TECNOLÓGICO DE MONTERREY®

Subject 3
ASSEMBLY LANGUAGE

What is the assembly language?

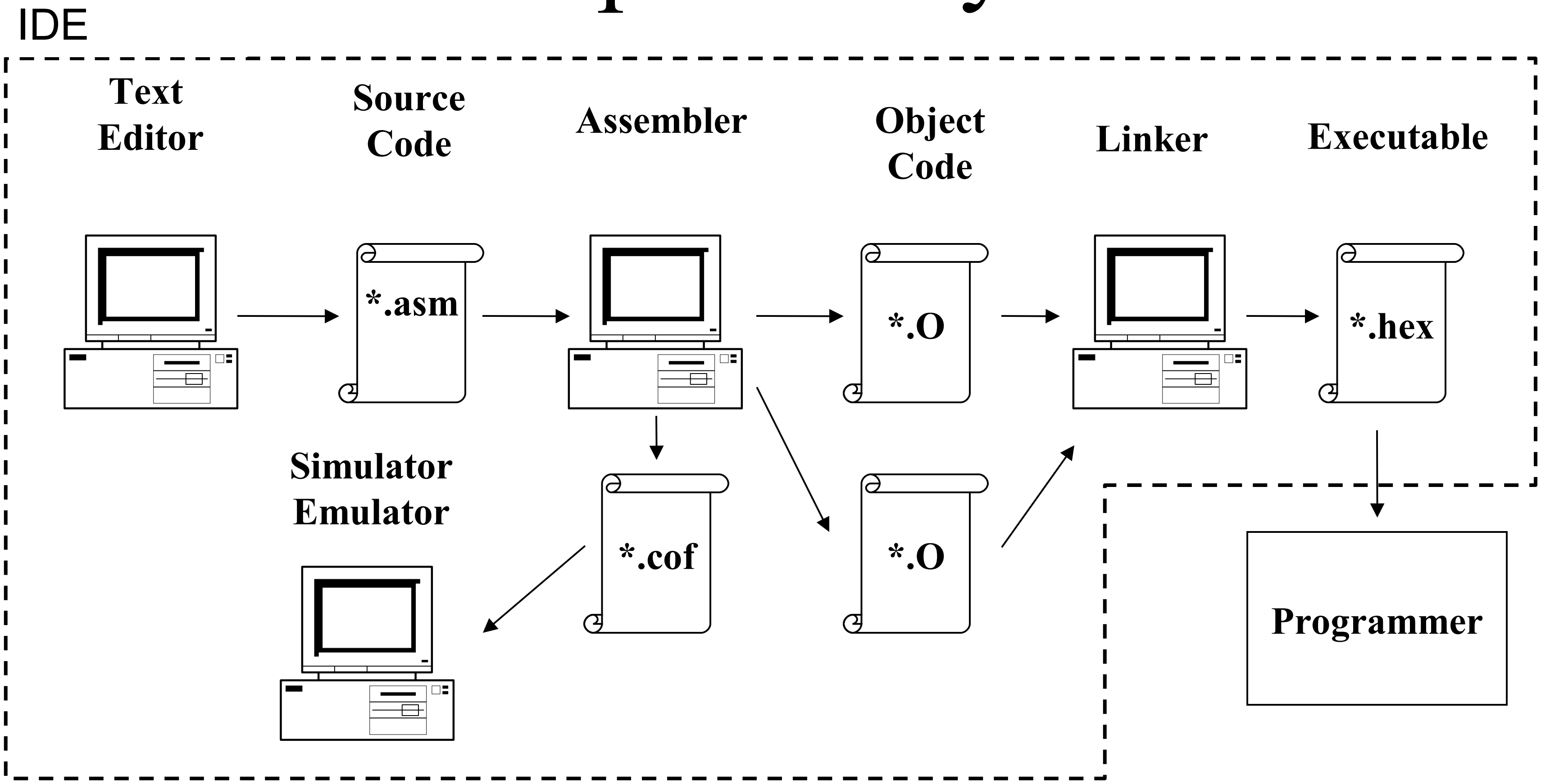
Is a computer language that is in between a mid or high level language and the binary codes that the CPU executes (machine language).

HIGH-MID LEVEL LANGUAGE	ASSEMBLY LANGUAGE			MACHINE LANGUAGE
For I = 1 to 10 do	Inicio:	movlw	0x01	0x0108
begin		movwf	0x30	0x010A
valor = valor + 2		movff	0x30,0x40	0xC030
end		movlw	0x00	0xE000
		movf	0x40,W	0x5040

Cross-assembler?

- The assembly language used to program the MCU is called “cross-assembler” since the code that generates is not executed in the development computer

Development Cycle



Language Elements

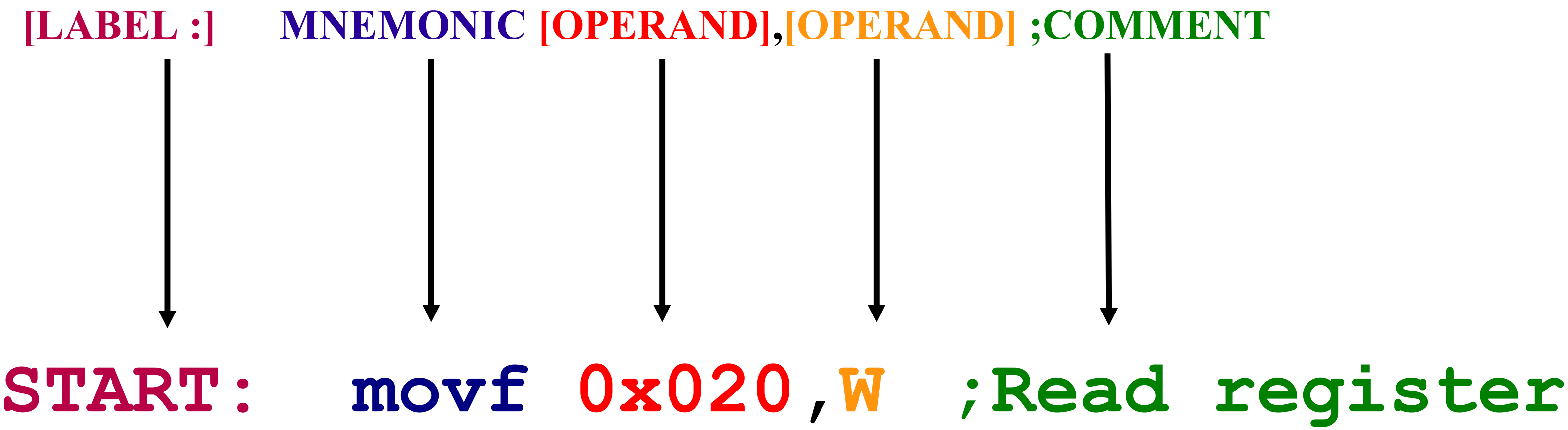
- Machine instructions
- Assembly directives
- Assembly control
- Comments

Source Code

Language Elements

- We will only cover the more basic elements since the purpose of the course is to understand the fundamental elements to understand a basic program and to be capable of reading high level code conversion to machine code.

Instruction format



[LABEL :] MNEMONIC [OPERAND],[OPERAND] ;COMMENT

```
START:   movf 0x020,W  ;Read register
```

Instruction Label

- Represents with a name the physical or relative address in the program memory where the instruction is located
  - Only the first 32 chars are considered
  - I suggest to end with the colon ":" character
  - Cannot contain spaces
  - Cannot start with a number

Coment

```
START: movf 0x020,W   ;Read register
```

- Comments:
  - – It must start with a semi colon “;”

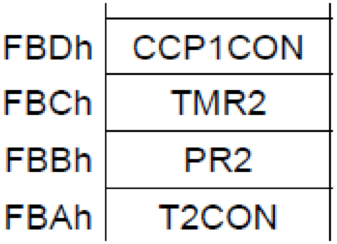
Predefined Symbols

- Names that the assembler handles “by default”, that represent registers and addresses used by the manufacturer. Typically they are contained in a file named like the particular device, but it can also be handled automatically by the IDE when the project is created
- PART_NO.INC → P18F45K22.INC

Instruction	**Equivalent**
bsf PORTA,0,A	**bsf 0x80,x00,A**

Expression evaluation

- To ease the legibility in a program the addresses, constants and operands can be represented in 3 ways.
  - Explicit movwf 0xBC,A
  - Predefined movwf TMR2, A
  - Expression movwf 0xBB + 1, A
  - Expression movwf PR2 +1, A

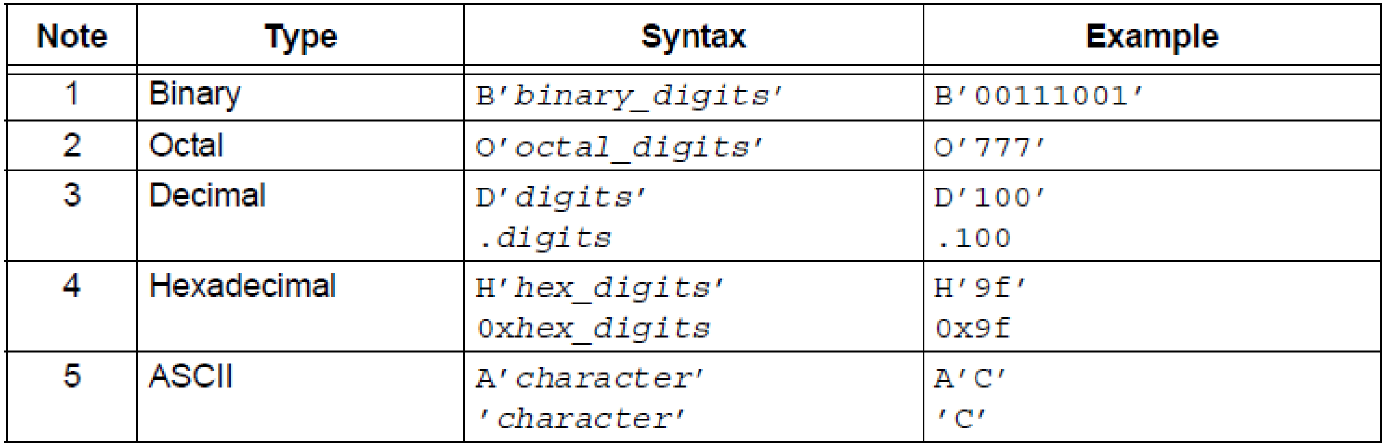


FBDh	CCP1CON
FBCh	TMR2
FBBh	PR2
FBAh	T2CON

Radix or numeric base

- To ease the legibility the assembly language allows to represent the numbers in several numeric bases. The can be configured general but can also be specified in each instruction by a prefix.

TABLE 3-3: RADIX SPECIFICATIONS – MPASM™ ASSEMBLER/MPLINK™ LINKER



Note	Type	Syntax	Example
1	Binary	B'*binary_digits*'	B'00111001'
2	Octal	O'*octal_digits*'	O'777'
3	Decimal	D'*digits*' .*digits*	D'100' .100
4	Hexadecimal	H'*hex_digits*' 0x*hex_digits*	H'9f' 0x9f
5	ASCII	A'*character*' '*character*'	A'C' 'C'

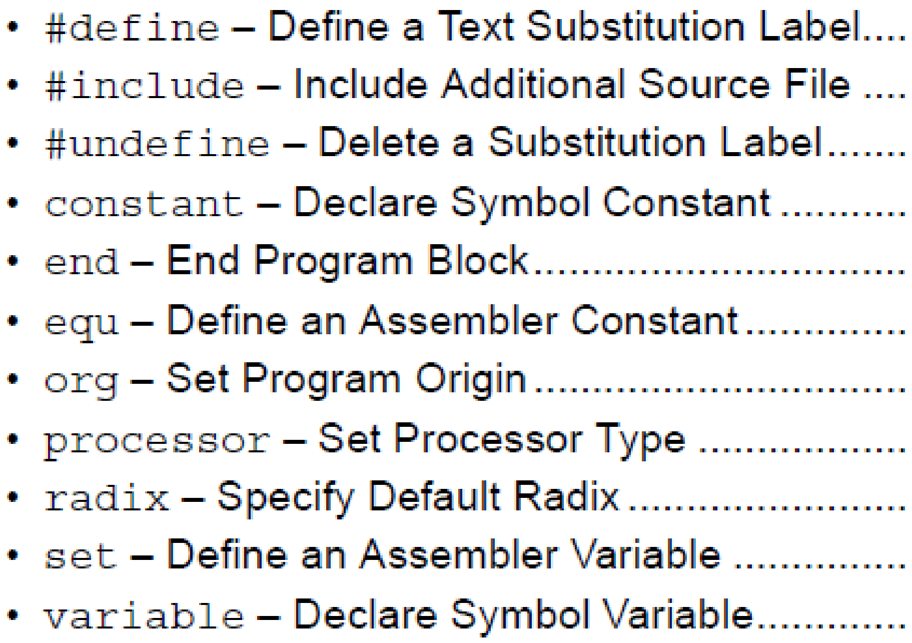
Assembly directives

Directives are assembler commands that appear in the source code but are not usually translated directly into opcodes. They are used to control the assembler: its input, and data allocation

- Control Directives
- Conditional Assembly Directives
- Data Directives
- Listing Directives
- Macro Directives
- Object File Directives

Control

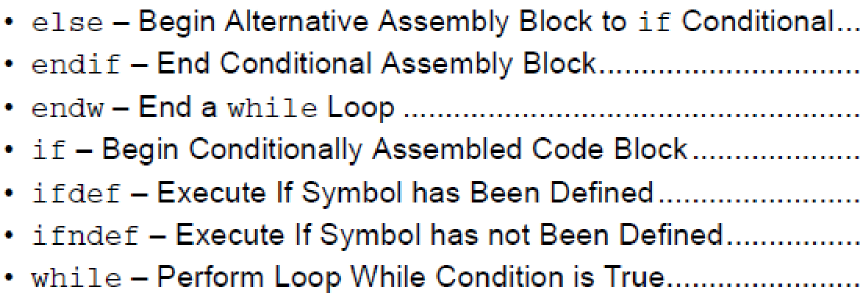
Control directives control how code is assembled



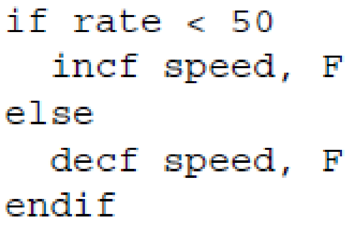
- #define – Define a Text Substitution Label....
- #include – Include Additional Source File ....
- #undefine – Delete a Substitution Label.......
- constant – Declare Symbol Constant ....
- end – End Program Block...............................
- equ – Define an Assembler Constant ....
- org – Set Program Origin ....
- processor – Set Processor Type ....
- radix – Specify Default Radix ....
- set – Define an Assembler Variable ....
- variable – Declare Symbol Variable.............

Conditional

Conditional assembly directives permit sections of conditionally assembled code. These are not run-time instructions like their C language counterparts. They define which code is assembled, not how the code executes



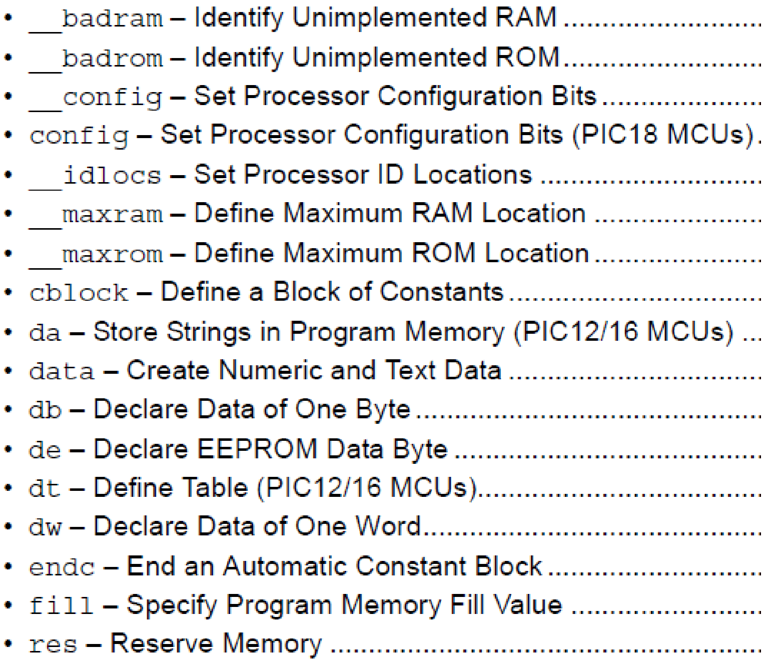
- else – Begin Alternative Assembly Block to if Conditional...
- endif – End Conditional Assembly Block..............................
- endw – End a while Loop ....
- if – Begin Conditionally Assembled Code Block...................
- ifdef – Execute If Symbol has Been Defined.......................
- ifndef – Execute If Symbol has not Been Defined...............
- while – Perform Loop While Condition is True......................



```
if rate < 50
  incf speed, F
else
  decf speed, F
endif
```

Data

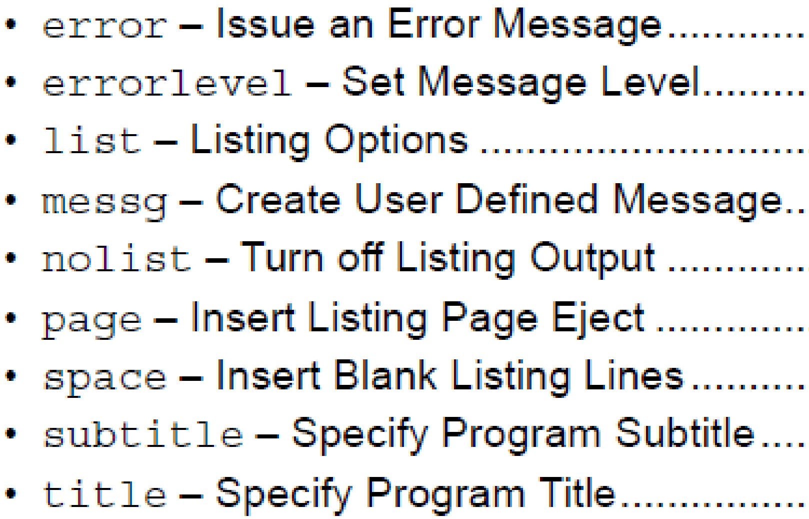
Data directives control the allocation of memory and provide a way to refer to data items symbolically, i.e., by meaningful names.



- __badram – Identify Unimplemented RAM ....
- __badrom – Identify Unimplemented ROM........................
- __config – Set Processor Configuration Bits ....
- config – Set Processor Configuration Bits (PIC18 MCUs).
- __idlocs – Set Processor ID Locations ....
- __maxram – Define Maximum RAM Location ....
- __maxrom – Define Maximum ROM Location ....
- cblock – Define a Block of Constants...............................
- da – Store Strings in Program Memory (PIC12/16 MCUs) ...
- data – Create Numeric and Text Data ....
- db – Declare Data of One Byte..
- de – Declare EEPROM Data Byte ....
- dt – Define Table (PIC12/16 MCUs)...................................
- dw – Declare Data of One Word.......................................
- endc – End an Automatic Constant Block ....
- fill – Specify Program Memory Fill Value ....
- res – Reserve Memory ....

Listing

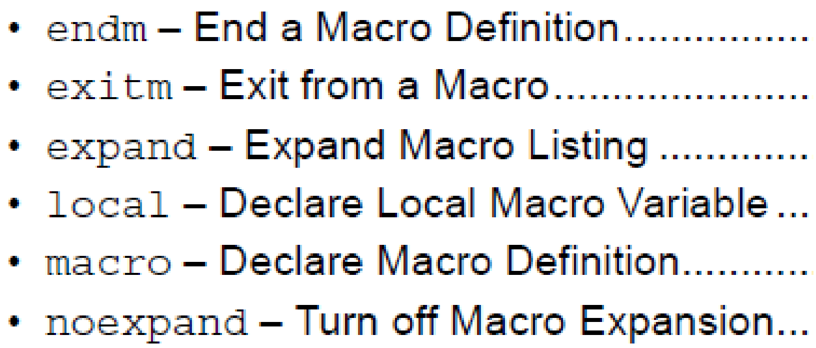
Listing directives control the MPASM assembler listing file format. These directives allow the specification of titles, pagination, and other listing control. Some listing directives also control how code is assembled.



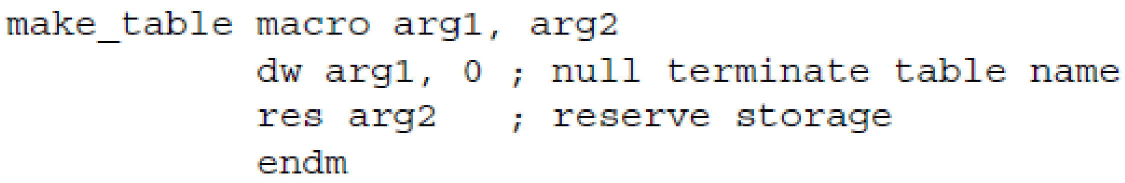
- error – Issue an Error Message...........
- errorlevel – Set Message Level........
- list – Listing Options ....
- messg – Create User Defined Message..
- nolist – Turn off Listing Output ....
- page – Insert Listing Page Eject ....
- space – Insert Blank Listing Lines..........
- subtitle – Specify Program Subtitle....
- title – Specify Program Title................

Macro

Macro directives control the execution and data allocation within macro body definitions.



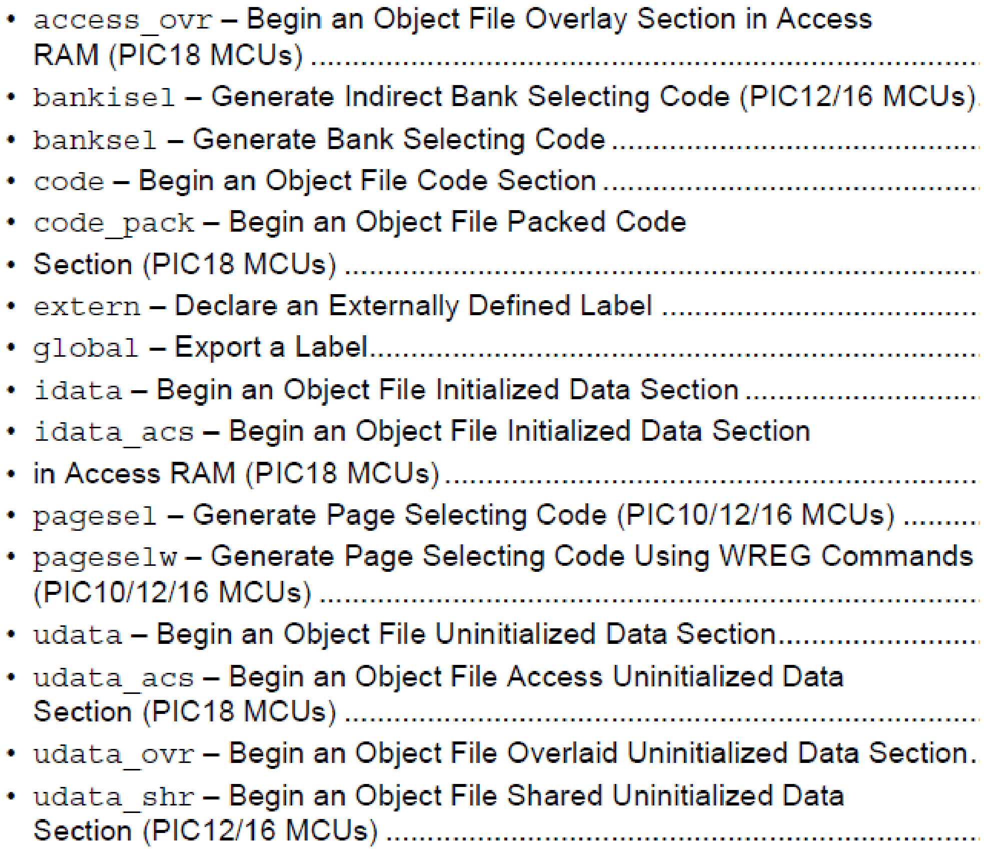
- endm – End a Macro Definition................
- exitm – Exit from a Macro......................
- expand – Expand Macro Listing ....
- local – Declare Local Macro Variable ...
- macro – Declare Macro Definition...........
- noexpand – Turn off Macro Expansion...



```
make_table macro arg1, arg2
           dw arg1, 0 ; null terminate table name
           res arg2   ; reserve storage
           endm
```

Objeto

Object file directives are used only when creating an object file.



- `access_ovr` – Begin an Object File Overlay Section in Access RAM (PIC18 MCUs) ....
- `bankisel` – Generate Indirect Bank Selecting Code (PIC12/16 MCUs).
- `banksel` – Generate Bank Selecting Code ....
- `code` – Begin an Object File Code Section ....
- `code_pack` – Begin an Object File Packed Code
- Section (PIC18 MCUs) ....
- `extern` – Declare an Externally Defined Label ....
- `global` – Export a Label..
- `idata` – Begin an Object File Initialized Data Section ....
- `idata_acs` – Begin an Object File Initialized Data Section
- in Access RAM (PIC18 MCUs) ....
- `pagesel` – Generate Page Selecting Code (PIC10/12/16 MCUs) ....
- `pageselw` – Generate Page Selecting Code Using WREG Commands (PIC10/12/16 MCUs) ....
- `udata` – Begin an Object File Uninitialized Data Section..
- `udata_acs` – Begin an Object File Access Uninitialized Data Section (PIC18 MCUs) ....
- `udata_ovr` – Begin an Object File Overlaid Uninitialized Data Section.
- `udata_shr` – Begin an Object File Shared Uninitialized Data Section (PIC12/16 MCUs) ....

Constant definition

- The equ directive

```
FOUR            equ     4         ;Assigns value of 4 to the FOUR symbol
OUT_PORT        equ     PORTB     ;Assigns a predefined name
ANOTHER_FOUR    equ     FOUR      ;Otra asignación
TWO             equ     FOUR/2 ;Se permiten operaciones
```

Definición de Variables

- The set directive
  - Assings a symbol to the file register (data adress)

```
MY_VAR   set 0x00   ;Assings name MY_VAR to adress 0x00
```

Constant definitions to allocate in ROM

- Directive db
  - Defines a constant in memory program.

```
COPYRG_MSG:    db     "Copyright 2012"     ;Ascii literal
CONSTANTS:     db     0x01,0x03            ;Constant table
MY_MIX:        db     'A',"Hola",04h       ;Mezclado
```

Cuando se usa en la generación de código relocalizable se puede usar para definir espacio de RAM.

The “org” directive

- Assings the preceding code the physical address where it will be allocated in program memory

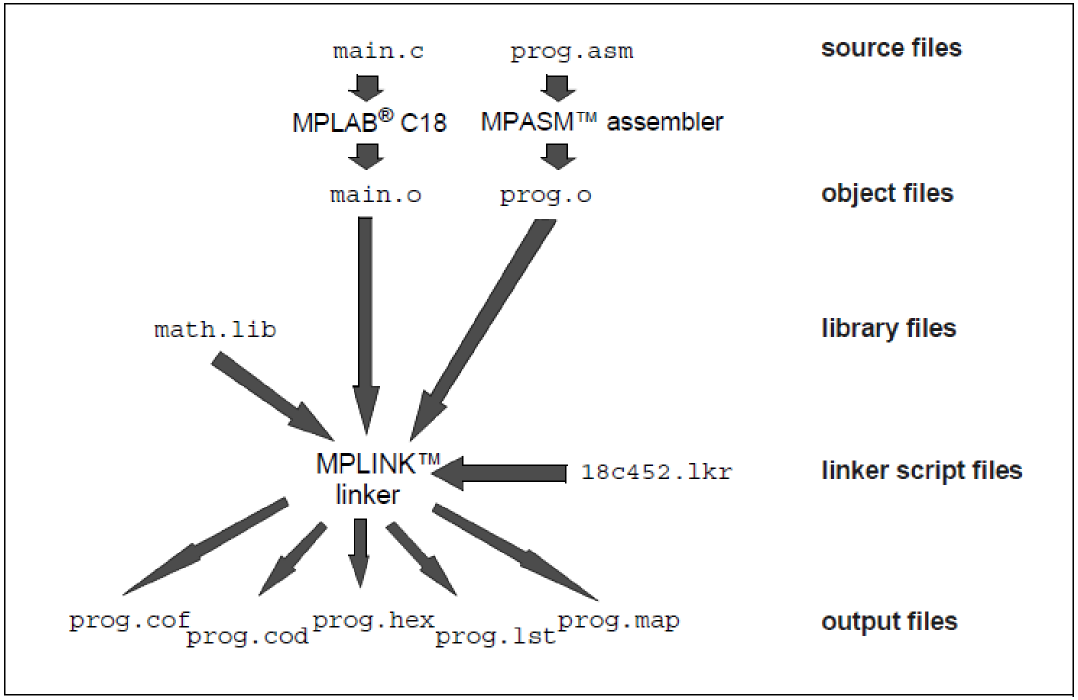
```
               org 0x0000  ;Reset address
               bra main_p  ;At rest, execute main program

               org 0x0100  ;Code allocated at 0x0100 address
main_p:  addwf 0x40,WREG,A
```

More about directives

- The assembler has the capability to handle independent code modules. These directives are out of the scope of this course

The Linker

- The Linker is a program that is used to merge several "object" files in to single code. Also control the program and data memory physical allocation (addresses)

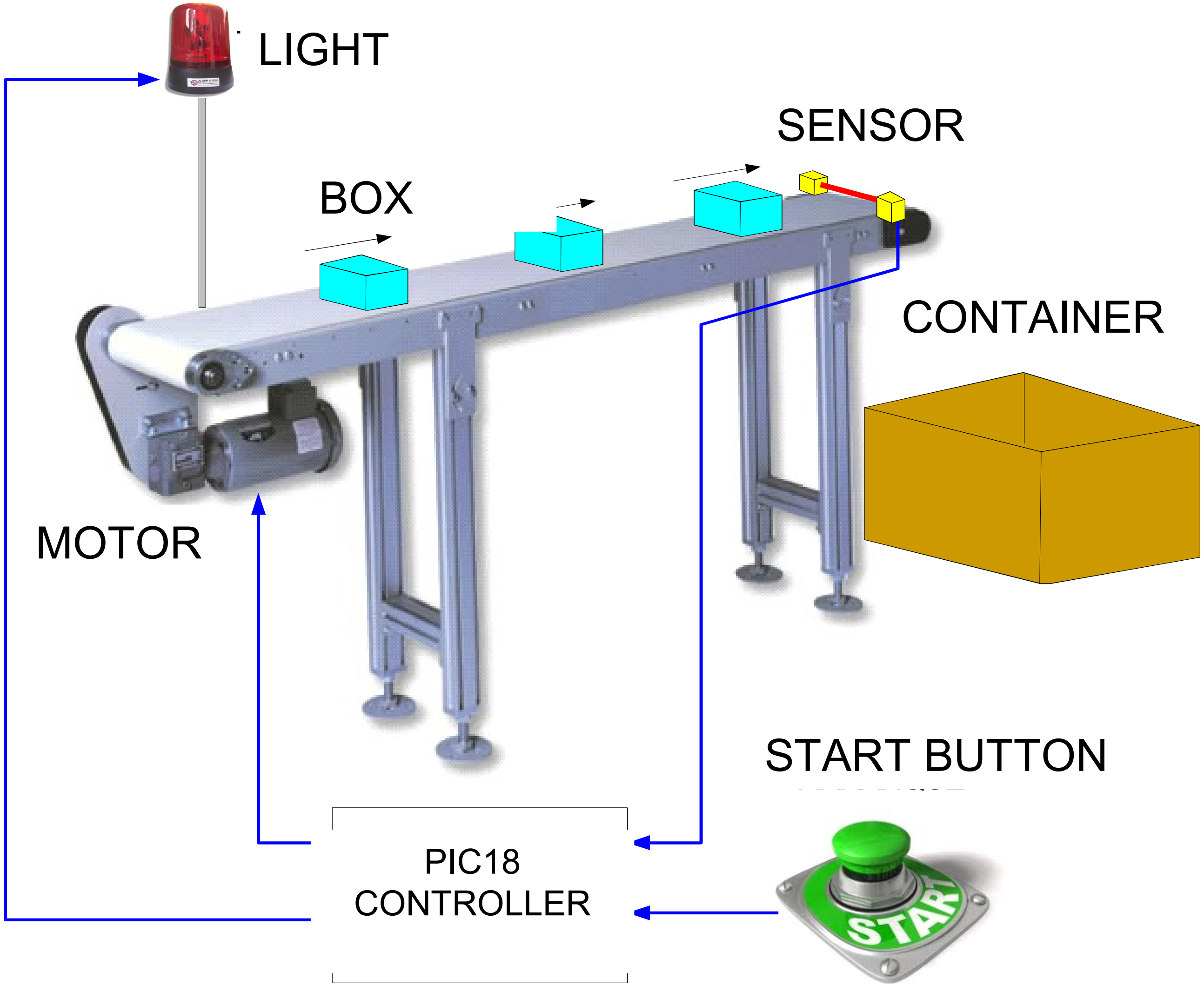


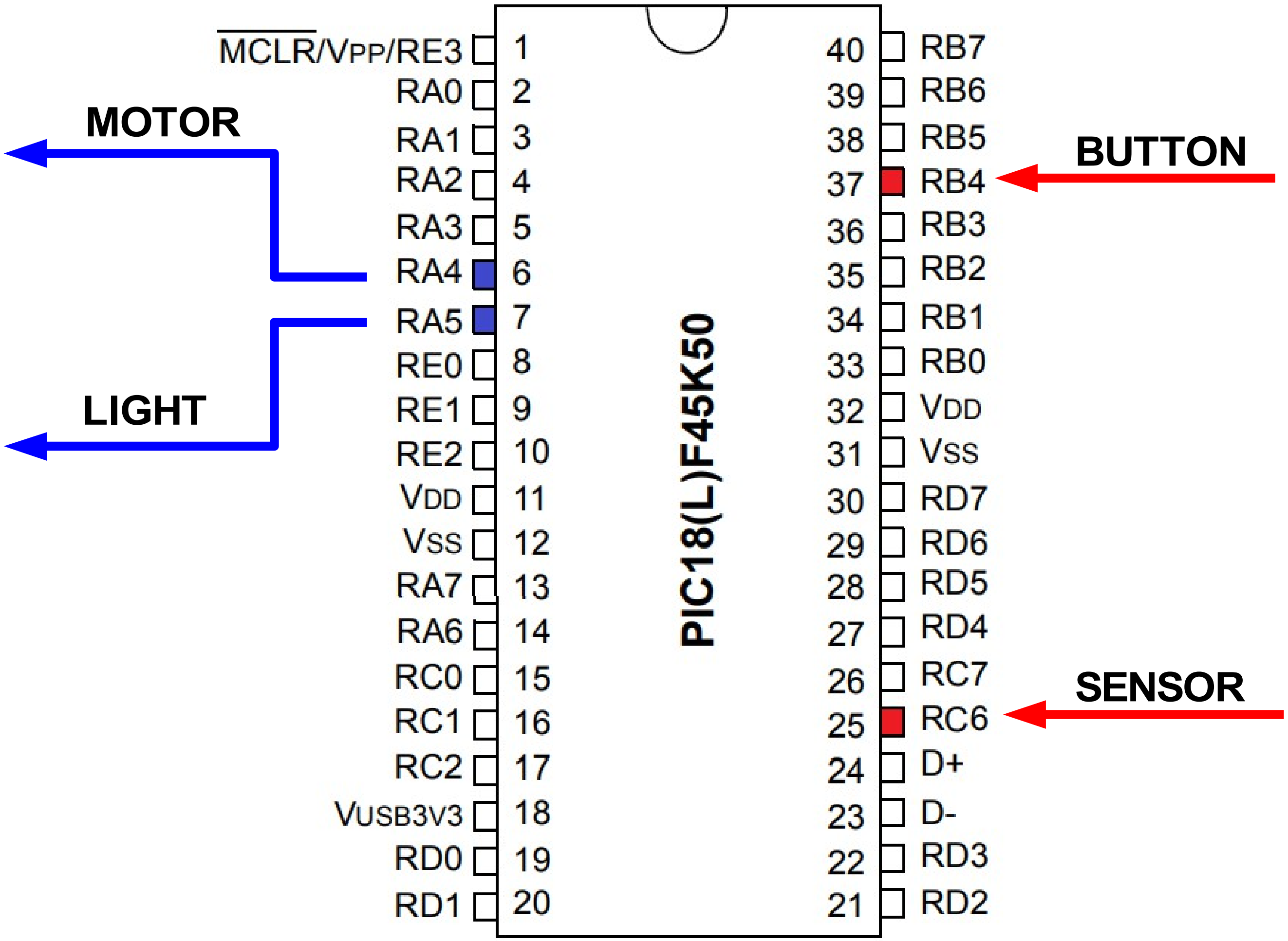
Programming

Programming process

- Understand the problem to solve
- Define the solution strategy
- Define the algorithm(s)
- Code
- Test and code depuration (debug)
- Release
- Maintenance

Conveyor belt controller



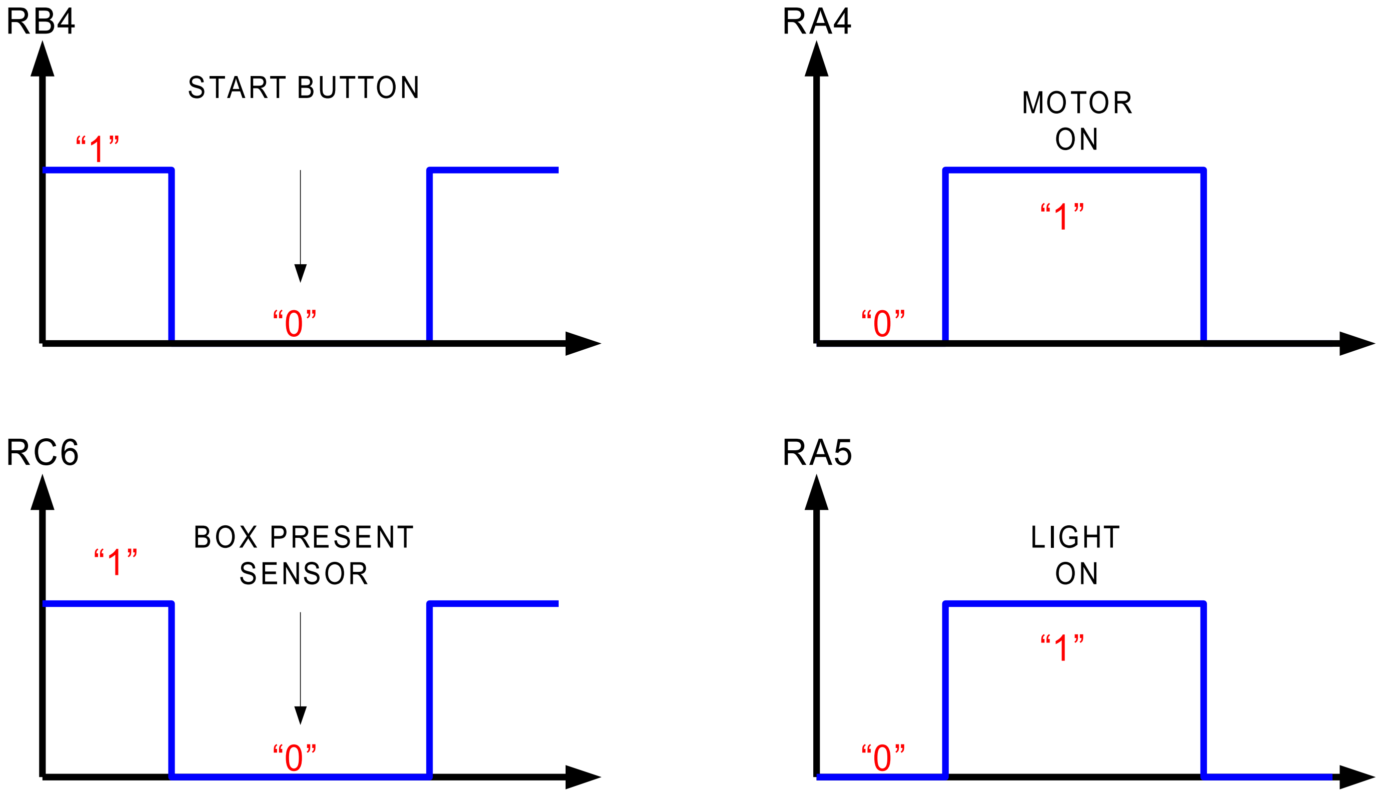

MOTOR
LIGHT
BUTTON
SENSOR
PIC18(L)F45K50
MCLR/VPP/RE3 1
RA0 2
RA1 3
RA2 4
RA3 5
RA4 6
RA5 7
RE0 8
RE1 9
RE2 10
VDD 11
VSS 12
RA7 13
RA6 14
RC0 15
RC1 16
RC2 17
VUSB3V3 18
RD0 19
RD1 20
40 RB7
39 RB6
38 RB5
37 RB4
36 RB3
35 RB2
34 RB1
33 RB0
32 VDD
31 VSS
30 RD7
29 RD6
28 RD5
27 RD4
26 RC7
25 RC6
24 D+
23 D-
22 RD3
21 RD2

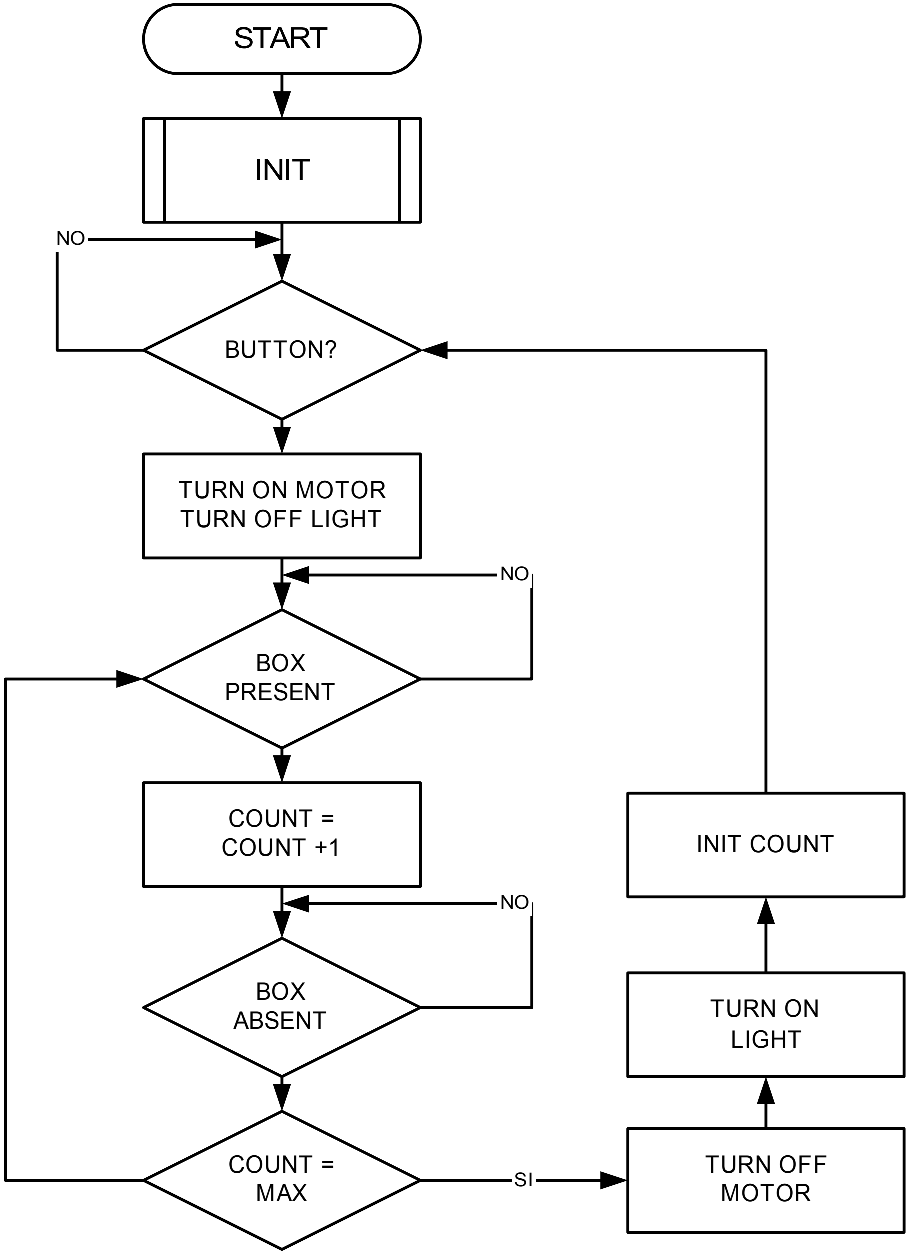
Conveyor belt controller

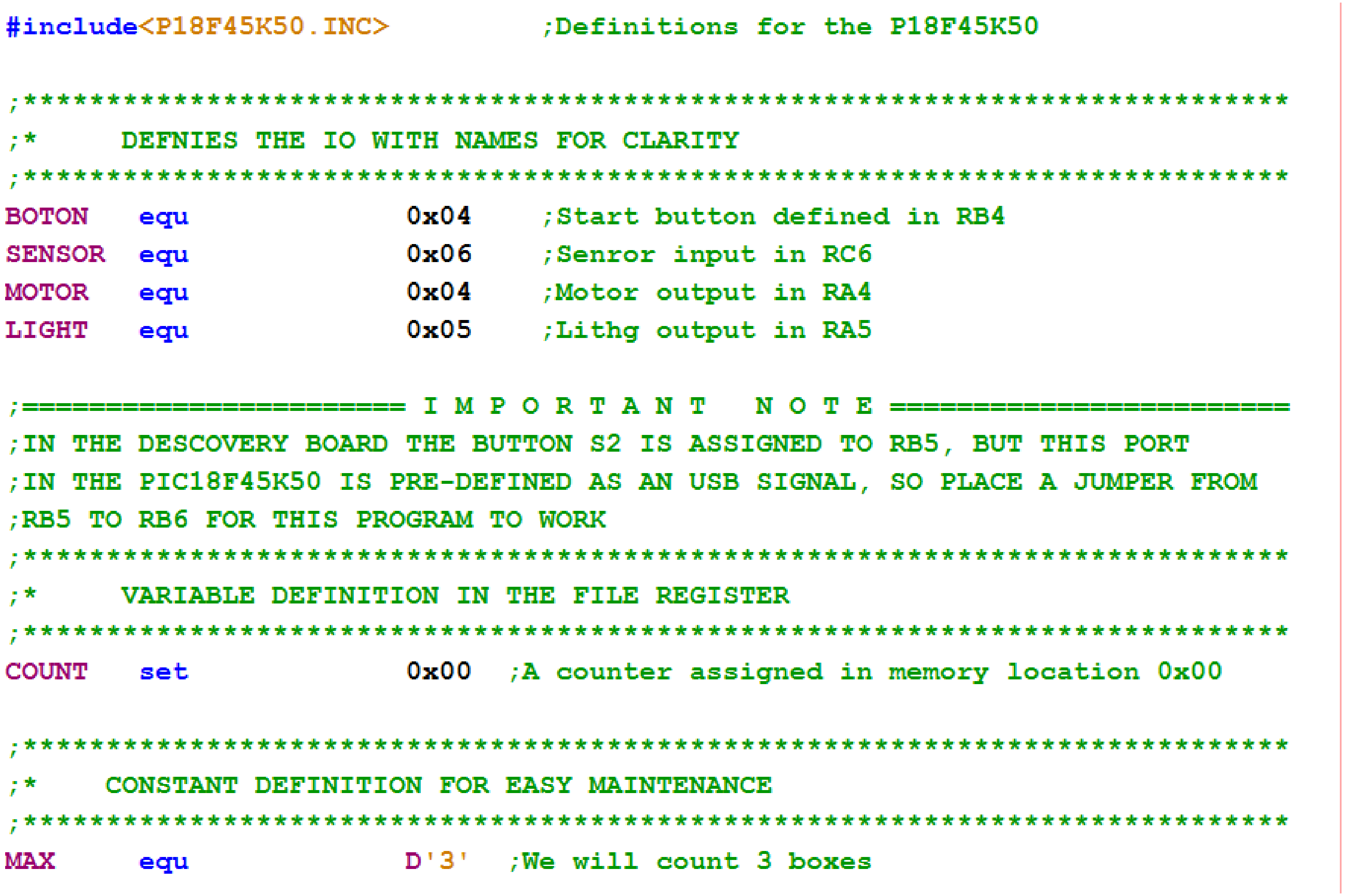
- The control program must do the following:
  - Wait for the start button to be pressed
  - Count the number of boxes
  - If the number is equal to MAXIMO
    - Stop the conveyor belt
    - Turn the light
    - Wait again for the start buton

Conveyor belt controller

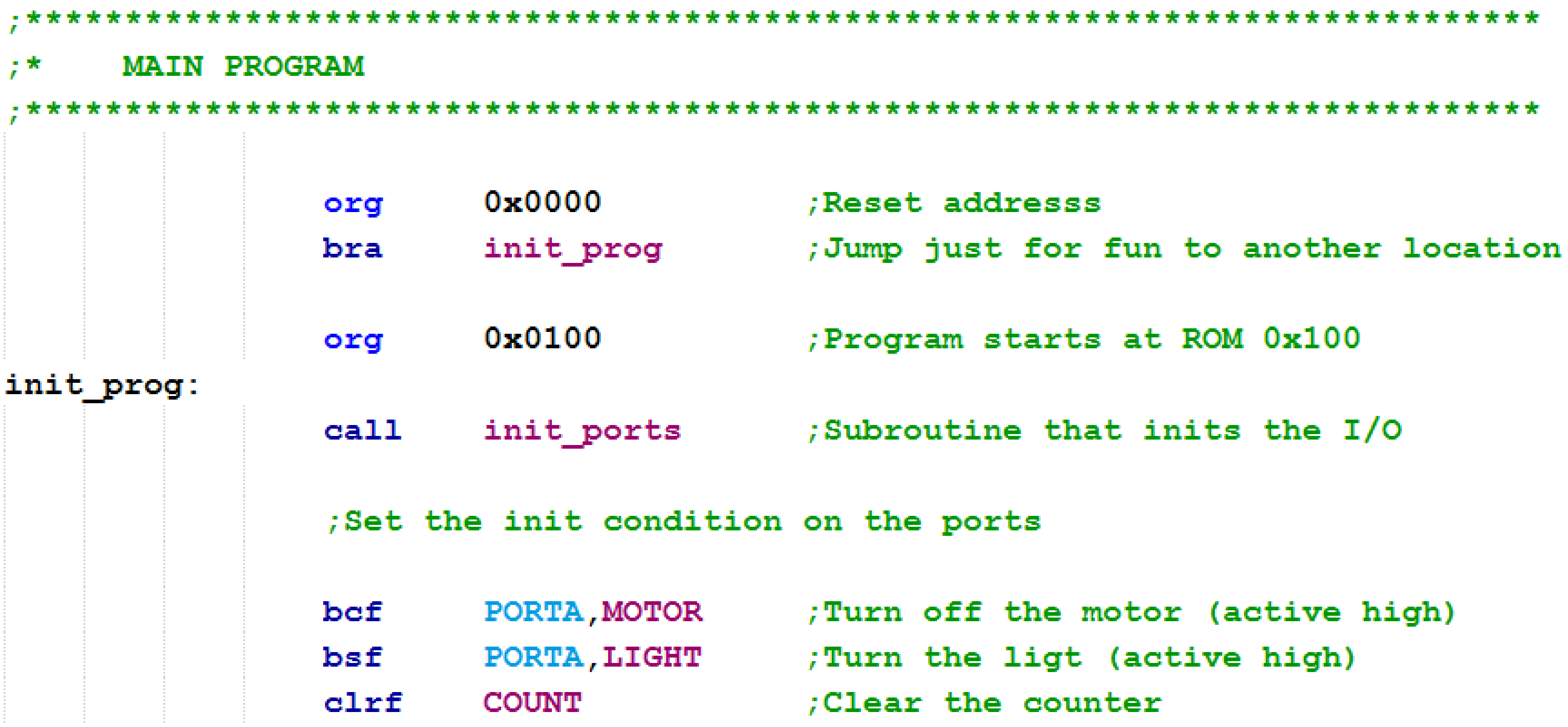
- Signals for actuators and from sensors:



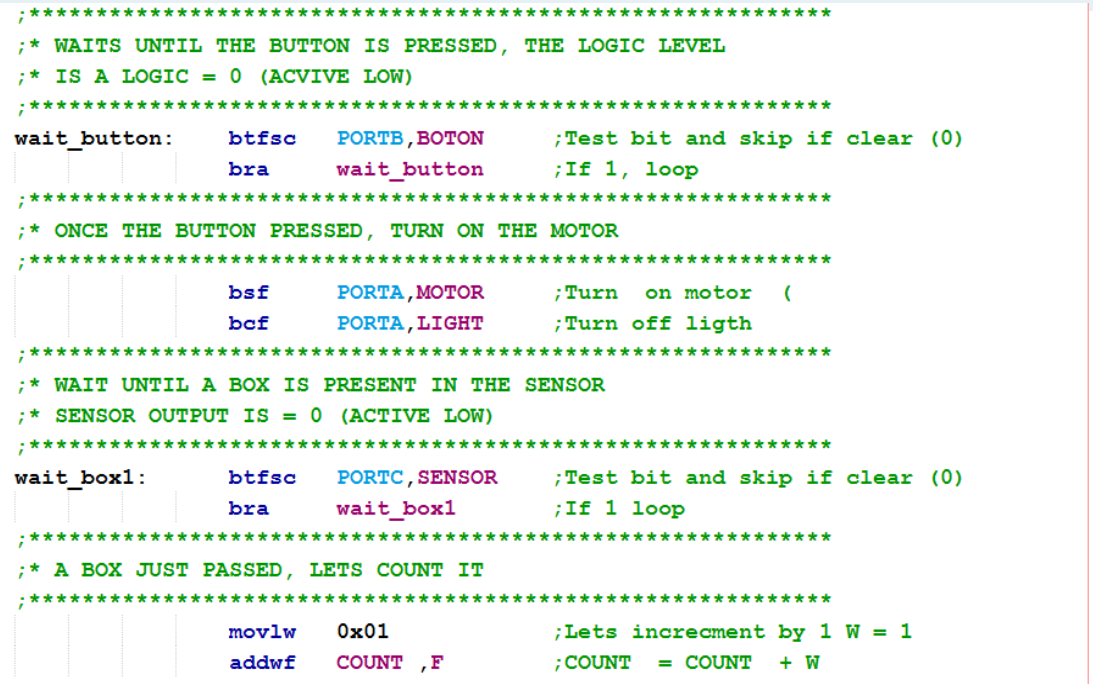

START
INIT
NO
BUTTON?
TURN ON MOTOR
TURN OFF LIGHT
NO
BOX
PRESENT
COUNT =
COUNT +1
NO
BOX
ABSENT
COUNT =
MAX
SI
TURN OFF
MOTOR
TURN ON
LIGHT
INIT COUNT



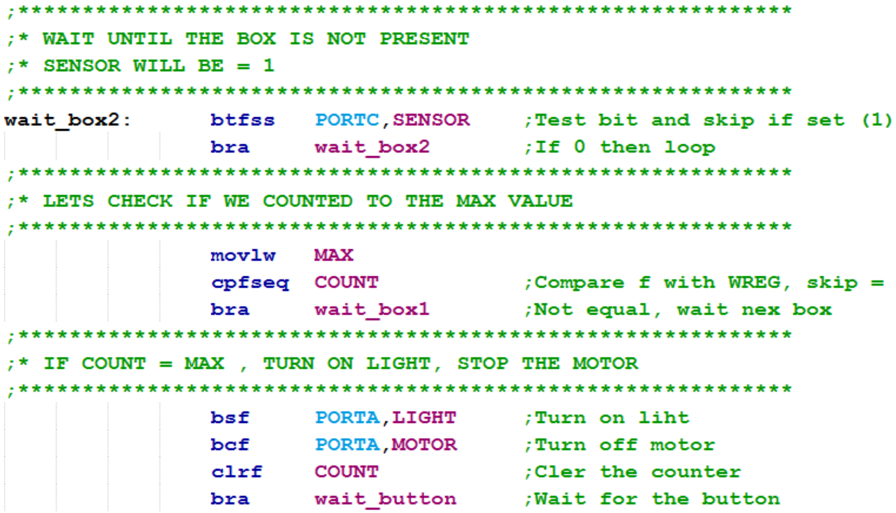
```
#include<P18F45K50.INC>         ;Definitions for the P18F45K50

;****************************************************************************
;*      DEFNIES THE IO WITH NAMES FOR CLARITY
;****************************************************************************
BOTON   equu            0x04    ;Start button defined in RB4
SENSOR  equ             0x06    ;Senror input in RC6
MOTOR   equ             0x04    ;Motor output in RA4
LIGHT   equ             0x05    ;Lithg output in RA5

;======================= I M P O R T A N T   N O T E ========================
;IN THE DESCOVERY BOARD THE BUTTON S2 IS ASSIGNED TO RB5, BUT THIS PORT
;IN THE PIC18F45K50 IS PRE-DEFINED AS AN USB SIGNAL, SO PLACE A JUMPER FROM
;RB5 TO RB6 FOR THIS PROGRAM TO WORK
;****************************************************************************
;*      VARIABLE DEFINITION IN THE FILE REGISTER
;****************************************************************************
COUNT   set             0x00  ;A counter assigned in memory location 0x00

;****************************************************************************
;*     CONSTANT DEFINITION FOR EASY MAINTENANCE
;****************************************************************************
MAX     equ             D'3'  ;We will count 3 boxes
```



```
;******************************************************************************
;*    MAIN PROGRAM
;******************************************************************************

                org     0x0000          ;Reset addresss
                bra     init_prog       ;Jump just for fun to another location

                org     0x0100          ;Program starts at ROM 0x100
init_prog:
                call    init_ports      ;Subroutine that inits the I/O

                ;Set the init condition on the ports

                bcf     PORTA,MOTOR     ;Turn off the motor (active high)
                bsf     PORTA,LIGHT     ;Turn the ligt (active high)
                clrf    COUNT           ;Clear the counter
```



```
;**************************************************************
;* WAITS UNTIL THE BUTTON IS PRESSED, THE LOGIC LEVEL
;* IS A LOGIC = 0 (ACVIVE LOW)
;**************************************************************
wait_button:    btfsc   PORTB,BOTON     ;Test bit and skip if clear (0)
                bra     wait_button     ;If 1, loop
;**************************************************************
;* ONCE THE BUTTON PRESSED, TURN ON THE MOTOR
;**************************************************************
                bsf     PORTA,MOTOR     ;Turn  on motor  (
                bcf     PORTA,LIGHT     ;Turn off ligth
;**************************************************************
;* WAIT UNTIL A BOX IS PRESENT IN THE SENSOR
;* SENSOR OUTPUT IS = 0 (ACTIVE LOW)
;**************************************************************
wait_box1:      btfsc   PORTC,SENSOR    ;Test bit and skip if clear (0)
                bra     wait_box1       ;If 1 loop
;**************************************************************
;* A BOX JUST PASSED, LETS COUNT IT
;**************************************************************
                movlw   0x01            ;Lets increcment by 1 W = 1
                addwf   COUNT ,F        ;COUNT  = COUNT  + W
```



```
;************************************************************
;* WAIT UNTIL THE BOX IS NOT PRESENT
;* SENSOR WILL BE = 1
;************************************************************
wait_box2:      btfss   PORTC,SENSOR    ;Test bit and skip if set (1)
                bra     wait_box2       ;If 0 then loop
;************************************************************
;* LETS CHECK IF WE COUNTED TO THE MAX VALUE
;************************************************************
                movlw   MAX
                cpfseq  COUNT           ;Compare f with WREG, skip =
                bra     wait_box1       ;Not equal, wait nex box
;************************************************************
;* IF COUNT = MAX , TURN ON LIGHT, STOP THE MOTOR
;************************************************************
                bsf     PORTA,LIGHT     ;Turn on liht
                bcf     PORTA,MOTOR     ;Turn off motor
                clrf    COUNT           ;Cler the counter
                bra     wait_button     ;Wait for the button
```

Output files

- *.O
  - Object code
- *.LST
  - Program listing
- *.COF
  - Files for the debug tool
- *.HEX
  - Execuable file (fINTEL HEX format)

The executable file

- The code that will be executed by the microcontroller can ve generated in several output formats:
  - Extended Intel hex (readable text)
  - Short Intel hex (readable text)
  - Motorola S records (readable text)
  - Tektronix (readable text)
  - Binary (Sequential binary file fist byte starts at 0x0000)

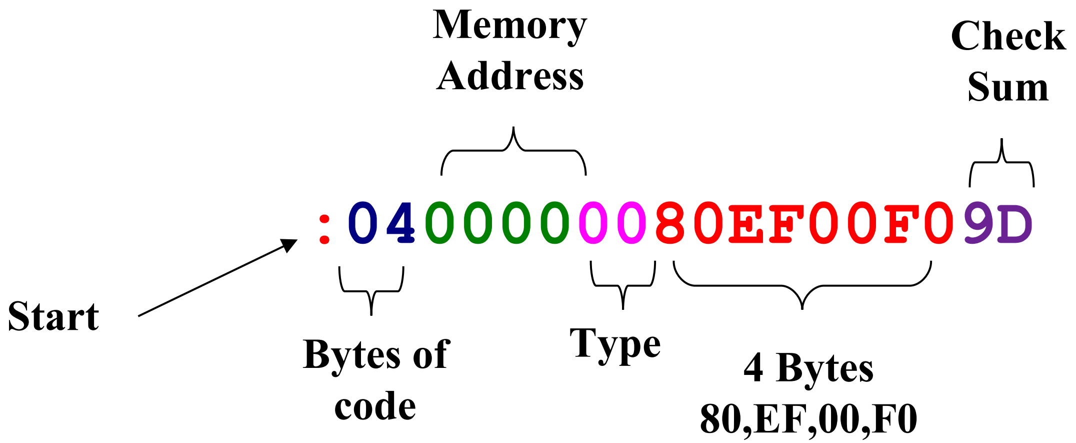
The Intel Hex.

- Used by many chip programmers is the more popular.
  - Each line is called a “register”
  - There are four types of registers

• Data type	0x00
• End of file	0x01
• Segmentation	0x02
• Allows extended addresses (32bits)	0x04
• Start of linear adress (32bits)	0x05

Intel hex file

```
:020000040000FA
:0400000080EF00F09D
:1001000095EC00F080928086006A80B0FED78082F5
:10011000808680A4FED7010E002680B4FED7030E91
:100120000062F7D78096006AF0D70F01806A896A6B
:0A013000E00E386FF50E926E12001B
:00000001FF
```


Memory Address

Check Sum

:0400000080EF00F09D

Start

Bytes of code

Type

4 Bytes 80,EF,00,F0

Documentation

- Programs in assembly as in any language must be documented in detail to ease future maintenance.
- The assembly language is very “cryptic” if we compare it to high mid and high level languages. This is the main reason that documentation becomes critical. High and mid level languages inherently documented by its syntax.
- A program with no documentation is very hared if not imposible to maintain and could end up in the trash. After some time, even the programmer forgets how it Works.
- *“Firmware is the most expensive thing in the universe, yet we do Little to control its cost”(Jack Ganssle dixit), maintenance is one of the biggest cost (me dixit)*

Documentation

- It is recommended that each program starts with the following information
  - Author
  - Creation date
  - Original file name
  - Version review and date
  - Layout of the ports and signal flow
  - A general abstract on how it works,
  - Tools employed for the development and version number
    - IDE, Assembly/Linker/etc.
  - Additional modules that are required to generate executable
  - Special notes and instructions,

Documentation

- In the module it is recommended to have a section where we define:
  - Previous version number
  - Previous file name
  - Current version
  - Current file
  - Date of the revision
  - Abstract of the performed changes

Documentation.

- It is recommended that each procedure and subroutine has a header where we describe the purpose of the code. +
- Its recommended the use of prolific comments to indicate what that an instruction or a sequence are doing.
- It is recommended that the variables and constants have coherent names, to be as descriptive as possible avoiding very long names.
- There are many styles, standards already, use an existing one and adapt it to your needs
- If you make a living generate a code, make it with the pride that some one else can understand and mantain and say WOW

Example using the MPLAB (SUBJECT_03_1.ASM)