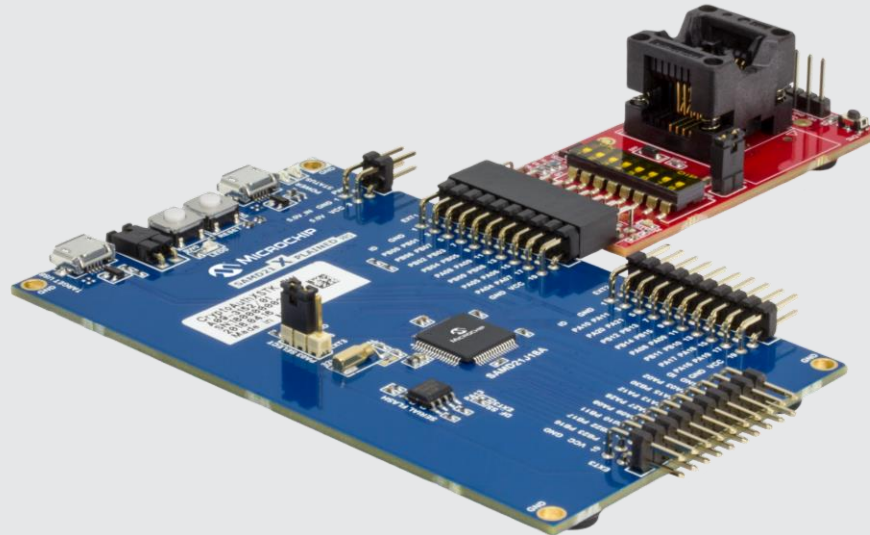


ANSI C

Lenguaje C para Sistemas Embebidos



Lenguaje C



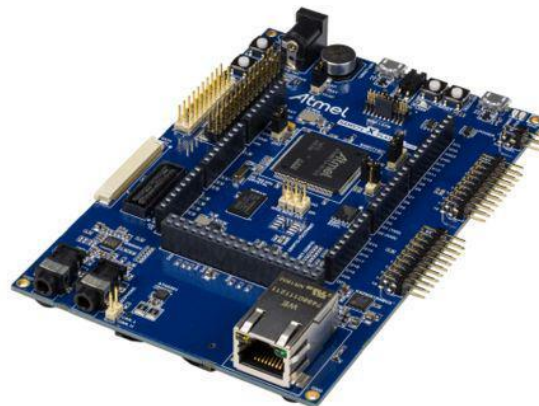
Instructor: Ivan Vargas Alvarado

Investigador y Desarrollador en el Área de
Sistemas Embebidos



Profesor: Ivan Vargas Alvarado

- Los **Sistemas Embebidos (embedded systems)** son sistemas operativos que son creados con la finalidad de ser controlados por microprocesadores o microcontroladores.
- Los sistemas embebidos surgen para cubrir necesidades específicas y no necesidades generales como las que cubre un ordenador
- Su funcionamiento en términos generales consta de:
 - Entrada (sensores y/o periféricos)
 - Proceso (Tiempo real)
 - Salida (respuesta, resultados, periféricos)



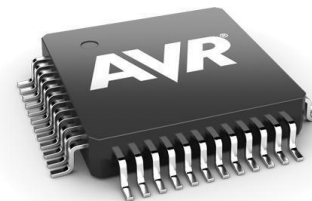
Es posible dividir el área de los sistemas embebidos en dos:

- **Hardware embebido:** es la parte física del sistema como: la placa base integrada y todos los componentes electrónicos de potencia que lo conforman.



- **Software embebido:** es la primera capa de código que se ejecuta en el controlador.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
000	4D	5A	80	00	01	00	00	00	04	00	10	00	FF	FF	00	00
010	40	01	00	00	00	00	00	00	40	00	00	00	00	00	00	00
020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
030	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	00
040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68
050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F
060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20
070	6D	6F	64	65	2E	0D	0A	24	00	00	00	00	00	00	00	00
080	50	45	00	00	4C	01	04	00	D2	4C	A5	4A	00	00	00	00
090	00	00	00	00	B0	00	0E	01	0E	01	01	43	00	00	00	00
0A0	00	00	00	00	00	00	00	00	00	10	00	00	00	00	00	00
0B0	00	00	00	00	00	00	40	00	00	10	00	00	00	02	00	00
0C0	01	00	00	00	00	00	00	00	03	00	0A	00	00	00	00	00
0D0	00	50	00	00	00	04	00	00	2F	79	00	00	02	00	00	00
0E0	00	10	00	00	00	10	00	00	00	00	01	00	00	00	00	00
0F0	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	00
100	00	30	00	00	80	00	00	00	00	00	00	00	00	00	00	00

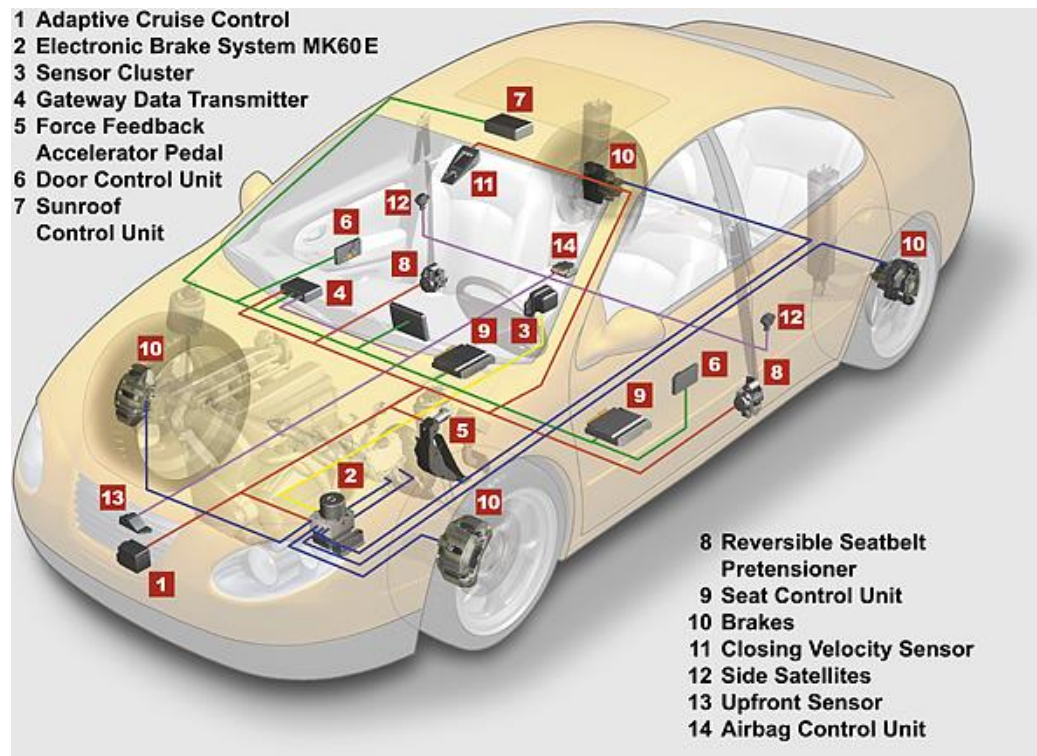


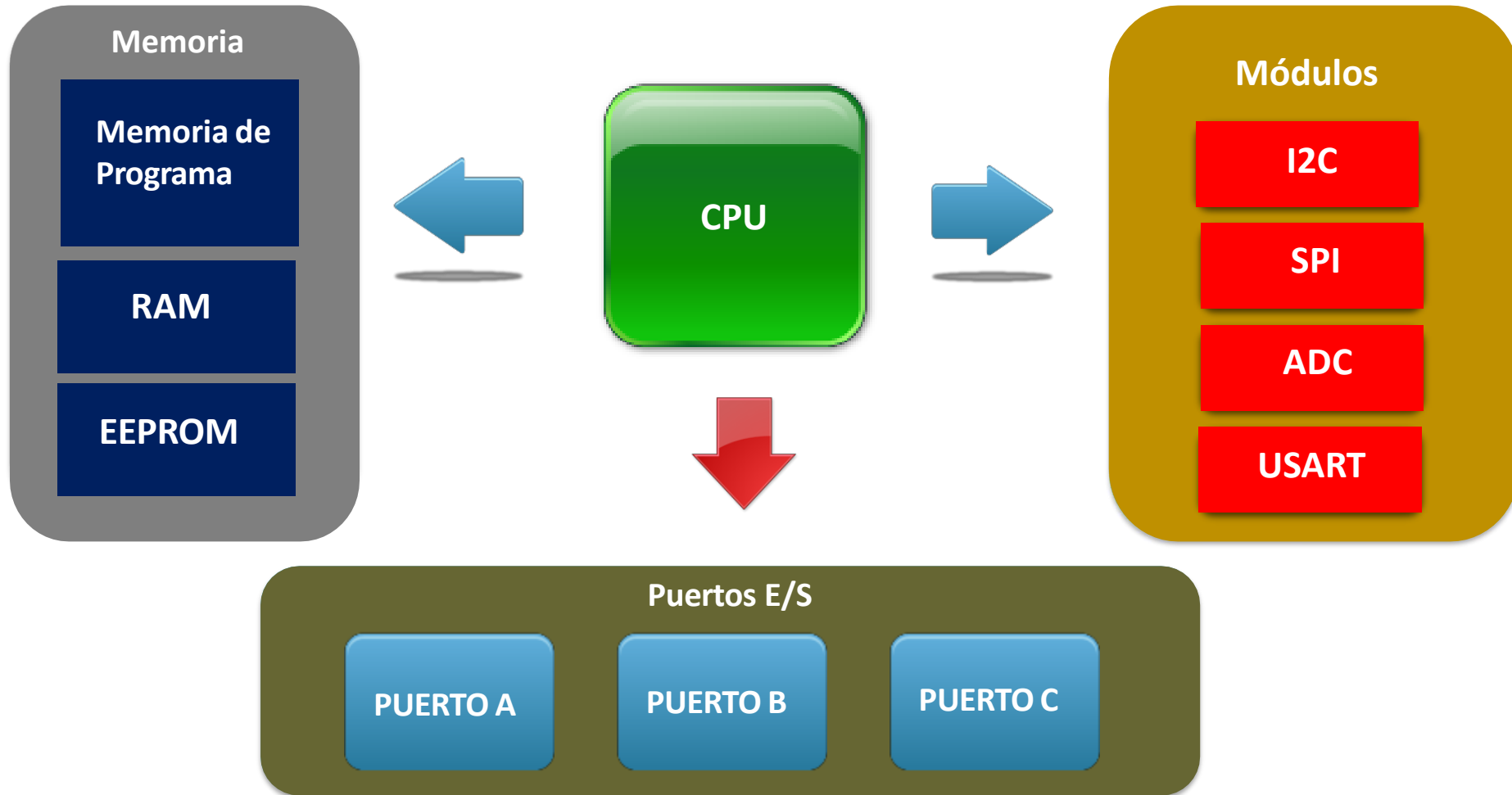
El diseño de un sistema embebido usualmente se orienta a:

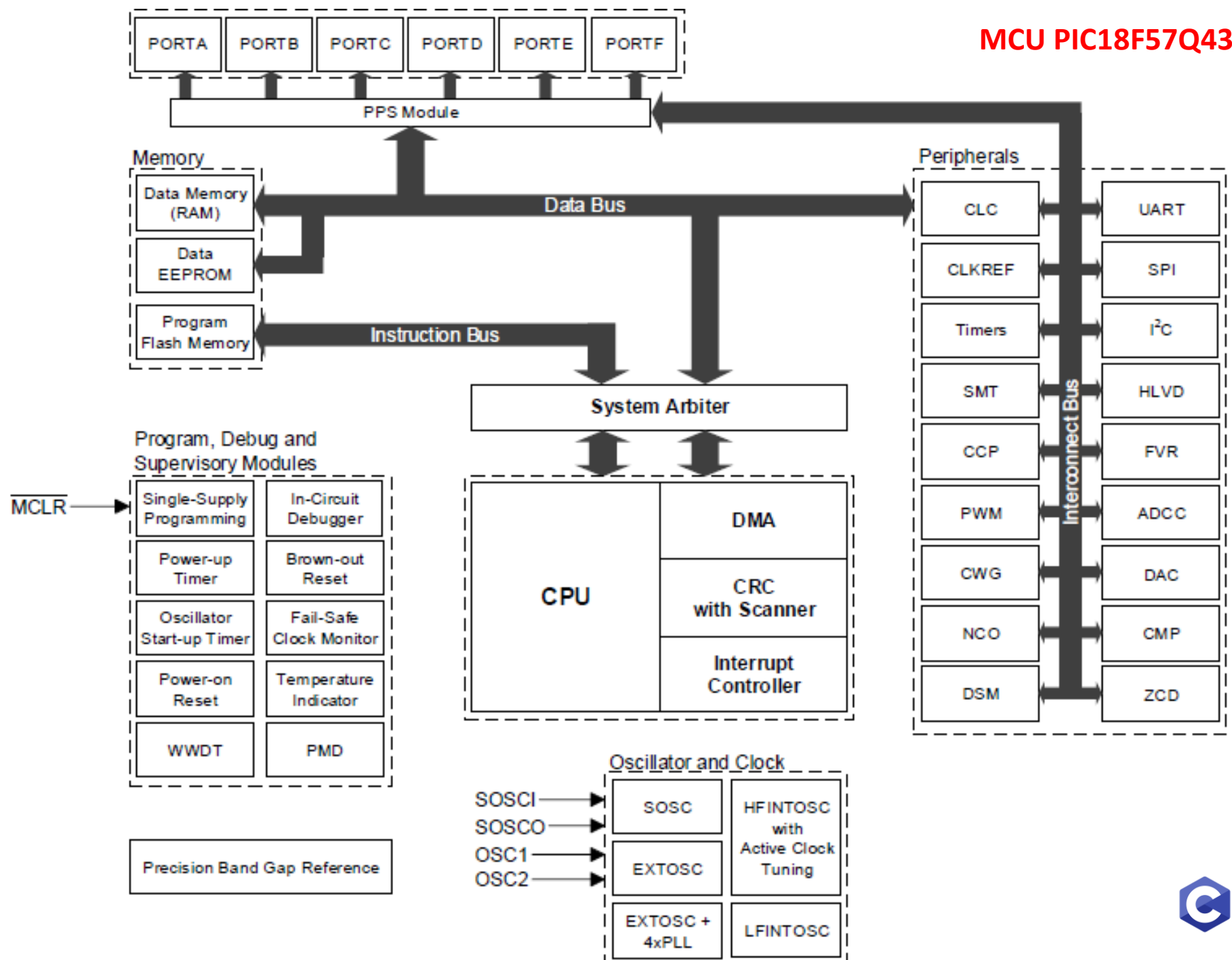
- Reducir su tamaño, su consumo y su costo
- Aumentar su confiabilidad
- Mejorar su desempeño
- Asegurar su determinismo y su tiempo de respuesta
- Atender la mayor cantidad de tareas posibles etc.

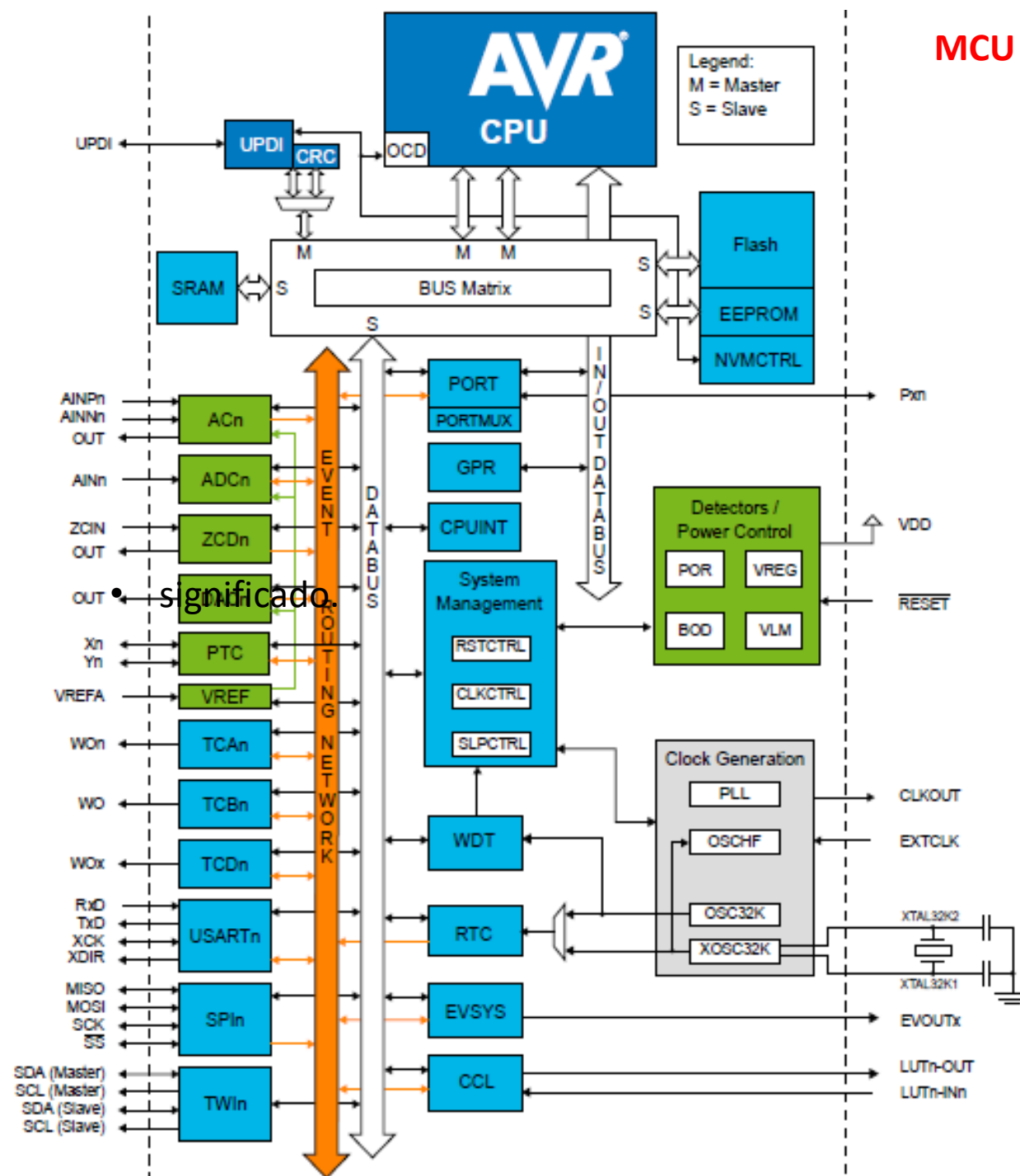


Los **sistemas embebidos** están diseñados para hacer una tarea en específico. Pero esto no significa que sean dispositivos independientes. Muchos de los sistemas embebidos consisten en pequeñas partes computarizadas dentro de un gran dispositivo que sirve un propósito en general. Por ejemplo un automóvil:

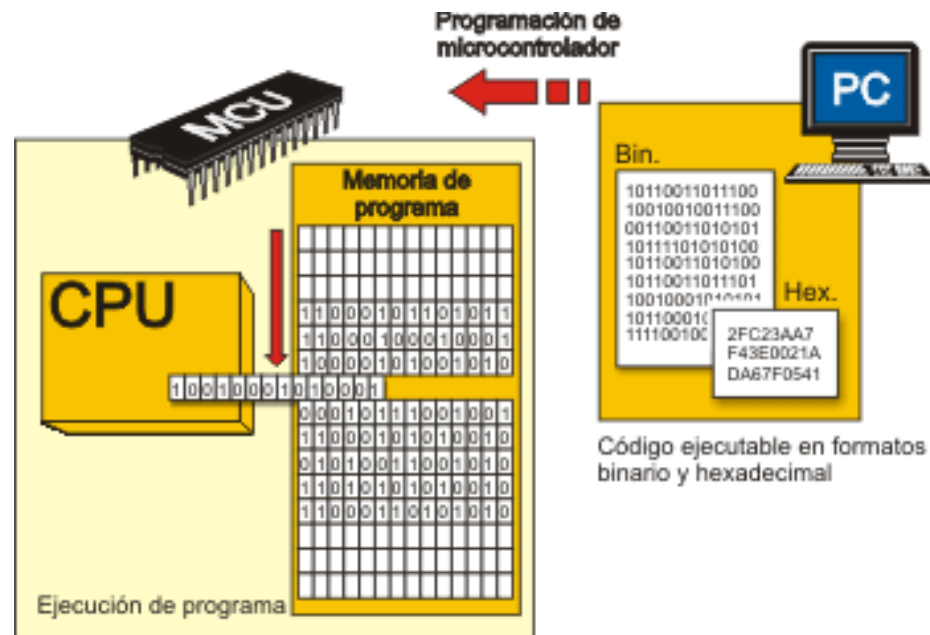






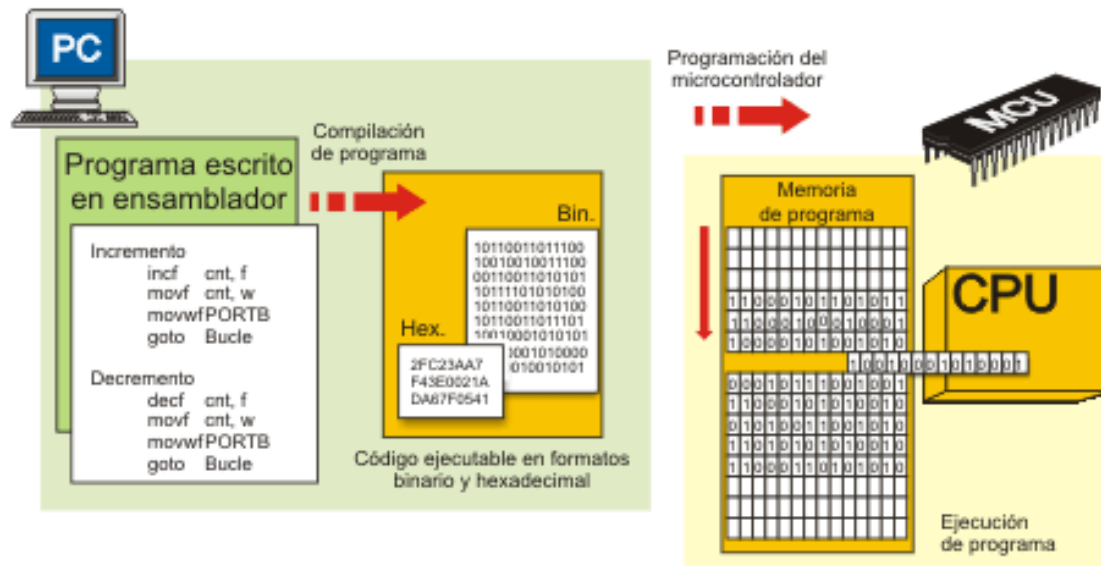


- El **microcontrolador** ejecuta el programa cargado en la memoria Flash. Esto se denomina el código ejecutable y está compuesto por una serie de ceros y unos, aparentemente sin significado.
- Dependiendo de la arquitectura del microcontrolador, el código binario está compuesto por palabras de 12, 14 o 16 bits de anchura. Cada palabra se interpreta por la CPU como una instrucción a ser ejecutada durante el funcionamiento del microcontrolador.



El lenguaje de máquina: Es el conjunto de instrucciones que pertenecen exclusivamente a una máquina, y solamente esta puede entenderlas.

El lenguaje ensamblador: Como el proceso de escribir un código ejecutable es considerablemente arduo, en consecuencia fue creado el primer lenguaje de programación denominado ensamblador (ASM). Las instrucciones en ensamblador consisten en las abreviaturas con significado y a cada instrucción corresponde una localidad de memoria. Un programa denominado ensamblador que traduce las instrucciones en ASM a código máquina.



A pesar de todos los lados buenos, el lenguaje ensamblador tiene algunas desventajas:

- Incluso una sola operación en el programa escrito en ensamblador consiste en muchas instrucciones, haciéndolo muy largo y difícil de manejar.
- Cada tipo de microcontrolador tiene su propio conjunto de instrucciones que un programador tiene que conocer para escribir un programa.
- Un programador tiene que conocer el hardware del microcontrolador para escribir un programa.

**ME DIJISTE QUE ERAS
UN LENGUAJE COMPILADO**



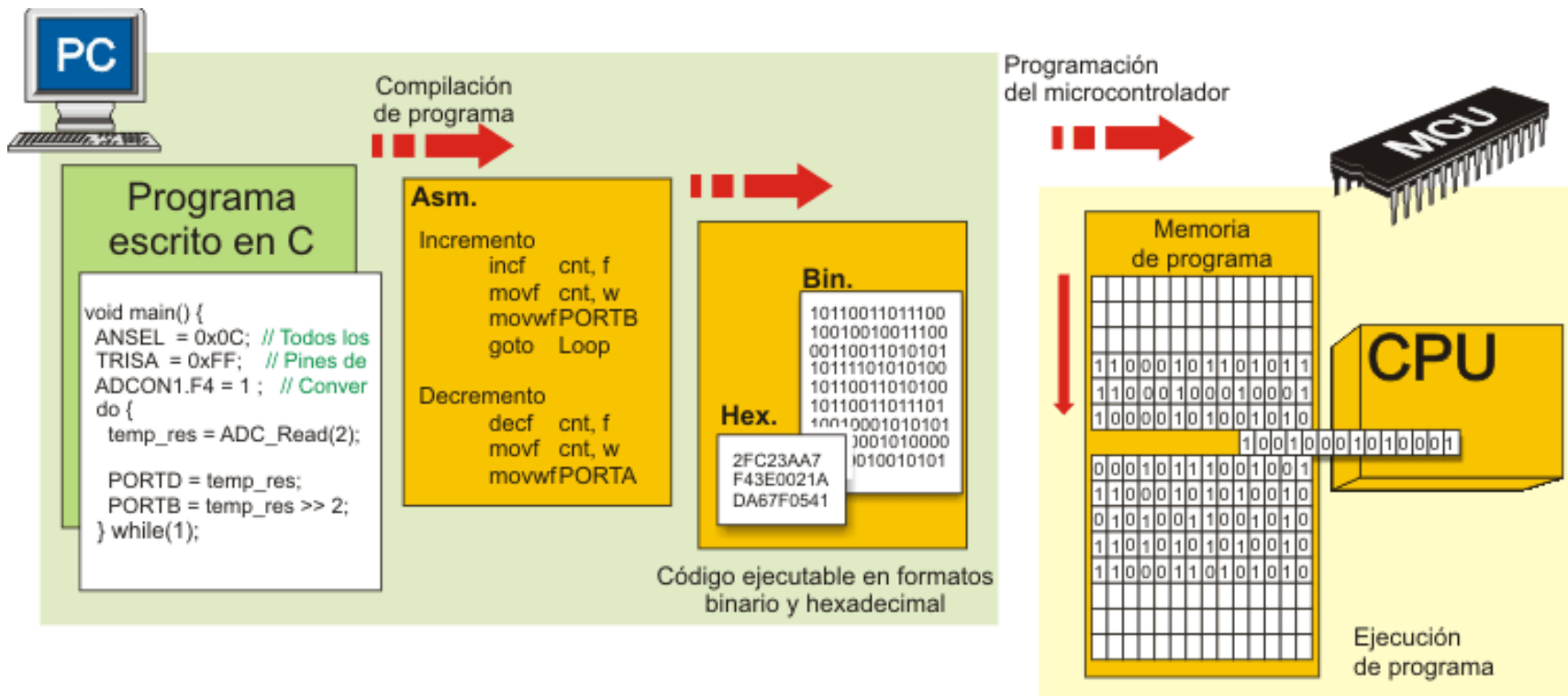
- Los lenguajes de programación de alto nivel (Basic, Pascal, C etc.) fueron creados con el propósito de superar las desventajas del ensamblador.
- En lenguajes de programación de alto nivel varias instrucciones en ensamblador se sustituyen por una sentencia.
- El programador ya no tiene que conocer el conjunto de instrucciones o características del hardware del microcontrolador utilizado.



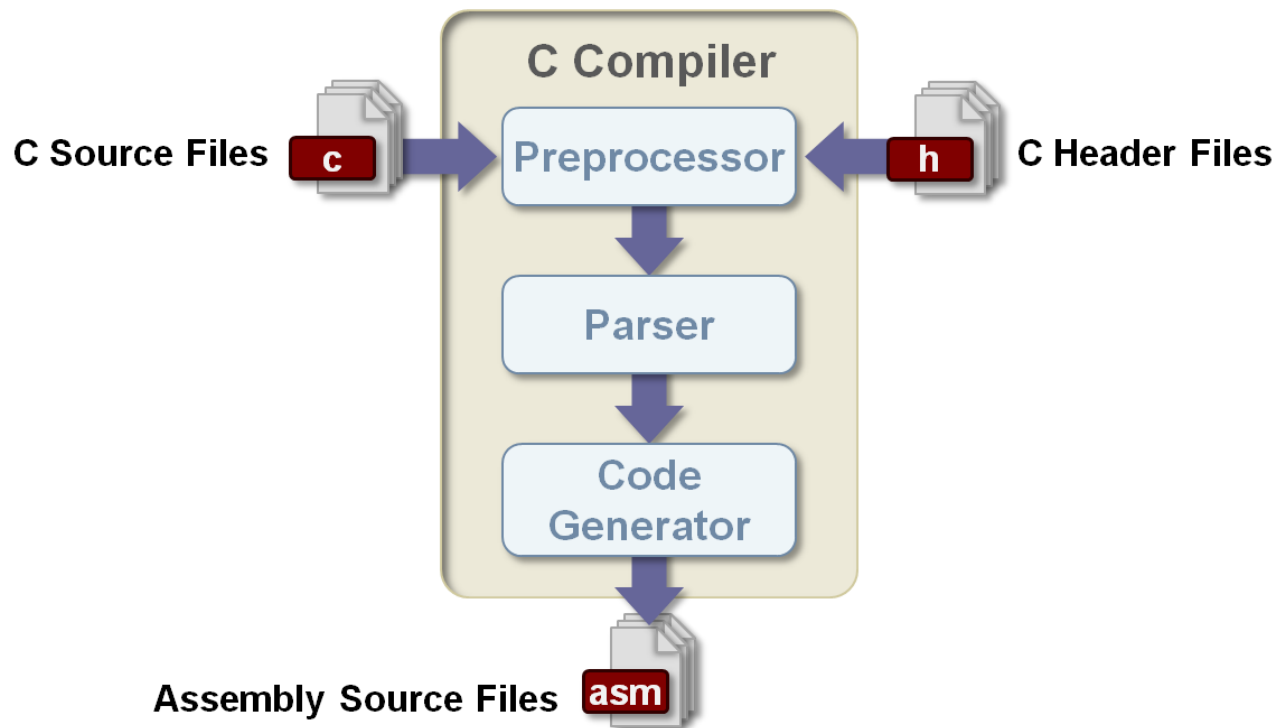
- **C** fue desarrollado en 1972 para escribir el sistema operativo Unix
- Una de las mejores características **lenguaje C** es que no está atado bajo ningún hardware o sistema en particular.
- **El lenguaje C** es ubicado como un lenguaje de programación de nivel medio, que combina los elementos de un lenguaje de alto nivel, con la eficiencia de un lenguaje ensamblador.
- Debido a la naturaleza de bajo nivel de la programación del sistema operativo, lo hace muy bueno para programación de **MCUs**.
- El primer estándar del **lenguaje C** se desarrollo en 1989 por ANSI, conocido como **ANSI C**
- **C** es compatible con compiladores para una amplia variedad de arquitecturas MCU
- **C** puede hacer casi cualquier cosa que el lenguaje ensamblador pueda hacer
- **C** suele ser más fácil y rápido para escribir código que el lenguaje ensamblador



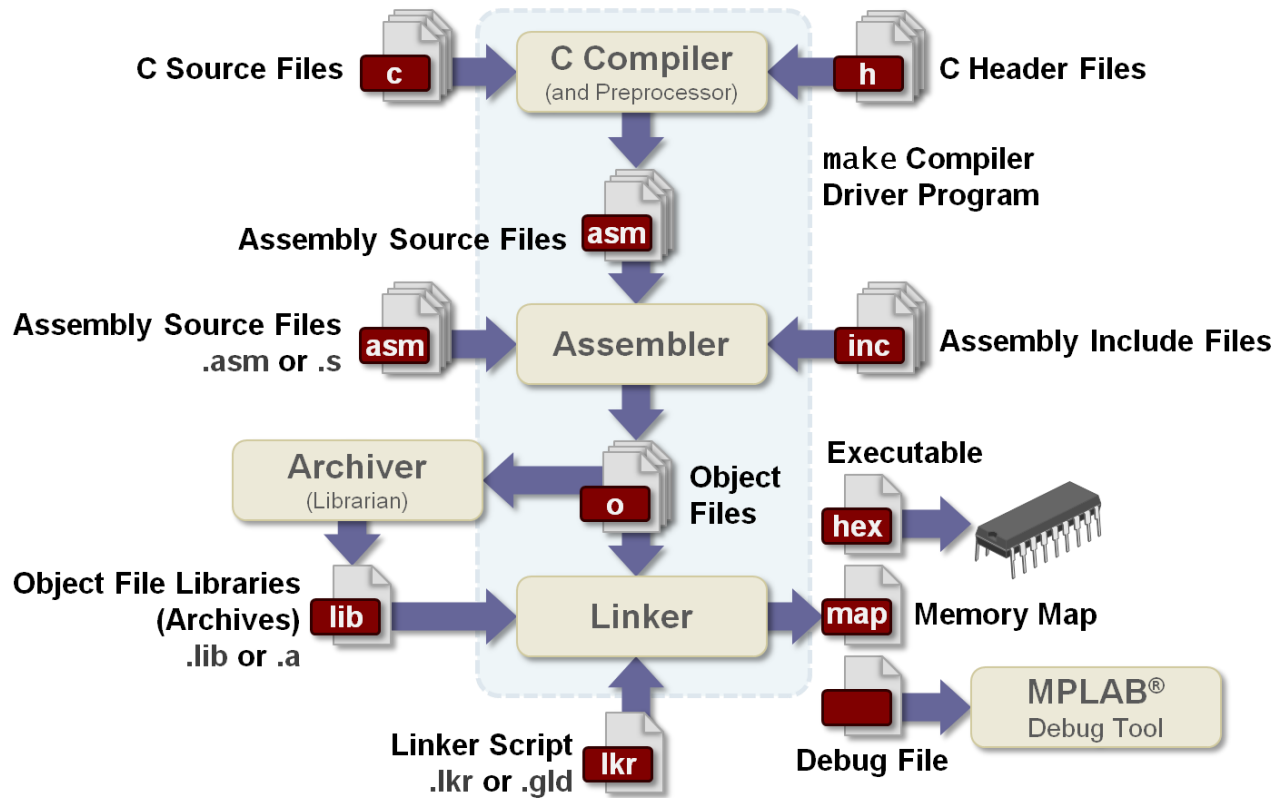
- Compilación de programa de un lenguaje de programación de alto nivel a bajo nivel.



El **compilador de C** consta de tres componentes principales: el preprocesador, el analizador y el generador de código



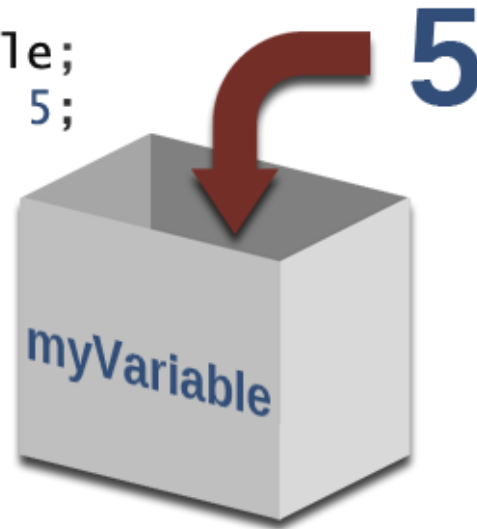
Suceden muchas cosas desde el momento en que presionas el botón "**compilar**" hasta que el **MCU** está ejecutando tu código.



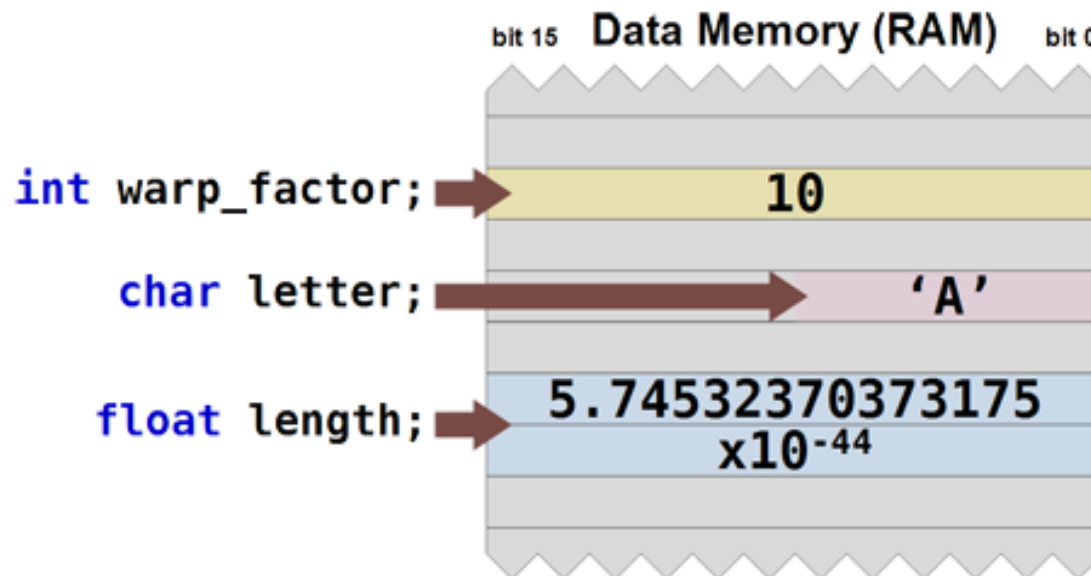
Una variable es un **nombre asignado a una o varias posiciones de memoria RAM** que se utilizan para almacenar los datos de un programa.

Una variable es simplemente un **contenedor para almacenar datos**, donde el contenedor ocupa uno o más bytes en la memoria RAM del microcontrolador.

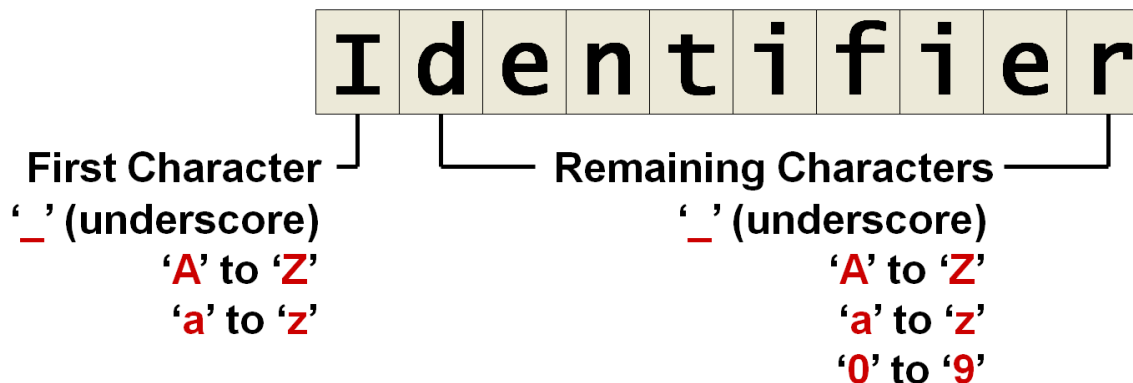
```
int myVariable;  
myVariable = 5;
```



Representación visual de cómo se pueden almacenar tres variables en la **memoria de datos de 16 bits** de un Microcontrolador de la familia **PIC24**.



- Un **identificador** es un nombre que se le da a un elemento de programa, como una **variable**, **función** o **matriz**. Este nombre puede usarse para referirse al elemento del programa sin conocer su ubicación específica en la memoria.
- Los **identificadores** en C deben ser cadenas de caracteres que incluye todas las letras del alfabeto inglés (tanto mayúsculas como minúsculas), los números del 0 al 9 y el guion bajo.
- El **primer carácter** de un identificador **NO** debe ser un número y bajo ninguna circunstancia un identificador puede contener un espacio.



Las **palabras reservadas** son palabras que tienen un significado especial para el compilador y no pueden usarse como identificadores.

Estas 32 palabras clave tienen significados específicos en ANSI C

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while



Un **tipo de dato** define los requisitos de almacenamiento, los requisitos de manipulación y el comportamiento de las variables y los parámetros de función.

Tipos de datos fundamentales en C:

Type	Description	Size (bits)
<code>char</code>	Single Character	8
<code>int</code>	Integer	16
<code>float</code>	Single Precision Floating Point Number	32
<code>double</code>	Double Precision Floating Point Number	64

Hay dos tipos de datos más: `void` y `enum` que se usan para aplicaciones especiales.

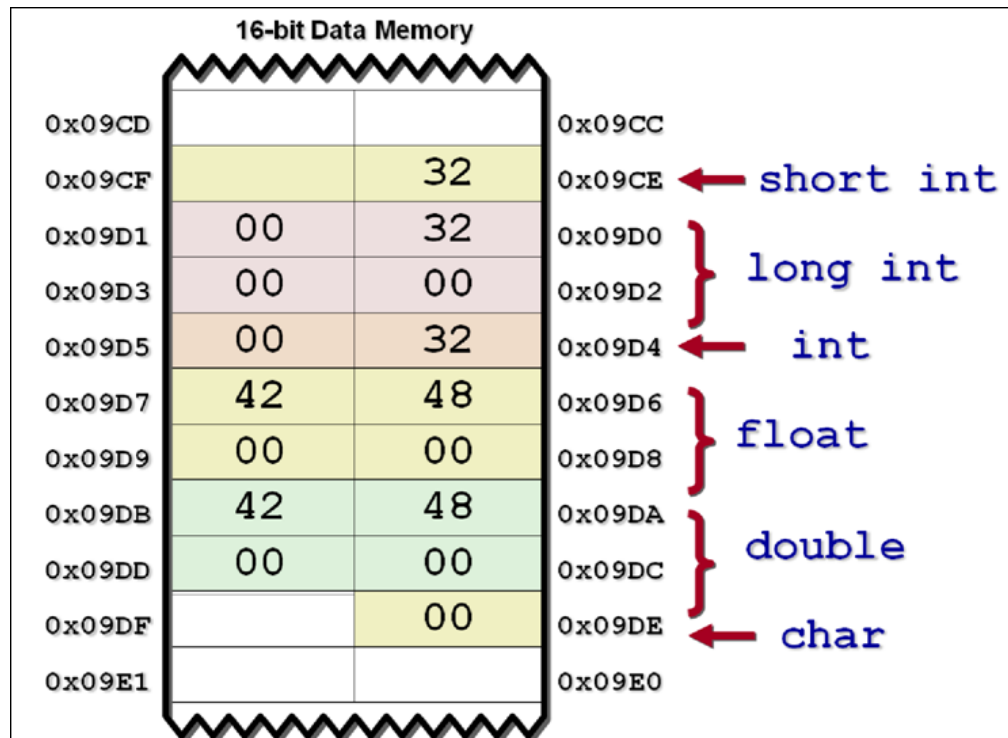


Los tipos de datos fundamentales pueden ser **modificados** por los prefijos **signed**, **unsigned**, **short** y **long**. Estas modificaciones cambian el rango de valores que se pueden representar mediante el tipo de datos fundamentales.

Qualified Type	Min	Max	Size (Bits)
<code>unsigned char</code>	0	255	8
<code>char</code> , <code>signed char</code>	-128	127	8
<code>unsigned short int</code>	0	65535	16
<code>short int</code> , <code>signed short int</code>	-32768	32767	16
<code>unsigned int</code>	0	65535	16
<code>int</code> , <code>signed int</code>	-32768	32767	16
<code>unsigned long int</code>	0	$2^{32}-1$	32
<code>long int</code> , <code>signed long int</code>	-2^{31}	$2^{31}-1$	32
<code>unsigned long long int</code>	0	$2^{64}-1$	64
<code>long long int</code> , <code>signed long long int</code>	-2^{63}	$2^{63}-1$	64



Representación visual de la longitud de **tipos de datos** en la memoria de **RAM de 16 bits** de un microcontrolador de la familia **PIC24**.



#include <stdint.h>

El archivo de cabecera de la biblioteca estándar de C **stdint.h** se utiliza para definir los tamaños de tipos enteros.

```
1  typedef signed char    int8_t;
2  typedef unsigned char  uint8_t;
3
4  typedef short          int16_t;
5  typedef unsigned short uint16_t;
6
7  typedef long           int32_t;
8  typedef unsigned long  uint32_t;
9
10 typedef long long      int64_t;
11 typedef unsigned long long uint64_t;
```



Los **archivos de encabezado** están asociados con un programa a través de la directiva **#include**.

Hay cuatro formas de utilizar esta directiva:

- 1) El archivo de encabezado está en la ruta del compilador

```
{...} #include <filename.h>
```

- 2) El archivo de encabezado está en el directorio del proyecto

```
{...} #include "filename.h"
```

- 3) El archivo de encabezado está en el subdirectorio del proyecto

```
{...} #include "subdirectory_name/filename.h"
```

- 4) El archivo de encabezado está en una ubicación específica fuera del directorio del proyecto


```
{...} #include "C:\path\to\filename.h"
```



Las **constantes son etiquetas (nombres)** que representan valores fijos que nunca cambian durante el curso de un programa.


Hay dos formas de crear constantes en C:

- 1) **Etiquetas de sustitución de texto:** Este tipo de constante no necesita residir en la memoria del microcontrolador de la misma manera que lo hace una variable. En cambio, es algo que se puede manejar en tiempo de compilación como una sustitución de texto



```
1 #define PI 3.14159
```

- 1) **Variables constantes:** Cuando se define la variable y su tipo es modificado por la palabra reservada **const** hará que su valor sea inalterable.



```
const type identifier = value;
```



Sintaxis

```
{...} printf ( " ControlString " , arg1 , ... , argN ) ;
```

Puntos clave:

- Imprime *ControlString* en salida estándar (el terminal en una PC, típicamente un UART en microcontroladores)
- Todos los parámetros separados por comas son opcionales excepto *ControlString*
- Los parámetros del argumento (*arg1 ... argN*) pueden ser variables u otros datos que se incrustarán dentro de *ControlString*
- La funciones **printf()** requiere una gran cantidad de memoria. Es mejor usarlo solo para depurar a menos que necesite toda su funcionalidad.



La función **printf()**:

ControlString

```
printf("a = %d\nb = %d\n", a, b);
```

1st Line 2nd Line arg1 arg2

New Line New Line

El código anterior produciría el siguiente resultado:

```
> a = 5
  b = 10
  -
```

Especificadores de formato:

Format Specifier	Meaning
%c	Single character
%s	String (all characters until '\0')
%d	Signed decimal integer
%o	Unsigned octal integer
%u	Unsigned decimal integer
%x	Unsigned hexadecimal integer with lower case digits (e.g. 1a5e)
%X	Same as %x but with upper case digits (e.g. 1A5E)
%f	Signed decimal value (floating point)
%e	Signed decimal value with exponent (e.g. 1.26e-5)
%E	Same as %e but uses upper case E for exponent (e.g. 1.26E-5)
%g	Same as %e or %f, depending on size and precision of value
%G	Same as %g but will use capital E for exponent



Operador	Operación	Ejemplo	Resultado
*	Multiplicación	$x * y$	Producto de x y y
/	División	x / y	Cociente de x y y
%	Modulo	$x \% y$	Resto de x dividido por y
+	Adición	$x + y$	Suma de x e y
-	Sustracción	$x - y$	Diferencia de x y y
+ (unario)	Positivo	$+ x$	Valor de x
- (unario)	Negativo	$-x$	Valor negativo de x



Los dos últimos operadores aritméticos generales son el operador de **incremento** y **decremento**.

Ambos son operadores unarios, por lo que solo se requiere un operando y se pueden colocar en cualquier lado del operando

Operador	Operación	Ejemplo	Resultado
++	Incremento	<code>x ++</code> <code>++ x</code>	Use <code>x</code> y luego incremente <code>x</code> en 1. Incremente <code>x</code> en 1, luego use <code>x</code> .
-	Decremento	<code>x --</code> <code>--x</code>	Usa <code>x</code> , luego disminuye <code>x</code> en 1. Disminuye <code>x</code> en 1, luego usa <code>x</code> .



El operador de asignación simple "=" asigna el valor de su derecha a la variable de su izquierda.

variable = expresión;

Operador	Operación	Ejemplo	Resultado
=	Asignación	x = y	Asignar x el valor de y



Operador	Operación	Ejemplo	Resultado
<code>+=</code>	Asignación compuesta	<code>x += y</code>	<code>x = x + y</code>
<code>-=</code>	Asignación compuesta	<code>x -= y</code>	<code>x = x - y</code>
<code>*=</code>	Asignación compuesta	<code>x *= y</code>	<code>x = x * y</code>
<code>/=</code>	Asignación compuesta	<code>x /= y</code>	<code>x = x / y</code>
<code>%=</code>	Asignación compuesta	<code>x%= y</code>	<code>x = x% y</code>
<code>&=</code>	Asignación compuesta	<code>x &= y</code>	<code>x = x & y</code>
<code>^=</code>	Asignación compuesta	<code>x ^= y</code>	<code>x = x ^ y</code>
<code> =</code>	Asignación compuesta	<code>x = y</code>	<code>x = x y</code>
<code><<=</code>	Asignación compuesta	<code>x <<= y</code>	<code>x = x << y</code>
<code>>>=</code>	Asignación compuesta	<code>x >>= y</code>	<code>x = x >> y</code>



Los **operadores relacionales** se utilizan para comparar dos valores y decirle si la comparación que se realiza es verdadera o falsa.

Operador	Operación	Ejemplo	Resultado (FALSO = 0, VERDADERO ≠ 0)
<	Menos que	<code>x < y</code>	1 si <code>x</code> es menor que <code>y</code> , de lo contrario 0
<=	Menor o igual a	<code>x <= y</code>	1 si <code>x</code> es menor o igual que <code>y</code> , de lo contrario 0
>	Mas grande que	<code>x > y</code>	1 si <code>x</code> es mayor que <code>y</code> , de lo contrario 0
> =	Mayor o igual a	<code>x > = y</code>	1 si <code>x</code> es mayor o igual que <code>y</code> , de lo contrario 0
==	Igual a	<code>x == y</code>	1 si <code>x</code> es igual <code>ay</code> , de lo contrario 0
!=	No igual a	<code>x! = y</code>	1 si <code>x</code> no es igual <code>ay</code> , de lo contrario 0



En las expresiones condicionales, cualquier valor distinto de cero se interpreta como VERDADERO. Un valor de 0 siempre es FALSO.



Uno de los errores más grandes que cometen los nuevos programadores de C es usar **=** cuando quieren decir **==**. Es muy importante comprender que se trata de dos operadores completamente diferentes.

- "=" se utiliza para asignar un valor a una variable.
- "==" se utiliza para comparar dos valores de equivalencia



Tenga cuidado de no confundir = y ==. ¡No son intercambiables!

Ejemplo

```
1 void main(void)
2 {
3     int x = 2;           //Inici
4     if (x = 5)           //If x
5     {
6         printf("Hi!");   //..disp
7     }
8 }
```



Los operadores lógicos se utilizan normalmente en la toma de decisiones, como en una declaración **if**. Se pueden utilizar para ejecutar código de forma selectiva según el resultado de la condición.

Operador	Operación	Ejemplo	Resultado (FALSO = 0, VERDADERO ≠ 0)
&&	Y lógico	<code>x && y</code>	1 si tanto <code>x ≠ 0</code> como <code>y ≠ 0</code> , de lo contrario 0
	OR lógico	<code>x y</code>	0 si tanto <code>x = 0</code> como <code>y = 0</code> , de lo contrario 1
!	NO lógico	<code>!X</code>	1 si <code>x = 0</code> , de lo contrario 0



En las expresiones condicionales, cualquier valor distinto de cero se interpreta como VERDADERO. Un valor de 0 siempre es FALSO.



Los **operadores bit a bit** realizan álgebra booleana. Cada uno de estos operadores realiza sus operaciones en cada bit de los operandos.

Operador	Operación	Ejemplo	Resultado (para cada posición de bit)
&	Y bit a bit	$x \& y$	1, si 1 tanto en x y y 0, si 0 en x o y o ambos
	O bit a bit	$x y$	1, si 1 en x o y o ambos 0, si 0 tanto en x y y
^	XOR bit a bit	$x \wedge y$	1, si 1 en x o y pero no ambos 0, si 0 o 1, tanto en x y y
~	Bitwise NOT (complemento de uno)	$\sim x$	1, si 0 en x 0, si 1 en x



Los **operadores de desplazamiento** mueven los bits de un operando hacia la izquierda o hacia la derecha el número especificado de bits.

Operador	Operación	Ejemplo	Resultado
«	Desplazar a la izquierda	<code>x << y</code>	Desplazar <code>x</code> por <code>y</code> bits a la izquierda
»	Desplazar a la derecha	<code>x >> y</code>	Desplaza <code>x</code> por <code>y</code> bits a la derecha



Los operadores de direccionamiento de memoria:

Operador	Operación	Ejemplo	Resultado
&	Dirección de	&X	Puntero a x
*	Indirección	*pag	El objeto o función al que apunta p
[]	Suscripción	x [y]	El y- ésimo elemento de la matriz x
.	Miembro de la estructura / unión	xy	El miembro llamado y en la estructura o unión x
->	Miembro de la estructura / unión por referencia	p-> y	El miembro denominado y en la estructura o unión a la que apunta p



Operator	Operation	Example	Result
<code>()</code>	Function Call	<code>foo (x)</code>	Passes control to the function with the specified arguments
<code>sizeof</code>	Sizeof an object or type in bytes	<code>sizeof x</code>	The number of bytes <code>x</code> occupies in memory
<code>(type)</code>	Explicit type cast	<code>(short) x</code>	Converts the value of <code>x</code> to the specified type
<code>?:</code>	Conditional expression	<code>x ? y : z</code>	The value of <code>y</code> if <code>x</code> is true, else the value of <code>z</code>
<code>,</code>	Sequential evaluation	<code>x, y</code>	Evaluates <code>x</code> then <code>y</code> , else result is value of <code>y</code>



```
#include <stdio.h>

#define PI 3.14159

int main(void)
{
    float radius, area;

    //Calculate area of circle
    radius = 12.0;
    area = PI * radius * radius;
    printf("Area = %f", area);
}
```



ANSI C

Lenguaje C



MMJ

Smart Electronics

GRACIAS

