Embedded Systems CSEN701

Dr. Catherine Elias

Eng. Abdalla Mohamed

Office: C1.211
Mail : abdalla.abdalla@guc.edu.eg

Eng. Mohamed Elshafie

Office: C1.211
Mail : Mohamed.el-shafei@guc.edu.eg

Eng. Maysarah El Tamalawy

Office: C7.201
Mail : Maysarah.mohamed@guc.edu.eg

Outline :

- Recap.
- Memory Architecture.
- C programing.
- Memory types.

Outline :

- **Recap.**
- Memory Architecture.
- C programing.
- Memory types.

What is Embedded Systems?

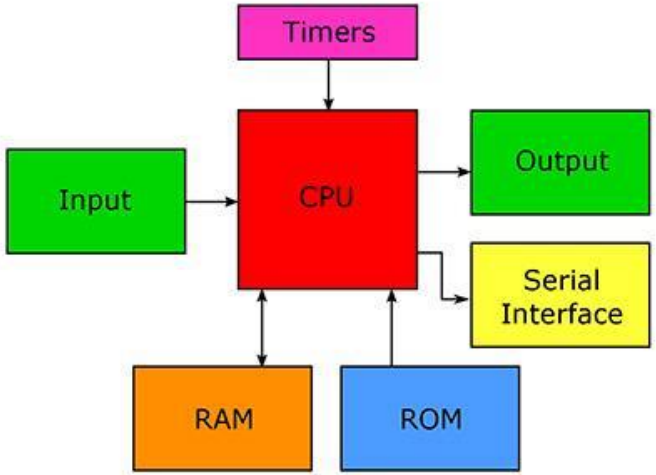
- ES is a microprocessor-based system that is built to control and perform a dedicated Function (special-purpose-system) or range of functions and is not designed to be programmed by the end user, Unlike the PC which is considered as a programmable general-purpose-computer .

Embedded Systems characteristics.

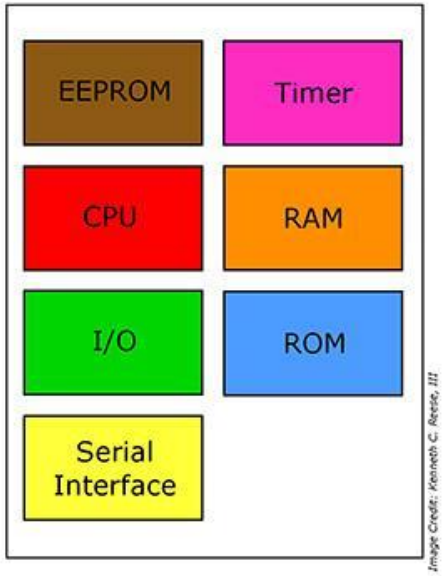
- Heterogeneous System, Reactive and Efficient.
- Networked, Maintainable, Reliable and Safe.

Microprocessor vs Microcontroller

Microprocesser: CPU
and several supporting chips.



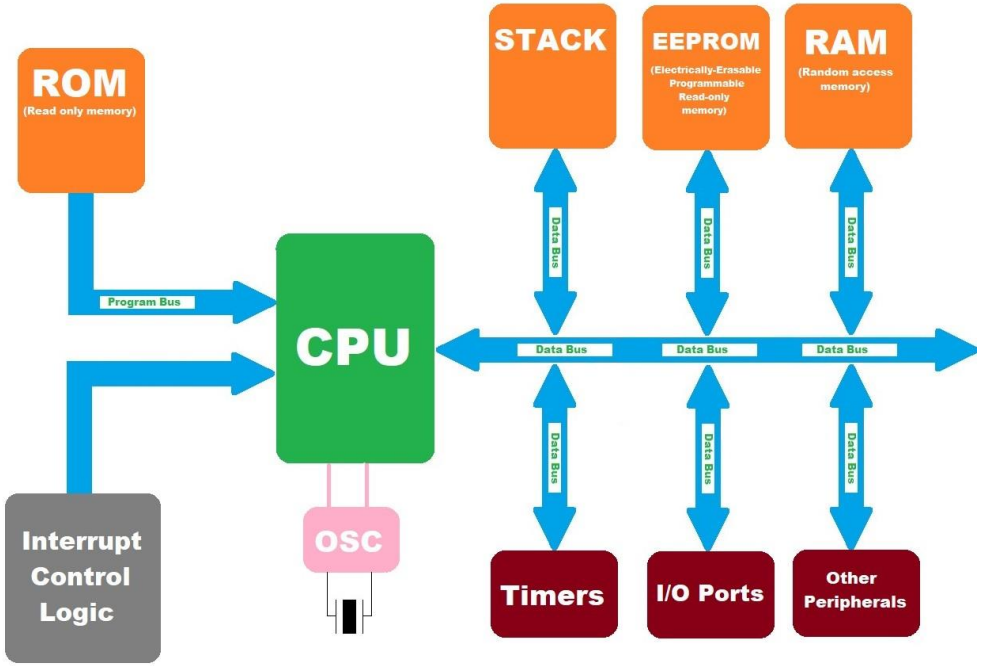
Microcontroller: CPU
on a single chip.



Outline :

- Recap.
- **Microcontroller Architecture.**
- Memory types.
- C programing.

Microcontroller Architecture.



Outline :

- Recap.
- Microcontroller Architecture.
- **Memory types.**
- C programing.

Memory Types:

Volatile memory

- SRAM
- DRAM

Loses data when power is off

Non-Volatile memory

- PROM
- EPROM
- EEPROM

Preserves data when power is off

Memory Types:

Volatile memory:

SRAM (Static Random Access Memory):

- SRAM is faster than other types of RAM due to simplified design.
- SRAM consumes less power than DRAM.
- SRAM are Larger than DRAM so it takes more space and it is more expensive.

DRAM (Dynamic Random Access Memory):

- DRAM is smaller in size and less complex than SRAM.
- DRAM is less expensive and cost efficient.
- DRAM has high latency and consumes more power than SRAM.

Memory Types:

Non-Volatile memory:

- **PROM** (Programmable Read Only Memory): This type of ROM can be programmed once.
- **EPROM** (Erasable PROM): This type can be erased and reprogrammed through UV rays.
- **EEPROM** (Electrically Erasable PROM): This type can be erased electrically and reprogrammed.
- **Flash Memory**: It is an EEPROM with Larger size.

Memory Architecture.

- Memory consists of many smaller blocks called Registers each Register contains data.
- Each register has a specific address which we can access its data through it.

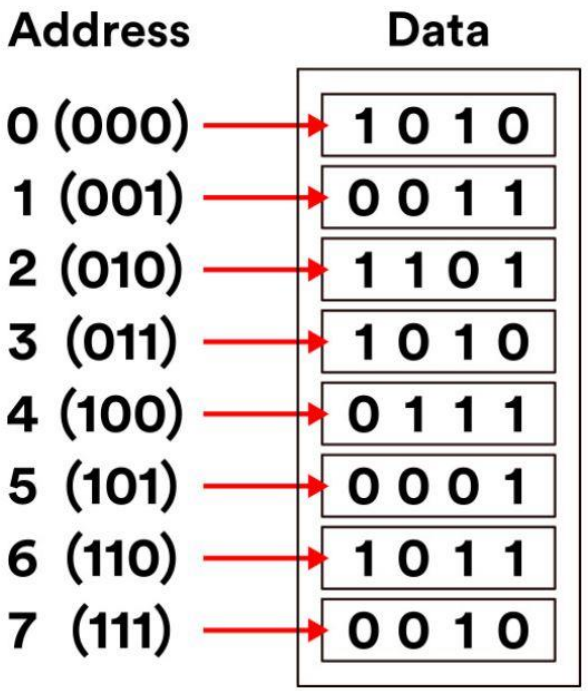


Figure 1. Basic Memory: Addressing an array of 8 × 4-bit registers

Outline :

- Recap.
- Microcontroller Architecture.
- Memory types**.**
- **C programing.**

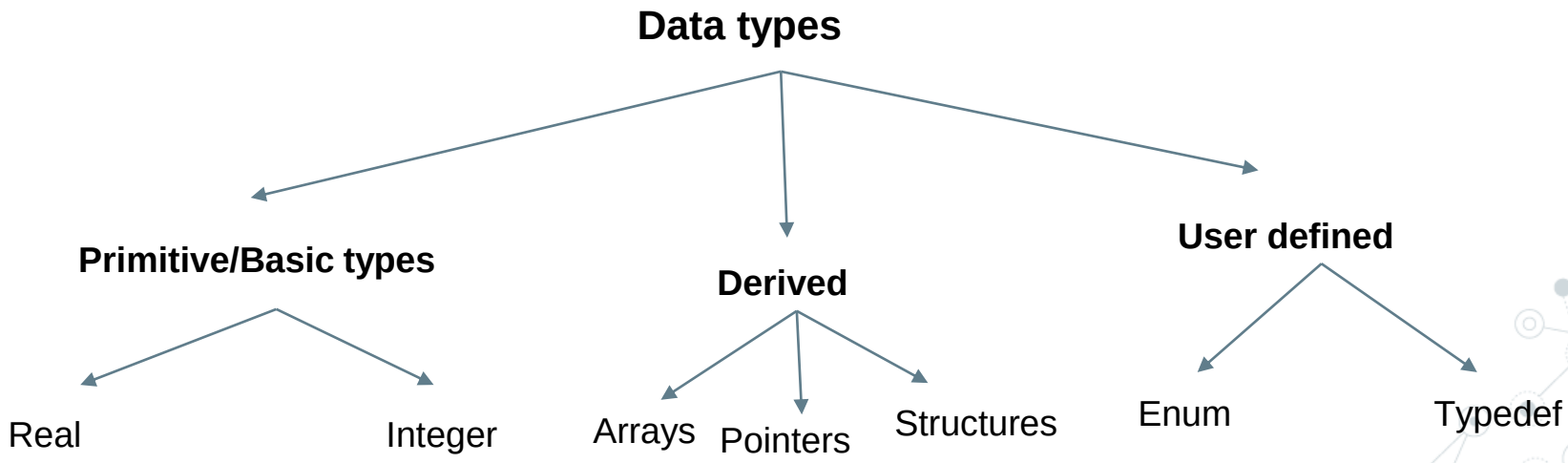
C Programming:

Why C Programming ?

- C is a low-level language that provides direct access to hardware resources, allowing for efficient and fast code execution. This is crucial in embedded systems where performance is often a critical factor.

- C have Pointers which allow direct manipulation of memory, which is crucial for tasks like dynamic memory allocation.

C Programming:



C Programming:

Primitive data types.

Real data types

- **float**: Usually 4 bytes (32 bits).
- **double**: Typically 8 bytes (64 bits).
- **long double**: 16 bytes.

Integer data types

- **Char:** 1 byte.
- **short :** 2 bytes.
- **Int :**4 bytes.
- **Long :**8 bytes

C Programming:

```c
#include <stdio.h>

int main() {
    int intVariable =10;
    char charVariable="A";
    float floatVariable=3.14;
    double doubleVariable=10000;

    printf("Size of int: %d bytes\n", sizeof(intVariable));
    printf("Size of char: %d bytes\n", sizeof(charVariable));
    printf("Size of float: %d bytes\n", sizeof(floatVariable));
    printf("Size of double: %d bytes\n", sizeof(doubleVariable));

    return 0;
}
```

```
Size of int: 4 bytes
Size of char: 1 bytes
Size of float: 4 bytes
Size of double: 8 bytes
```

C Programming:

How to define Array in C:

```c
int main()
{
    // Declaration and initialization of an array of integers
    int intArray[5] = {10, 20, 30, 40, 50};
    // Declaration and initialization of an array of chars or define a string
    char String[8]="CSEN701";
    // Accessing and printing individual elements
    for (int i = 0; i < 5; i++)
    {
        printf("intArray[%d]: %d\n", i, intArray[i]);
    }

    return 0;
}
```

- There is no string in c language as data type so we can make array of chars as a String

NB Length of the string in C is equal to normal length+1.

C Programming:

Struct:A `struct` in C is a user-defined data type that allows you to group together variables under a single name to represent a concept or entity.

```c
// Define the struct
struct sperson {
    char name[50];
    int age;
    float height;
};
```

```c
int main() {
    // Declare a variable of type sperson
    struct sperson person1;

    // Assign values to the members of the struct
    char name[100]="Maysarah";
    strcpy(person1.name, name);
    person1.age = 23;
    person1.height = 1.73;

    // Print out the information
    printf("Name: %s\n", person1.name);
    printf("Age: %d\n", person1.age);
    printf("Height: %.2f meters\n", person1.height);

    return 0;
}
```

C Programming:

A pointer in C: is a variable that holds the memory address of another variable, allowing direct access and manipulation of that variable's data.

```c
#include <stdio.h>

int main() {
    int num = 10;       // Declare an integer variable
    int *ptr;           // Declare a pointer variable

    ptr = &num;         // Assign the address of 'num' to the pointer 'ptr'

    printf("Value of num: %d\n", num);
    printf("Address of num: %p\n", &num);
    printf("Value stored at pointer: %d\n", *ptr);
    printf("Address stored in pointer: %p\n", ptr);

    return 0;
}
```

```
D:\Teaching_C>Pointers
Value of num: 10
Address of num: 0061FF18
Value stored at pointer: 10
Address stored in pointer: 0061FF18
```

C Programming:

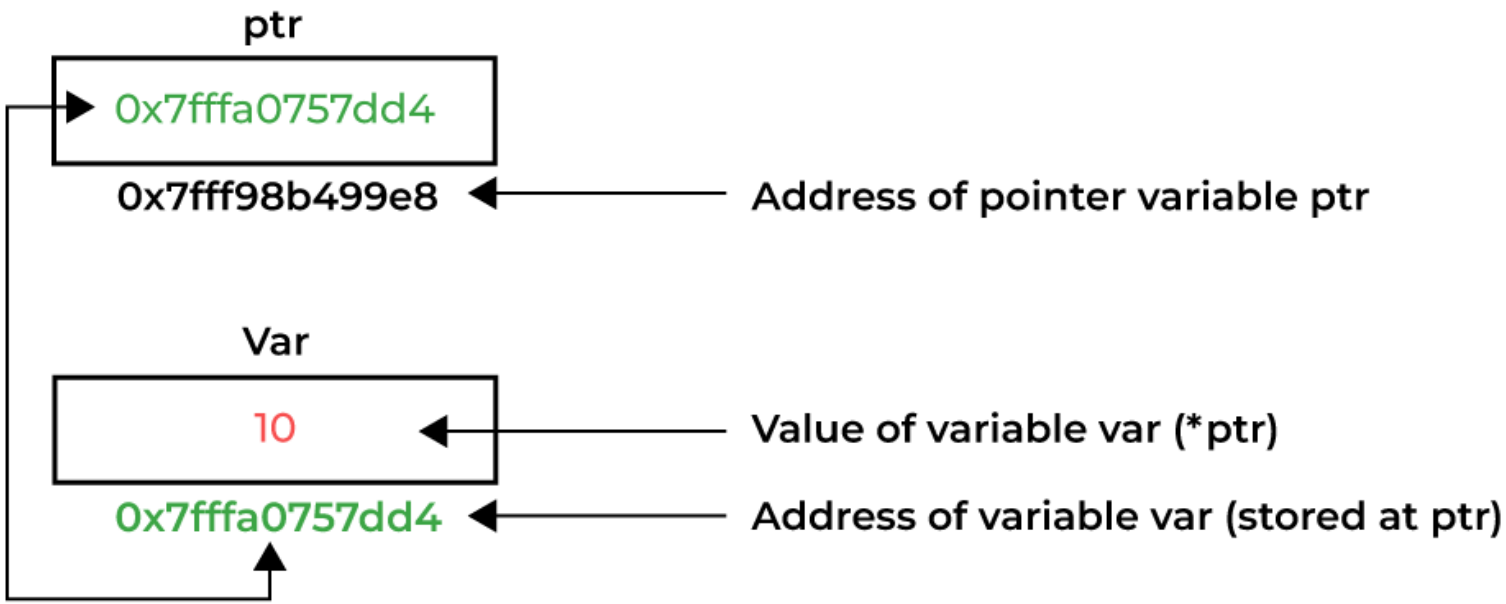
- **Pointer Declaration and Initialization:** Pointers must be declared with a matching data type and initialized with the address of a compatible variable.
- **Dereferencing a Pointer (*):** The * operator retrieves the value stored at the memory address pointed to by a pointer.
- **Getting the Address (&):** The & operator obtains the memory address of a variable.

C Programming:

Types of Pointers:

- **Null Pointer:** A pointer that doesn't point to any valid memory location.
- **Void Pointer:** A pointer without a specific type, used for generic operations.
- **Function Pointer:** A pointer that points to a function, allowing dynamic function calls.
- **Pointer to Pointer:** A pointer that holds the address of another pointer.
- **Array Pointer:** A pointer that can be used to navigate an array.
- **Wild Pointer:** An uninitialized pointer, pointing to a random memory location.

Pointers in C.



Pass By Value VS Pass By address

```c
void increment( int a){
    a = a+5 ; // a new Local variable is created
    printf( " the value of 'a' inside increment  is %d   " , a)  ;
    printf( " the address of 'a' is %p \n " , &a) ;
} // local variable value is lost once we get out of the function
int main( void) {
    int x  = 5  ;
    printf( " the value of 'x' before increment  is %d  " , x)  ;
    printf( " the address of 'x' is %p \n " , &x) ;
    increment(x) ;
    printf( " the value of 'x' after increment  is %d  " , x) ;
// X is passed by value . the value of x is copied to a local variable a
    // real value of x in not altered !!!!
    }
```

Console

```
<terminated> (exit value: 0) tut_2_CSEN701.exe [C/C++ Application] C:\Users\Abdalla\eclipse-workspace\tut_2_CSEN701\Debug\tut_2_CSEN701.exe (10/5/23
the value of 'x' before increment  is 5   the address of 'x' is 00000060383ffc0c
 the value of 'a' inside increment  is 10    the address of 'a' is 00000060383ffbe0
 the value of 'x' after increment  is 5
```

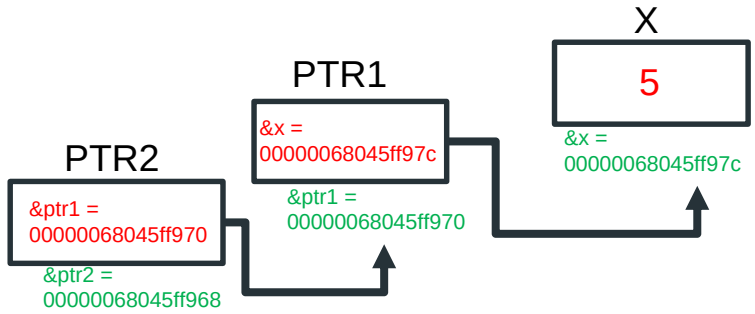
```c
void increment( int * a){
    *(a) = *(a)+5 ; // pointer to integer is introduced to carry the address of x
    printf( " the value of 'a' inside increment  is %d   " , *a)  ;
    printf( " the address of 'a' is %p \n " , a) ;
}
int main( void) {
    int x  = 5  ;
    printf( " the value of 'x' before increment  is %d  " , x)  ;
    printf( " the address of 'x' is %p \n " , &x) ;
    increment(&x) ;
    printf( " the value of 'x' after increment  is %d  " , x) ;
// X is passed by address . the address of x is sent to the function
    // real value of x is edited !!!!
    }
```

Console

```
erminated> (exit value: 0) tut_2_CSEN701.exe [C/C++ Application] C:\Users\Abdalla\eclipse-workspace\tut_2_CSEN701\Debug\tut_2_CSEN701.exe (10/5/23, 6:11 PM)
the value of 'x' before increment  is 5   the address of 'x' is 0000003f23fff94c
 the value of 'a' inside increment  is 10    the address of 'a' is 0000003f23fff94c
 the value of 'x' after increment  is 10
```

Pointers in C.

```
int main ( void ){
    int x = 5 ;
    int *ptr1= &x ;
    int **ptr2 = &ptr1  ;  Pointer to pointer to integer
printf( " value of X : =  %d \n" , x ) ;
printf( " value of address X = %p \n "  , &x ) ;
printf( " value of the pointer 1  = %p  \n" , ptr1 ) ;
printf( " value of address pointer 1  = %p  \n" , &ptr1 ) ;
printf( " value of Asterisk POINTER 1 = %d \n " , *ptr1 ) ;
printf( " value of ASTERISK address of pointer 1  = %p  \n  " , *&ptr1 ) ;
printf( " value of pointer 2 = %p   \n " , ptr2 ) ;
printf( " value of asterisk pointer 2 = %p   \n " , *ptr2 ) ;
printf( " value of double asterisk pointer 2 = %d   \n " , **ptr2 ) ;
printf( " value of address  pointer 2 = %p   \n " , &ptr2 ) ;
printf( " value of  asterisk address  pointer 2 = %p   \n " , *&ptr2 ) ;
}
```



value of X : = 5
value of address X = 00000068045ff97c
value of the pointer 1 = 00000068045ff97c
value of address pointer 1 = 00000068045ff970
value of Asterisk POINTER 1 = 5
value of ASTERISK address of pointer 1 = 00000068045ff97c
value of pointer 2 = 00000068045ff970
value of asterisk pointer 2 = 00000068045ff97c
value of double asterisk pointer 2 = 5
value of address pointer 2 = 00000068045ff968
value of asterisk address pointer 2 = 00000068045ff970

C Programming:

- **Example for using pointers to arrays:**

```c
int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int *ptr = arr;  // Initialize pointer to point to the start of the array

    // Display the addresses
    printf("Address of arr[0]: %d\n", &arr[0]);
    printf("Address of ptr: %d\n", ptr);

    // Increment the pointer
    ptr++;

    // Display the addresses after incrementing
    printf("Address of arr[1]: %d\n", &arr[1]);
    printf("Address of ptr after incrementing: %d\n", ptr);

    return 0;
}
```

```
D:\Teaching_C>pointers_arrays_ex
Address of arr[0]: 6422280
Address of ptr: 6422280
Address of arr[1]: 6422284
Address of ptr after incrementing: 6422284
```

Why the difference is 4 ?
→ As the pointer when incremented it increases by its size.(int= 4 bytes).

C Programming:

Problem1:

- Reverse Array: Write a function that accepts an array and its size, and reverses the elements in-place.

```c
void reverseArray(int *arr, int size) {
    int start = 0;
    int end = size - 1;
    int temp;

    while (start < end) {
        temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}
```

```
D:\Teaching_C>problem1
Original Array: 1 2 3 4 5
Reversed Array: 5 4 3 2 1
```

```c
int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    printf("Original Array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }

    reverseArray(arr, size);

    printf("\nReversed Array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Check the following links for more practice :

https://www.youtube.com/watch?v=AWliApDc61w&list=PL2_aWCzGMAwLSqGsERZGXGkA5AfMhcknE&index=1

*https://www.youtube.com/playlist?list=PL2_aWCzGMAwLZp6LMUKI3cc7pgGsasm2_*

THANK YOU