Macros

By: Yehia M. Abu Eita

Outlines

- Introduction
- Object-like macros
- Function-like macros
- Famous macros
- Macros Vs. functions
- Macros Vs. typedefs

Introduction

- Macros are **values represented by a text**, this value could be a number, or a block of code.
- Macros are defined using the “`#define`”, `#define PI 3.14`.
- It is processed by a text replacement tool called **preprocessor**.
- It is replaced **during compilation** time not in the execution time.
- Macros can be written in **multiple lines** using ‘****’ as a line termination.
- Macros increases your code readability.

Object-like macros

- These macros define text for constants.
- Example:
  - `#define PI 3.14`
  - `#define RADIUS 5`
  - `circleArea = PI * RADIUS * RADIUS; // replaced with circleArea = 3.14 * 5 * 5;`

Function-like macros

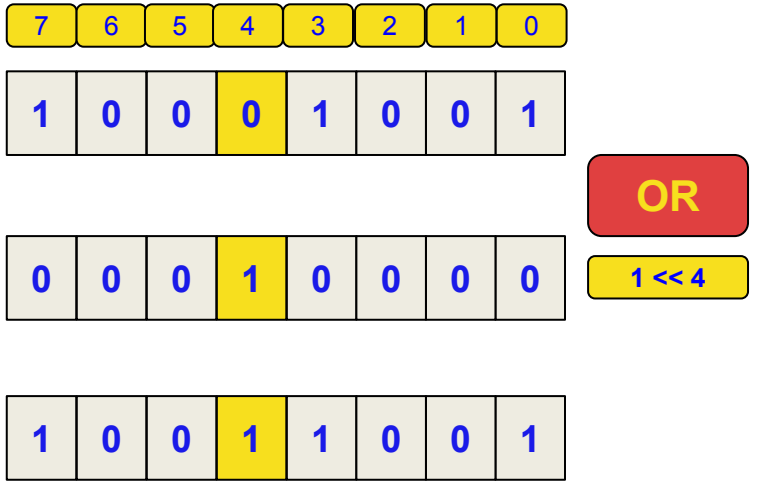
- These macros define text for small functions.
- Example:
  - `#define PI 3.14`
  - `#define RADIUS 5`
  - `#define AREA(R) PI * R * R // Function-like macro`
  - `circleArea = AREA(5); // replaced with circleArea = 3.14 * 5 * 5;`
- Function-like macros may cause some wrong results:
  - `#define SQUARE(X) X * X`
  - `x = SQUARE(2+3); // replaced with x = 2+3*2+3;` **x will be 11 not 25**
  - **Solution:** `#define SQUARE(X) (X) * (X)` **- > replaced with** `x = (2+3)*(2+3);`

Famous macros

- There are famous macros widely used in embedded systems.
- Examples:
  - `#define MAX(X,Y) ((X)>(Y))?(X):(Y) // Macro to find Maximum of two numbers`
  - `#define MIN(X,Y) ((X)<(Y))?(X):(Y) // Macro to find Minimum of two numbers`
  - `#define SET_BIT(X,BIT_NO) X|=(1<<BIT_NO) // Set bit Macro`
  - `#define CLR_BIT(X,BIT_NO) X&=~(1<<BIT_NO) // Clear bit Macro`
  - `#define READ_BIT(X,BIT_NO) ((X & (1<<BIT_NO))>>BIT_NO) // Read bit Macro`
  - `#define TOGGLE_BIT(X,BIT_NO) X^=(1<<BIT_NO) //Toggle bit Macro`

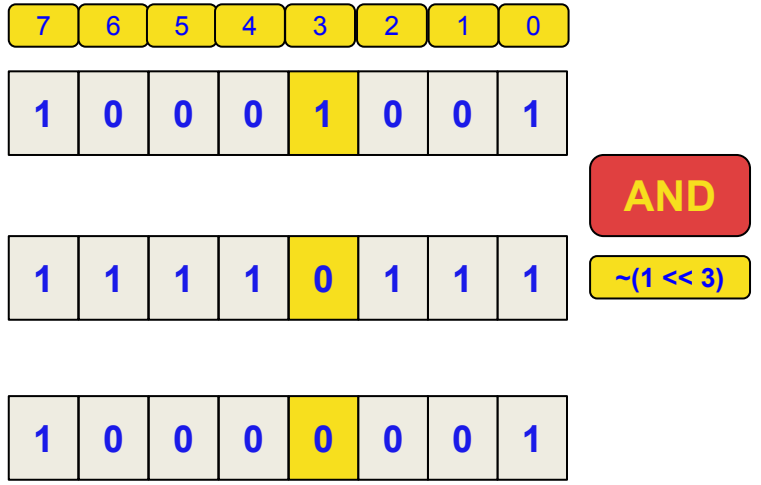
Famous macros

- **Set bit macro:**
  - `#define SET_BIT(X,BIT_NO) X|=(1<<BIT_NO)`
  - `SET_BIT(X,4);`
  - **Will be replaced with** `X|=(1<<4);`
  - `X = X | (1<<4);`
  - **X can be a hardware register in the microcontroller.**



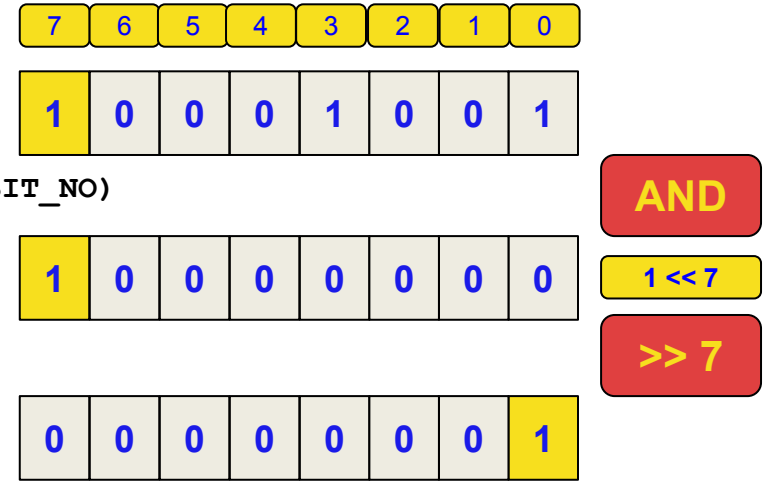
Famous macros

- **Clear bit macro:**
  - `#define CLR_BIT(X,BIT_NO) X&=~(1<<BIT_NO)`
  - `CLR_BIT(X,3);`
  - **Will be replaced with** `X&=~(1<<3);`
  - `X = X & ~(1<<3);`
  - **X can be a hardware register in the microcontroller.**



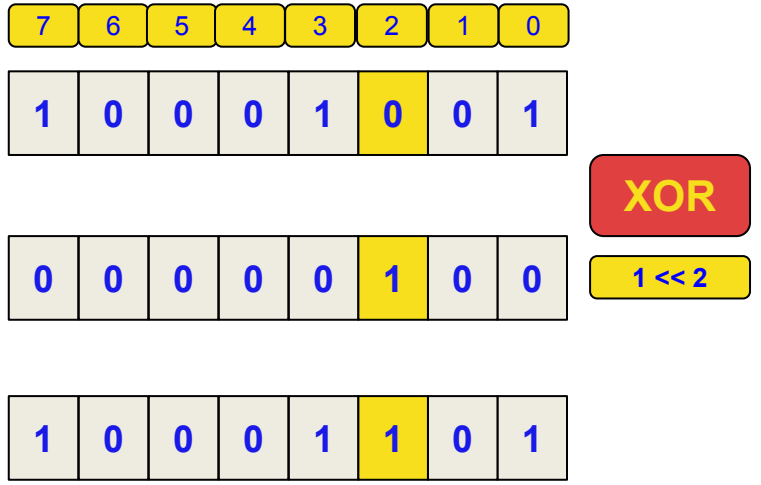
Famous macros

- **Read bit macro:**
  - `#define READ_BIT(X,BIT_NO) ((X&(1<<BIT_NO))>>BIT_NO)`
  - `READ_BIT(X,7);`
  - **Will be replaced with** `((X&(1<<7))>>7);`
  - This will **result** in **1 if the bit is 1**.
  - This will **result** in **0 if the bit is 0**.
  - **X can be a hardware register in the microcontroller.**



Famous macros

- **Toggle bit macro:**
  - `#define TOGGLE_BIT(X,BIT_NO) X^=(1<<BIT_NO)`
  - `TOGGLE_BIT(X,2);`
  - **Will be replaced with** `X^=(1<<2);`
  - `X = X ^ (1<<2);`
  - **X can be a hardware register in the microcontroller.**



Macros Vs. functions

- **Function-like macros** have **advantages** over regular and inline functions:
  - **Reduced execution time**, no context switch time exists.
  - **Less data memory allocation**, only replaced text.
- **Regular and Inline functions** have **advantages** over function-like macros:
  - **Easy error detection and traceability**.
  - **Reduced lines of code**, no code replacement, **reduces program memory size.**
  - **More control on input argument and return types**.
  - **Functions can return values**.

Macros Vs. typedefs

- Defining type using typedef:
  - `typedef unsigned int* ptoint;`
  - `ptoint x, y; // x and y are pointers to integers`
- Defining type using macros:
  - `#define PTOINT unsigned int*`
  - `PTOINT x, y; // Only x is a pointer to integer, but y is an integer`
  - Above is replaced with `unsigned int* x, y;`

Summary

- Now you are familiar with macros, define, use, and types.
- Remember, put function-like macros arguments between (and) to avoid faulty results.
- Remember, Macros are replaced during the compilation time.
- Using set, clear, read, and toggle macros in you embedded systems applications is useful.