Knowledge of baseline

Light Up LED : Instruction Macro - Assembly

What we want to achieve

Let's take a look at a small, but significant change to the source described in the first tutorial.

We have said that one of the essential actions that programming languages allow is to replace absolute values with symbols or "labels".

We have seen how this allows you to create readable and easily handled sources, as well as constituting an element of first abstraction from the purely hardware level.

By "abstraction layer" we mean here a mode that allows you to write a source that is as independent as possible from the arrangement of the resources in the memory map and from the characteristics of the registers that manage the peripherals. This, as we have seen, allows you to use a label instead of an absolute value, a value that will be assigned by the compiler based on the resources made available with the inclusion of ***.inc files*** or similar.

This necessity becomes indispensable when writing relocatable programs and is particularly advanced when making use of high-level languages, equipped with libraries, functions, etc.

In the first exercise we saw how to pass, in the definition of the output pin for the LED control, from an absolute address "port, pin" in hexadecimal to a pair of equivalent labels.

In our case, the **GPI0** and **GP5** labels, which we have seen to be determined by the ***p12F519.inc*** file, are "pre-packaged" labels reserved by the Assembler. On the other hand, we create the LED label from scratch with the **#define directive**, where **GPI0,GP5** matches the address of the bit in that particular port. From the moment of definition onwards, for that source, instead of writing:

```
Bsf 6,5                 ; turn on the LED
```

On the basis of the previous equivalence I will be able to write:

```
Bsf GPI0,GP5            ; turn on the LED
```

Let's see how now the line of instruction takes on a different aspect: the action becomes clearer, having made use of labels that "remember" what you are referring to.

However, it should also be obvious that the abstraction of labels somehow hides the underlying hardware reality, making it transparent to the user; and this is a risk that is run the higher the degree of abstraction, for example with a C language, where complex initialization actions of registers and ports are performed with a single command, making the source gain in simplicity, but making the user lose the knowledge of the

complexities that condense and thus prevent the use of peripherals and modules outside of the available functions and libraries.

In Assembly, based only on the native opcodes of the processor, the use of functions and libraries is also possible, but in general it is necessary to create them and this action, if on the one hand can be heavy work, on the other hand keeps the programmer always in contact with the hardware.

A first step of source "simplification" is achieved with a more intensive use of symbols.

In the diagram, we have connected an LED to the GP5 pin, which corresponds to bit 5 for GPIO register.

#define

We can transform this situation into a symbolic form with a label that summarizes the various components, creating an ad hoc definition. This is still achieved with the directive of the Assembler **#define**.

 The directive **#define** requires the compiler to replace the declared label with text, which can be an absolute value or another label. This allows you to use only symbolic values in the list, substituting the absolutes and, at the same time, allows a greater readability of the source as the assigned labels will preferably have some logical (mnemonic) connection with the absolute represented.

The syntax is simple:

#define	sp	<label>	sp	[object]	sp	[; comment]

It should be noted that the **#define** Directive, unlike the other Directives, has to **start in the first column**. At least one space separates the various components of the row.

It is possible to omit the comment, but also the string, when you want to define only the label. The **label** is the name that we want to define and that we can link to a string. If we don't declare any strings, the label is still defined, even if it doesn't match a specific value. The Assembler will use this definition when compiling.
Then the line:

```
#define    New_name
```

It defines the **label New_name**, in fact it just occupies that specific name and inserts it in the list of labels. In practice, we can also define a label without ever using it in the program or using it for purposes other than linking to a real address.
On the other hand:

```
#define    driver    PORTB,5
```

establishes the connection between the label **driver** and bit 5 of **PORTB** .

If, for some reason in the logic of the program, a definition needs to be cancelled, it will be used

the directive **#undefine**, followed by the label that you want to remove from the list of defined labels.

Obviously, the labels we create can be named as you prefer: so we can define **LED** or **LED_Rosso** or **LED1** or even **Gustavo**, but it is appropriate that the names assigned are in some way "mnemonics" that remind the reader of their function.

It should now be clearer the concept of "label" and the importance that symbols have in writing a good source. We'll see this even better in the various tutorials.

Using Label

The use of labels makes the program more readable and more portable, and allows you to make changes in a matter of moments.

We then define the word "LED" as the label to which the compiler will replace the command bit of the LED.

```
;*****************************************************************
;                            DEFINITIONS
;
;LED Control Output
;-------------------
#define    LED    GPIO,GP5              ; Assign Label
```

From now on, the compiler will replace **it with** GPI0,GP5 every time it encounters the **LED label**, which in turn will be filled in as bit 5 of register 6.

The "LED" is now linked through symbols to its actual hardware situation. So,

having to turn on the LED, the line:

```
      Bsf GPIO,GP5            ; turn on the LED    (1)
```

It will be replaceable with the equivalent:

```
      Bsf LED                 ; turn on the LED    (2)
```

The build process will look like this:

1. the compiler meets the **bsf LED** line, where **LED** is the object
2. according to the **defined** imposed, it replaces the label with **GPIO,GP5**
3. **GPIO,GP5** are detected as defined by the file **processorname.inc** and replaced by the corresponding absolute values
4. the compiler renders the correct binary code in the **.hex** executable to bring the bit connected to the LED to 1

It should be noted that, once the label with the correct GP bit match in the register has been defined **GPIO**, this label can be used whenever you want to refer to this bit.

Obviously, this procedure will be all the more effective the more complex the program will be, allowing a considerable advantage: if we want to move the LED to another GP, for example GP2, if we limit ourselves to line (1) we would have to rewrite it in the entire source in the new **form bsf GPIO,GP2.**
On the other hand, once the definition of the LED label has been adopted and therefore the use of the line (2), it will no longer need to be modified, since it will be sufficient to adjust the definition **#define the GPI0,GP2 LEDs only once**

In fact, in the proposed template we make use of a semi-graphic stroke to immediately visualize the correlation between GPIO bits and function.

```
;*****************************************************************
;             DEFINITION OF PORT IN USE
;
;GPIO map
;|  5  |  4  |  3  |  2  |  1  |  0  |
;|-----|-----|-----|-----|-----|-----|
;| LED |     |     |     |     |     |
;
;#define    GPIO,GP0     ;
;#define    GPIO,GP1     ;
;#define    GPIO,GP2     ;
;#define    GPIO,GP3     ;
;#define    GPIO,GP4     ;
#define    LED  GPIO,GP5     ; LED between pin
```

It also gives us a quick basis for definitions: by decommenting, i.e. removing the **;** From the desired line we can transform the line itself from a comment to an active part of the program. The editor with its coloring of the various fields gives us an additional help to understand the line.

MPASM - le label

If you don't have any ideas on the subject, it may be useful to consult the pages related to **the label rules used in MPASM.** Let us add a few words on the subject here**.**

Important to consider is the fact that, the first time a label appears in the Assembly source, it corresponds to its declaration.

If you don't use a specific directive for the declaration, they can be declared directly, simply by writing them starting in the first column. For example,:

```
fiat_lux    Bsf LED         ; turn on the LED
```

The label **fiat_lux** refers to the entire line on which it is written and corresponds as a numeric value to that of the address of the program memory in which the BSF instruction will be placed. Here the label declares itself "by itself" without the use of #define.

 In the case of label duplication or other syntactic errors, the Assembler will generate error messages and will usually **be unable to complete the compilation.**

Macro Instructions

But we can go a step further in the use of methods to improve the structure of the source.
One of these is the use of **MACROs**, short for **macro instruction**.

We have mentioned that **MPASM** is a **Macro-Assembler** , and the creation of macro instructions is the main feature of a Macro-Assembler.

 **A macro instruction, as the name suggests, is a series of instructions, or other source elements, grouped under a symbolic name so that they can be executed by means of a single command.**

Let's see one application in our case. To turn

on the LED we used the line:

```
        bsf   GPIO, GP5     ; LED ON
```

which contains the **BSF statement**.

With a first symbolic approach we defined **LED = GPIO,GP5.**

Now, with a next step, let's transform the whole row into a single symbol, i.e. let's assign it a label. This replaces a line of statement with a single label: it's a macro statement.

```
Accendi_LED  MACRO
             bsf LED        ; LED ON
             ENDM
```

The **MACRO** directive is the necessary key: the place between the command and its **END** closure is considered by the Macro -Assembler compiler as equivalent to the indicated label.
That is, by calling the label **LEDON** it will be as if you have written what is defined in the macro.

MACRO - ENDM

A "Macro" is defined by the directive of the same name and consists of a list of opcodes or other components of the Assembler that are collected under a single label. When, during its action, the compiler encounters the directive, it is replaced with what was indicated in the source when the macro was defined.
The syntax allows you to use multiple lines:

Label	sp	MACRO	sp	nameMacro	sp	[Parameters]	sp	[; comment]
		Instructions Instructions						
		ENDM						

Note that the #define statement for the label is **not used, as seen before:** it defines itself by appearing in the first column, while the **MACRO-ENDM pair establishes the "content" of the label itself.**

The simplest use of a macro might be to name a series of repetitive instructions to avoid having to rewrite them multiple times in the source. During compilation, the macro label will be replaced with this content.
The content of the rows can be the most varied, including other macros as well. In addition, macros can be collected in files to be included with the **#include** directive, in order to form libraries of useful functions that can be called as needed.

Note that macros must be defined in the source before they are called.

Macros can take on complex aspects and also use parameters; We will see some of these possibilities during the next exercises.

In this case, the content is a single line, but it can be much more complex, implying several lines;

The compiler process in our case is as follows:

1. The compiler encounters the label Accendi_LED in the source , which it finds defined as a macro
2. replaces the label with the **bsf LED line**
3. detects that the **LED label** is defined and replaces it with **GPIO,GP5**
4. notes that the **GPIO** and **GP5** labels are defined in the ***.inc file*** and replaces them with the relative values
5. generates the ***.hex*** file with the correct binary code to bring the **GP5 bit to 1**

All this substitution process is completely the responsibility of the compiler; The only obligation left for the user is to have included only labels that have been previously defined. If a label is not defined before its use, or its definition is mistakenly duplicated, the compiler generates an error list and does not proceed to create the ***.hex*** file until the error is corrected.

We can have fun writing other macros as well:

```
Set_Osccal    MACRO
              movwf  OSCCAL
              END

Clear_GPIO    MACRO
              clrf   GPIO
              END

GP5_out       MACRO
              movlw b'11011111' ; PORT Direction Mask
             tris  GPIO         ; To the Management
             END

Blocca_PC    MACRO
             goto  $
             END
```

In this case, the part of the source that performs the operations on the pin will simply boil down to:

```
MAIN:
   Clear_GPIO
   GP5_out
   Accendi_LED
   Blocca_PC
```

It's not C or BASIC, but a macro-intensive assembly: in the compilation, the Macro-Assembler will replace each label with the instructions indicated in the creation of the macros.
By compiling this version, you will be able to see in the **.lst** file the macro expansion performed by MPASM.

In practice, the use of macros is at the discretion of the user, but it must be considered that their purpose is essentially to facilitate the work of the programmer and make the source more agile and readable.
However, it is not the case to exaggerate and it is necessary, depending on the intended use, to evaluate the best choice between macro and subroutine, which we will see in the next exercise.

For more details on macros, you can consult these pages, where they compare two common techniques in writing a program with subroutines, which we will see in the next tutorials.

It should be noted that, just as a label must be defined before it can be used, so must a macro. If we don't define the labels or macros before using them, the compiler will generate an error message.

Given the minimal program, the macro is also minimal; We will see in the course of the exercises a more massive use of more complex macros.

A note

Processor documentation is no longer included at the end of the sources, in order not to weigh them down.

However, these documentation texts are available separately and, if desired, can be included at the end of the source with a copy-and-paste.

We remind you that the inclusion of the text after the `END statement` means that it is present in the source, but does not affect the compilation in the slightest.

12F519 - 1Aw_519.asm

```
;*****************************************************************
;-----------------------------------------------------------------
;
;    Title          :  Assembly & C Course - Tutorial 1Aw_519
;                      Turn on an LED connected to the GP5 pin.
;                      The LED lights up when the power supply
;                      arrives and remains on as long as it is
;                      present

;    PIC            :  12F519
;    Support        :  MPASM
;    Version        :  1.0
;    Date           :  01-05-2013
;    Hardware Ref.  :
;    Author         :  Afg
;
;-----------------------------------------------------------------
;
; Pin use :
;   ---------------------------
;       12F519 @ 8 pin
;
;                    |__\/__|
;              Vdd -|1    8|- Vss
;              GP5 -|2    7|- GP0
;              GP4 -|3    6|- GP1
;         GP3/MCLR -|4    5|- GP2
;                    |______|
;
;    Vdd                 1: ++
;    GP5/OSC1/CLKIN      2: Out LED at Vss (R in series)
;    GP4/OSC2            3:
;    GP3/! MCLR/VPP      4:
;    GP2/T0CKI           5:
;    GP1/ICSPCLK         6:
;    GP0/ICSPDAT         7:
;    Vss                 8: --
;
;*****************************************************************
;=================================================================
;             DEFINITION OF PORT USE
;
; GPIO map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|-----|
;| LED |     |     |     |     |     |
;
;#define    GPIO,GP0    ;
;#define    GPIO,GP1    ;
;#define    GPIO,GP2    ;
;#define    GPIO,GP3    ;
;#define    GPIO,GP4    ;
#define    GPIO LED,GP5     ; LED between pin
                            and Vss
;
;*****************************************************************
; ################################################################

        LIST      p=12F519         ; Processor definition
        #include <p12F519.inc>
```

```
        Radix     DEC               ; Decimal Root

; ######################################################################
;======================================================================
;                           CONFIGURATION
;
; Internal oscillator, no WDT, no CP, pin4=GP3
  __config _IntRC_OSC & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF & _CPDF_OFF &
_MCLRE_OFF


; ######################################################################
;======================================================================
;                           LOCAL MACROS
Accendi_LED MACRO
            Bsf   LED      ; lights up the
              ENDM LED

; ######################################################################
;======================================================================
;                           RESET ENTRY
;
; movlw valore_calibrazione   First Intrinsic Instruction

; Reset Vector
RESET_VECTOR   ORG     0x00

 ; MOWF Internal Oscillator
        Calibration   OSCCAL

; ######################################################################
;======================================================================
;                        MAIN PROGRAM
;
MAIN:
; I/O initializations on reset
        CLRF    GPIO            ; GPIO preset latch to 0

; Assign GP5 Digital Output Function
; TRISGPIO        xx011111     GP5 out
        movlw   B'11011111'
        Tris    GPIO

; turns on LEDs by bringing the GP5 pin to level 1
        Accendi_LED

; Lock - Closed Loop
        Goto    $

;**********************************************************************
;======================================================================
;                             THE END
; End of source
        END
```

12F509-12F509 1Aw_5089.asm

```
;*****************************************************************
;-----------------------------------------------------------------
;
;    Title          :  Assembly & C - Tutorial 1Aw_5089
;                      Turn on an LED connected to the GP5 pin.
;                      The LED lights up when the power is turned on
;                      power supply and remains on as long as this
;                      is present
;
;    PIC            :  Code 12F508/509
;    Support        :  MPASM
;    Version        :  1.0
;    Date           :  01-05-2013
;    Hardware ref.  :
;    Author         :  Afg
;
;-----------------------------------------------------------------
;
; Pin use :
;   ----------------------------
;       12F508/509 @ 8 pin
;
;                    |__\/__|
;              Vdd -|1    8|- Vss
;              GP5 -|2    7|- GP0
;              GP4 -|3    6|- GP1
;         GP3/MCLR -|4    5|- GP2
;                    |______|
;
;    Vdd                 1: ++
;    GP5/OSC1/CLKIN      2: Out LED at Vss (R in series)
;    GP4/OSC2            3:
;    GP3/! MCLR/VPP      4:
;    GP2/T0CKI           5:
;    GP1/ICSPCLK         6:
;    GP0/ICSPDAT         7:
;    Vss                 8: --
;
;*****************************************************************
;=================================================================
;             DEFINITION OF PORT USE
;
; GPIO map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|-----|
;| LED |     |     |     |     |     |
;
;#define    GPIO,GP0    ;
;#define    GPIO,GP1    ;
;#define    GPIO,GP2    ;
;#define    GPIO,GP3    ;
;#define    GPIO,GP4    ;
#define    GPIO LED,GP5     ; LED between pin 2
                            and Vss
;
; ###############################################################
; Choice of #ifdef
 processor      12F509
        LIST      p=12F509          ; Processor Definition
        #include <p12F509.inc>
```

```
 #endif
 #ifdef 12F508
        LIST      p=12F508          ; Processor definition
        #include <p12F508.inc>
 #endif

        Radix     DEC               ; Decimal Root

; ######################################################################
;======================================================================
;                           CONFIGURATION
;
; Internal oscillator, no WDT, no CP, pin4=GP3
  __config _IntRC_OSC & _WDT_OFF & _CP_OFF & _MCLRE_ON

; ######################################################################
;======================================================================
;                        LOCAL MACROS
;
; Controls for the
 Accendi_LED LED
              MACRO
        Bsf
                END
              M
              LEDs

; ######################################################################
;======================================================================
;                           RESET ENTRY
;
; movlw valore_calibrazione   First Intrinsic Instruction

; Reset Vector
        ORG     0x00

 ; MOWF Internal Oscillator
        Calibration   OSCCAL


; ######################################################################
;======================================================================
;                          MAIN PROGRAM
;
MAIN:
; Reset Initializations
        CLRF    GPIO            ; GPIO preset latch to 0

; Assign GP5 Digital Output Function
; TRISGPIO        xx011111     GP5 out
        movlw   B'11011111'
        Tris    GPIO


; turns on LEDs by bringing the GP5 pin to level 1
        Accendi_LED

; Lock - Closed Loop
        Goto    $

;**********************************************************************
; ====================================================================
;                             THE END
```

```
; End of source
        END
```

10F200-10F202 1Aw_20x.asm

```
;*****************************************************************
;-----------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 1Aw_200
;                     Turn on an LED connected to the GP0 pin.
;                     The LED lights up when the power is turned on
;                     power supply and remains on as long as this
;                     is present

;    PIC            : 10F200/2
;    Support        : MPASM
;    Version        : 1.0
;    Date           : 01-05-2013
;    Hardware Ref.  :
;    Author         :Afg
;
;
;  Pin use :
;  ---------------------------------
;       10F200/202 @ 8 pin DIP           10F200/202 @ 6-pin SOT-23
;
;                   |__\/__|                     *______|
;              NC -|1    8|- GP3            GP0 -|1    6|- GP3
;             Vdd -|2    7|- Vss            Vss -|2    5|- Vdd
;             GP2 -|3    6|- NC             GP1 -|3    4|- GP2
;             GP1 -|4    5|- GP0                  |______|
;                  |______|
;
;                        DIP  SOT
;    NC                   1:     Nc
;    Vdd                  2:  5: ++
;    GP2/T0CKI/FOSC4      3:  4:
;    GP1/ICSPCLK          4:  3:
;    GP0/ICSPDAT          5:  1: Out LED to Vss
;    NC                   6:     Nc
;    Vss                  7:  2: --
;    GP3/MCLR/VPP         8:  6:
;
;*****************************************************************
;=================================================================
;            DEFINITION OF PORT USE
;
; GPIO map
; | 3 | 2 | 1 | 0 |
; |---|---|---|---|
; | in |      |     | LED |
;
#define   GPIO LED,GP0      ; LED between pin and Vss
;#define       GPIO,GP1     ;
;#define       GPIO,GP2     ;
;#define       GPIO,GP3     ; Input only
;
; ################################################################
 #ifdef 10F200
        LIST      p=10F200     ; Processor Definition
        #include <p10F200.inc>
 #endif
 #ifdef 10F202
        LIST      p=10F202     ; Processor Definition
        #include <p10F202.inc>
```

```
 #endif

        Radix     DEC           ; Decimal Numbers Defaults

; ######################################################################
;======================================================================
;                            CONFIGURATION                             ;
;
; No WDT, no CP, pin4=GP3
  __config _CP_OFF & _MCLRE_OFF & _WDT_OFF

; ######################################################################
;======================================================================
;                        LOCAL MACROS
;
; Controls for the
 Accendi_LED LED
               MACRO
        Bsf
                END
               M
               LEDs

; ######################################################################
;======================================================================
;                           RESET ENTRY
;
; Reset Vector
RESET_VECTOR   ORG     0x00

 ; MOWF Internal Oscillator
        Calibration   OSCCAL

; ######################################################################
;======================================================================
;                          MAIN PROGRAM
;
MAIN:
; Reset Initializations
        CLRF    GPIO             ; GPIO preset latch to 0

; TRISGPIO        -------0     GP0 out
        movlw   b'11111110'
        Tris    GPIO             ; To the Management
                                  Register

; Lights up LEDs
        Accendi_LED

; block
        Goto    $

;**********************************************************************
; ====================================================================
;                              THE END
;
        END
```

16F526/505 - 1Aw_526.asm

```
;*****************************************************************
;-----------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 1A_526
;                     Turn on an LED connected to the GP5 pin.
;                     The LED lights up when the power is turned on
;                     power supply and remains on as long as this
;                     is present
;
;    PIC            : 16F526-16F505
;    Support        : MPASM
;    Version        : 1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;-----------------------------------------------------------------
;
;  Pin use :
;  ..................................
;       16F505 - 16F526 @ 14 pin
;
;                    |__\/__|
;              Vdd -|1    14|- Vss
;              RB5 -|2    13|- RB0
;              RB4 -|3    12|- RB1
;         RB3/MCLR -|4    11|- RB22
;              RC5 -|5    10|- RC0
;              RC4 -|6     9|- RC1
;              RC3 -|7     8|- RC2
;                   |______|
;
;    Vdd                       1: ++
;    RB5/OSC1/CLKIN            2: Out LED to Vss
;    RB4/OSC2/CLKOUT           3:
;    RB3/! MCLR/VPP            4:
;    RC5/T0CKI                 5:
;    RC4/[C2OUT]               6:
;    RC3                       7:
;    RC2/[Cvref]               8:
;    RC1/[C2IN-]               9:
;    RC0/[C2IN+]              10:
;    RB2/[C1OUT/AN2]          11:
;    RB1/[C1IN-/AN1/]ICSPC    12:
;    RB0/[C1IN+/AN0/]ICSPD    13:
;    Vss                      14: --
;
;    [ ] only 16F526
;
;*****************************************************************
;=================================================================
;            DEFINITION OF PORT USE
;
;P ORTC not used
;
; PRTB map
; | 5 | 4 | 3 | 2 | 1 | 0 |
; |---|---|---|---|---|---|
; | LED |   |   |   |   |   |
```

```
;
#define   LED PORTB,RB5     ; LED between pin and Vss
;#define       PORTB,RB3
;#define       PORTB,RB2
;#define       PORTB,RB1
;#define       PORTB,RB0

; ########################################################################
;                           PROCESSOR SELECTION
 #ifdef 16F526
      LIST p=16F526
      #include <p16F526.inc>
  #endif
  #ifdef 16F505
      LIST p=16F505
      #include <p16F505.inc>
 #endif

; ########################################################################
;                            CONFIGURATION
;
 #ifdef 16F526
; Internal Oscillator, 4MHz, No WDT, No CP, No MCLR
 __config _IntRC_OSC_RB4 & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF & _CPDF_OFF &
MCLRE_OFF
 #endif

 #ifdef 16F505
; Internal Oscillator, 4MHz, No WDT, No CP, No MCLR
 __config _IntRC_OSC_RB4EN & _WDT_OFF & _CP_OFF & _MCLRE_OFF
 #endif

; ########################################################################
;=======================================================================
;                        LOCAL MACROS
;
; Controls for the
 Accendi_LED LED
               MACRO
         Bsf
                 END
               M
               LEDs

; ########################################################################
;                           RESET ENTRY
;
; Reset Vector
        ORG     0x00

 ; MOWF Internal Oscillator
        Calibration   OSCCAL

; ########################################################################
;                          MAIN PROGRAM
;
Main:
; Reset Initializations
        CLRF    PORTB             ; Preset Port Latch to 0

; RB5 come out
        movlw   b'00011111'
        Tris    PORTB             ; To the Management Register
```

```
; Lights up LEDs
```

```
         Accendi_LED

; Loop
        Goto    $

;*****************************************************************
;                               THE END
        END
```