin decimales.
 - **float**: Almacena números de punto flotante con una precisión de 6 dígitos.
 - **double**: Almacena números de punto flotante con una precisión de 15 dígitos.
 - **void**: Se utiliza para indicar que una función no devuelve ningún valor o para declarar punteros genéricos. 

Tipos de datos(derivados)

- Pueden modificar el signo y/o entender la magnitud: **signed**, **unsigned**; **short**, **long**
- Usando el método **sizeof()** puede saberse el **tamaño en bytes** de una variable o tipo

Tipos de datos (Tamaño)

| Tipo | Almacenamiento |
|---|----------------|
| <code>char</code> , <code>unsigned char</code> , <code>signed char</code> | 1 byte |
| <code>short</code> , <code>unsigned short</code> | 2 bytes |
| <code>int</code> , <code>unsigned int</code> | 4 bytes |
| <code>long</code> , <code>unsigned long</code> | 4 bytes |
| <code>long long</code> , <code>unsigned long long</code> | 8 bytes |
| <code>float</code> | 4 bytes |
| <code>double</code> | 8 bytes |
| <code>long double</code> | 8 bytes |

Fuente:

[Doc Microsoft](#)

`Int *pUnteroVar=`

Representación de datos (RAM) de bytes o letras

- Un dato de tipo **char** (ex. 'A')(0x41) (65 DEC), de tamaño **1 byte**, tan solo ocupara una celda en la RAM. Y una cadena de letras(texto) ocupara tantas celdas contiguas en la RAM a partir de cierta dirección **@Addr**
- La representación de letras se rige bajo la normativa de [código ASCII](#)

Representación de datos (RAM) de bytes o letras

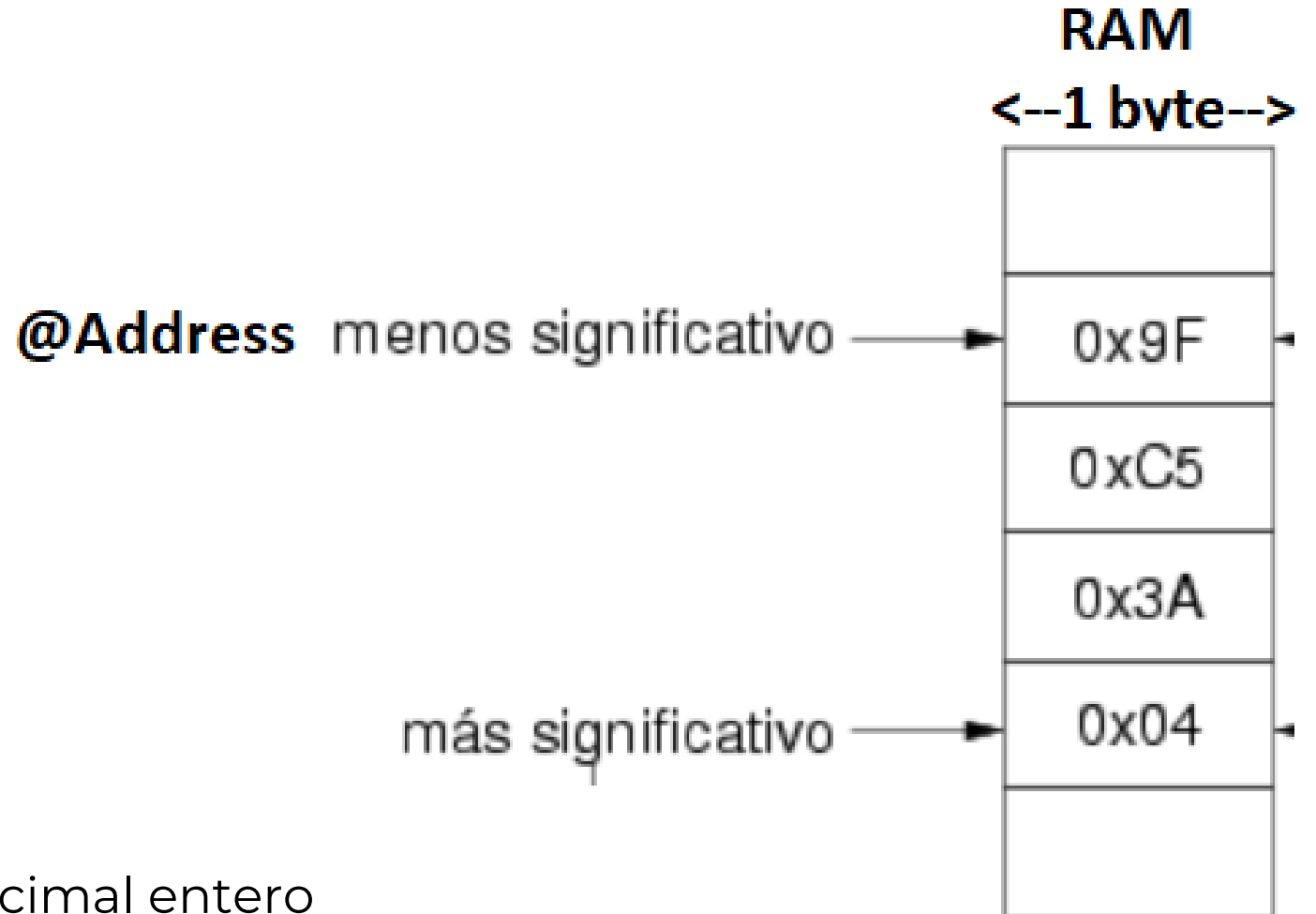
| | 1 byte | |
|-------|--------|-----|
| 0x120 | 0x4D | "M" |
| 0x121 | 0x69 | "i" |
| 0x122 | 0x20 | " " |
| 0x123 | 0x70 | "p" |
| 0x124 | 0x72 | "r" |
| 0x125 | 0x79 | "i" |
| 0x126 | 0x6D | "m" |
| 0x127 | 0x65 | "e" |
| 0x128 | 0x72 | "r" |
| 0x129 | 0x70 | " " |
| 0x12A | 0x20 | "p" |
| 0x12B | 0x72 | "r" |
| 0x12C | 0x6F | "o" |
| 0x12D | 0x67 | "g" |
| 0x12E | 0x70 | "r" |
| 0x12F | 0x61 | "a" |

Representación de datos (RAM) de numéricos o enteros

- Un dato entero **int** de tamaño ocupara tantas celdas contiguas en la RAM como su tamaño en bytes (4 bytes) , a partir de cierta dirección **@Addr**
- Si el tamaño es **de 4 bytes**, entonces los bytes menos(mas) significativos se cargan en RAM a partir de @Addr.
- El orden en que se almacena puede ser de Little Endian o Big Endian

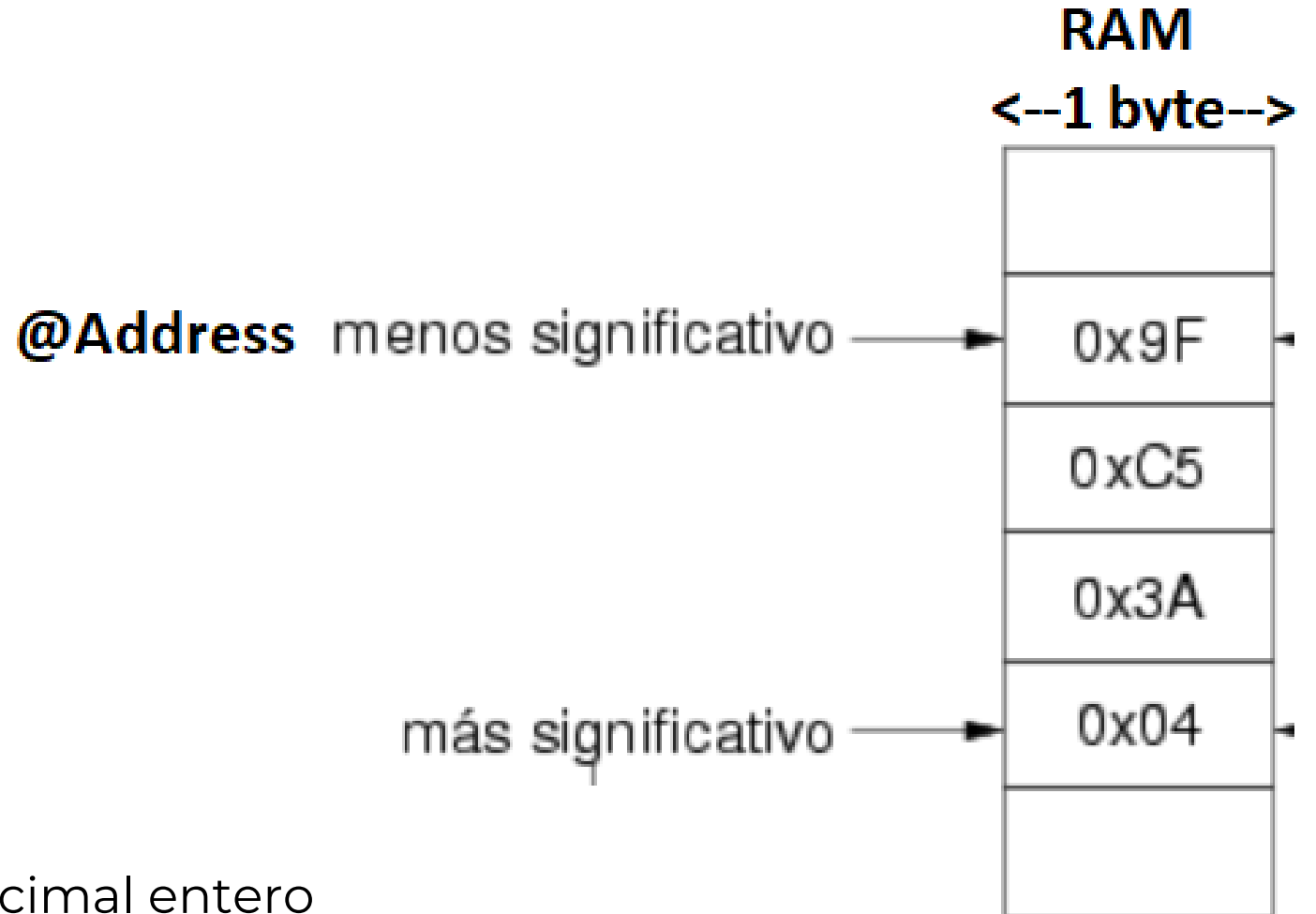
Representación de datos (RAM) de enteros

Ex. Representar el numero decimal entero (int) : **70960543**



Representación de datos (RAM) de double

Ex. Representar el numero decimal entero
(**double**) : **1.6** x 10 ⁻¹⁹

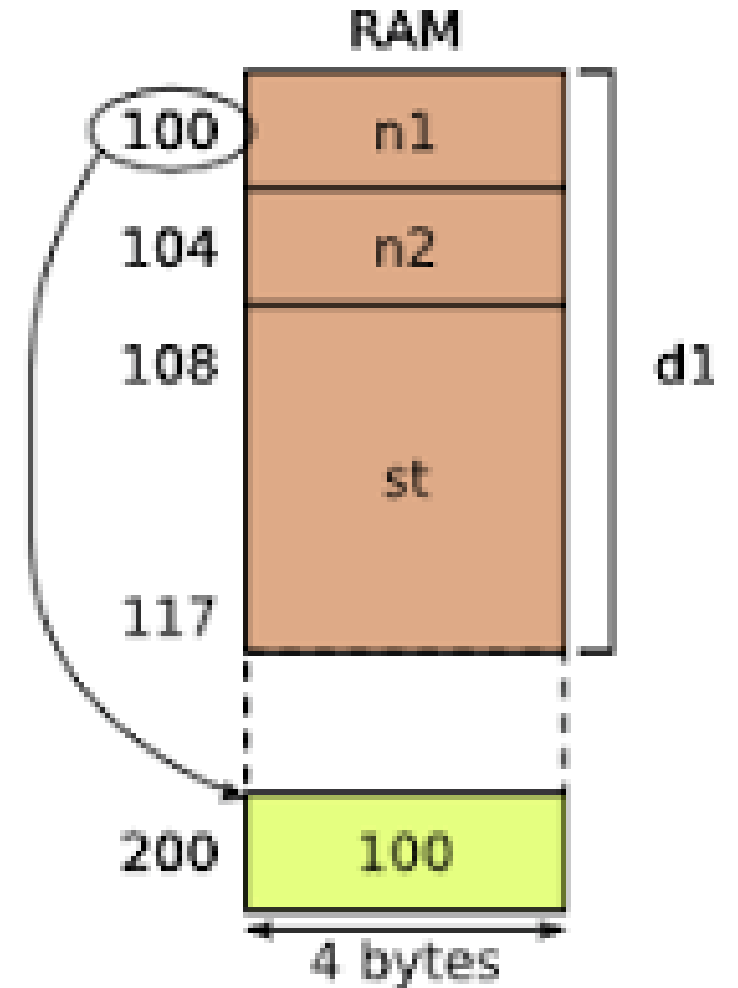


Representación de datos (RAM)

Ex. Representar varios tipos

DECLARANDO VARIABLES

```
int n1;  
float n2;  
char st[10];
```



Tipos de datos (Rangos)

- **char**, signed char: 8 bits, rango de -128 a 127
- unsigned char: 8 bits, rango de 0 a 255
- short int, signed short int: 16 bits, rango de -32768 a 32767
- unsigned short int: 16 bits, rango de 0 a 65535
- **int**, signed int, long int, signed long int: 32 bits, rango de -2147483648 a 2147483647
- unsigned int, unsigned long int: 32 bits, rango de 0 a 4294967295
- long long: 64 bits, rango de -9223372036854775808 a 9223372036854775807
- **float**: 32, seis dígitos de precisión con un rango de $3,4 * 10^{-38}$ a $3,4 * 10^{38}$.
- **double**: 64, diez dígitos de precisión con un rango de $1,7 * 10^{-308}$ a $1,7 * 10^{308}$.
- long double: 96, 17 dígitos de precisión con un rango de $3,4 * 10^{-4932}$ a $1,1 * 10^{4932}$.

Declaración de variables

- **<Tipo dato variable>** nombre_variable ; *// "Comentario"*
- <Tipo dato retorno> nombre_función(<Tipo dato variable> arg1, <Tipo dato variable> arg2,...) ; *// "Comentario"*


C



```
char letra;           // Declara una variable de tipo char
int numero;           // Declara una variable de tipo int
float decimal;        // Declara una variable de tipo float
double real;          // Declara una variable de tipo double
void mi_funcion();     // Declara una función que no devuelve valor
```

Operadores lógicos

Operadores Lógicos:

- **AND lógico (&):** Devuelve 1 si ambos operandos son distintos de cero, 0 de lo contrario.
- **OR lógico (||):** Devuelve 1 si al menos uno de los operandos es distinto de cero, 0 si ambos son cero.
- **NOT lógico (!):** Invierte el valor de un operando. Devuelve 1 si el operando es cero, 0 si es distinto de cero. 

Operadores de Bits:

- **AND bit a bit (&):** Realiza una operación AND en cada bit de los operandos.
- **OR bit a bit (|):** Realiza una operación OR en cada bit de los operandos.
- **XOR bit a bit (^):** Realiza una operación XOR en cada bit de los operandos.
- **Desplazamiento a la izquierda (<<):** Desplaza los bits de un operando hacia la izquierda.

Código ejemplo- Operadores lógicos

```
#include <stdio.h>

int main() {
    int a = 5, b = 10;

    // Operadores Lógicos
    if (a > 0 && b < 15) {
        printf("Ambas condiciones son verdaderas\n");
    }

    if (a > 5 || b < 10) {
        printf("Al menos una condición es verdadera\n");
    }

    if (!a) {
        printf("a es 0\n");
    }

    // Operadores de bits
    int resultado = a & b; // AND bit a bit
    printf("Resultado AND bit a bit: %d\n", resultado);

    resultado = a | b; // OR bit a bit
    printf("Resultado OR bit a bit: %d\n", resultado);

    resultado = a ^ b; // XOR bit a bit
    printf("Resultado XOR bit a bit: %d\n", resultado);

    resultado = a << 2; // Desplazamiento a la izquierda
    printf("Resultado desplazamiento a la izquierda: %d\n", resultado);

    resultado = a >> 1; // Desplazamiento a la derecha
    printf("Resultado desplazamiento a la derecha: %d\n", resultado);

    return 0;
}
```