Indian Institute of Technology Bombay

Embedded C

Programming the PIC18F using C18

Suryakant Toraskar
smtoraskar.iitbombay@gmail.com
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay

C Compiler

- A compiler converts a high level language program to machine instructions for the target processor.
- A cross compiler is a compiler that runs on a processor (usually a PC) that is different from the target processor.
- Most embedded systems are now programmed using the C/C++ language.

Many C Compilers available:

HI-TECH, PICCTM, www.htsoft.com

IAR-EWB-PIC, www.iar.com

CCS PIC18/XC8 C Compiler, www.ccsinfo.com

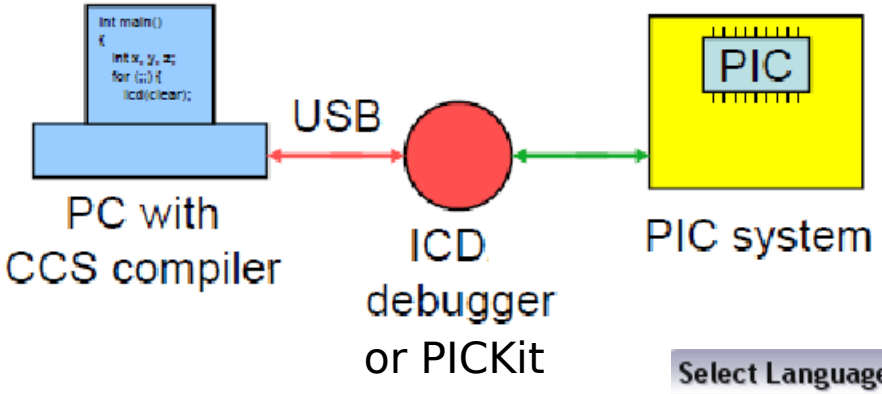
The C18 Compiler

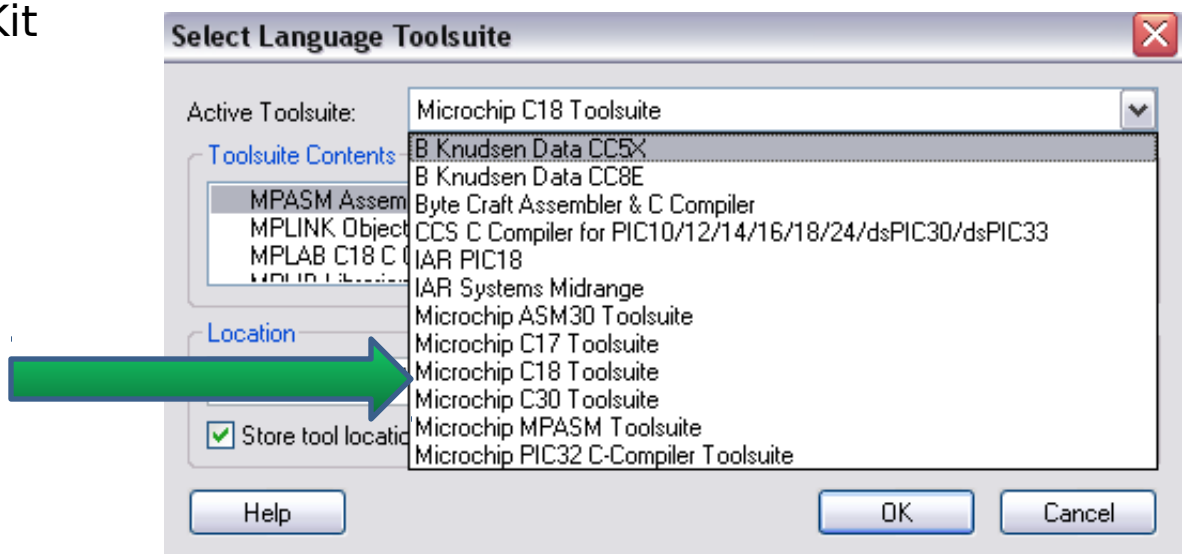
- The C18 compiler is a program used for programming the PIC in C-Language.
- Programming in C-Language greatly reduces

  development time.
- C is **NOT** as efficient as assembly

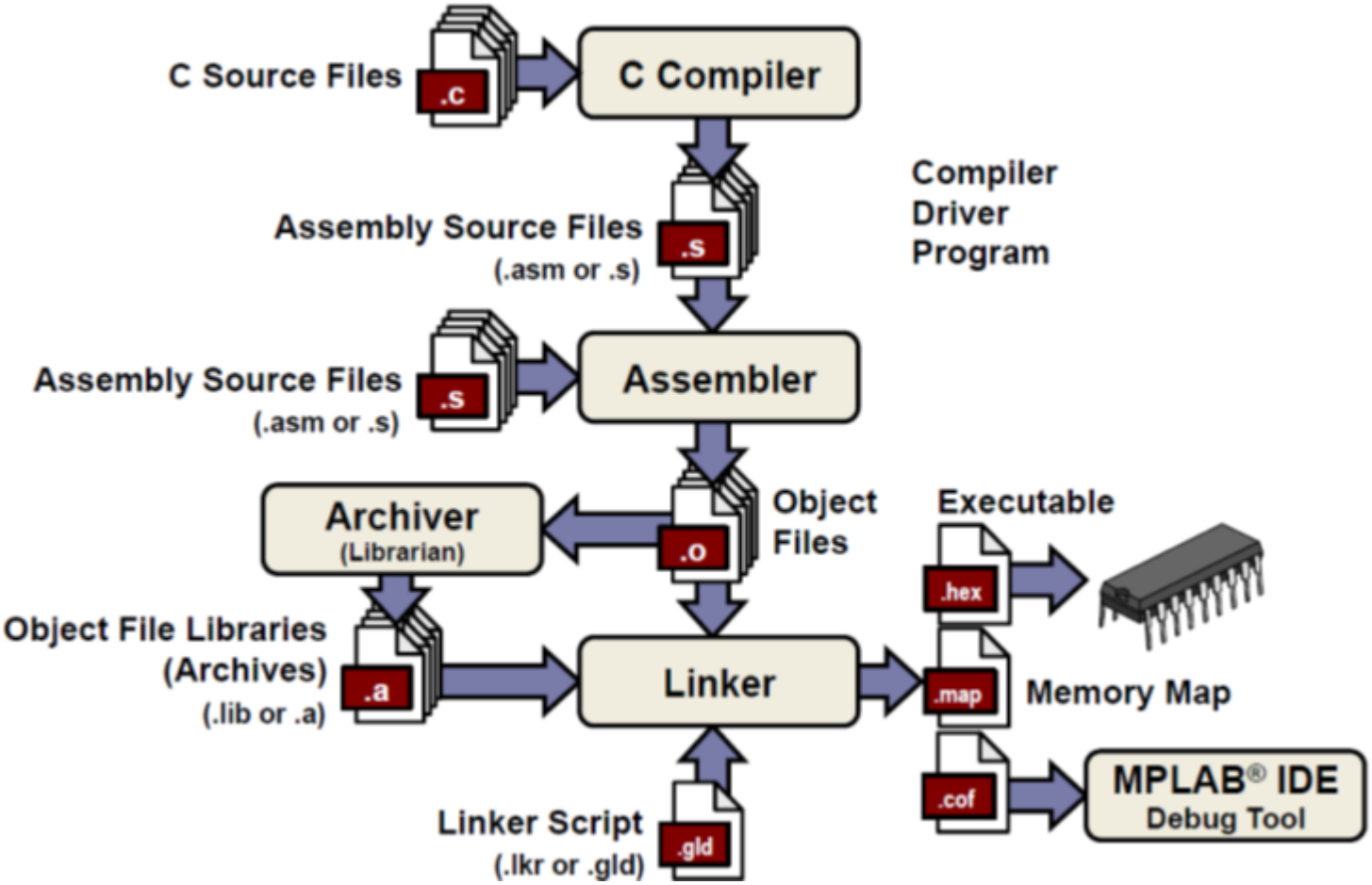
  *A good assembly programmer can usually do better than the compiler, no matter what the optimization level – C **WILL** use more memory*

MPLAB
and C Compilers / Assemblers

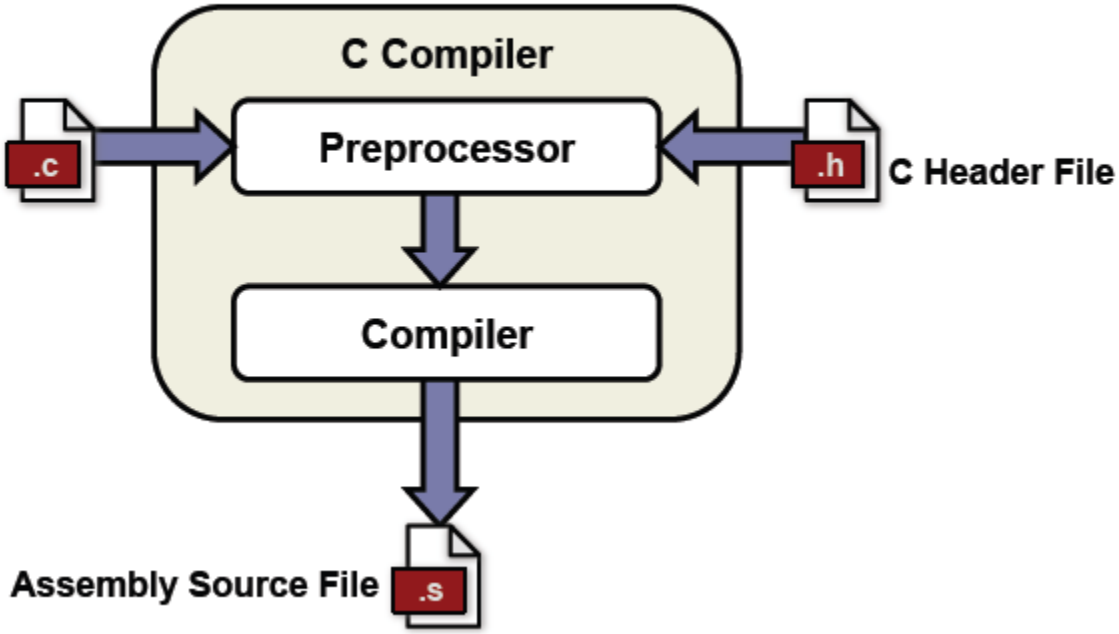




Development Tools Data Flow



Development Tools Data Flow...



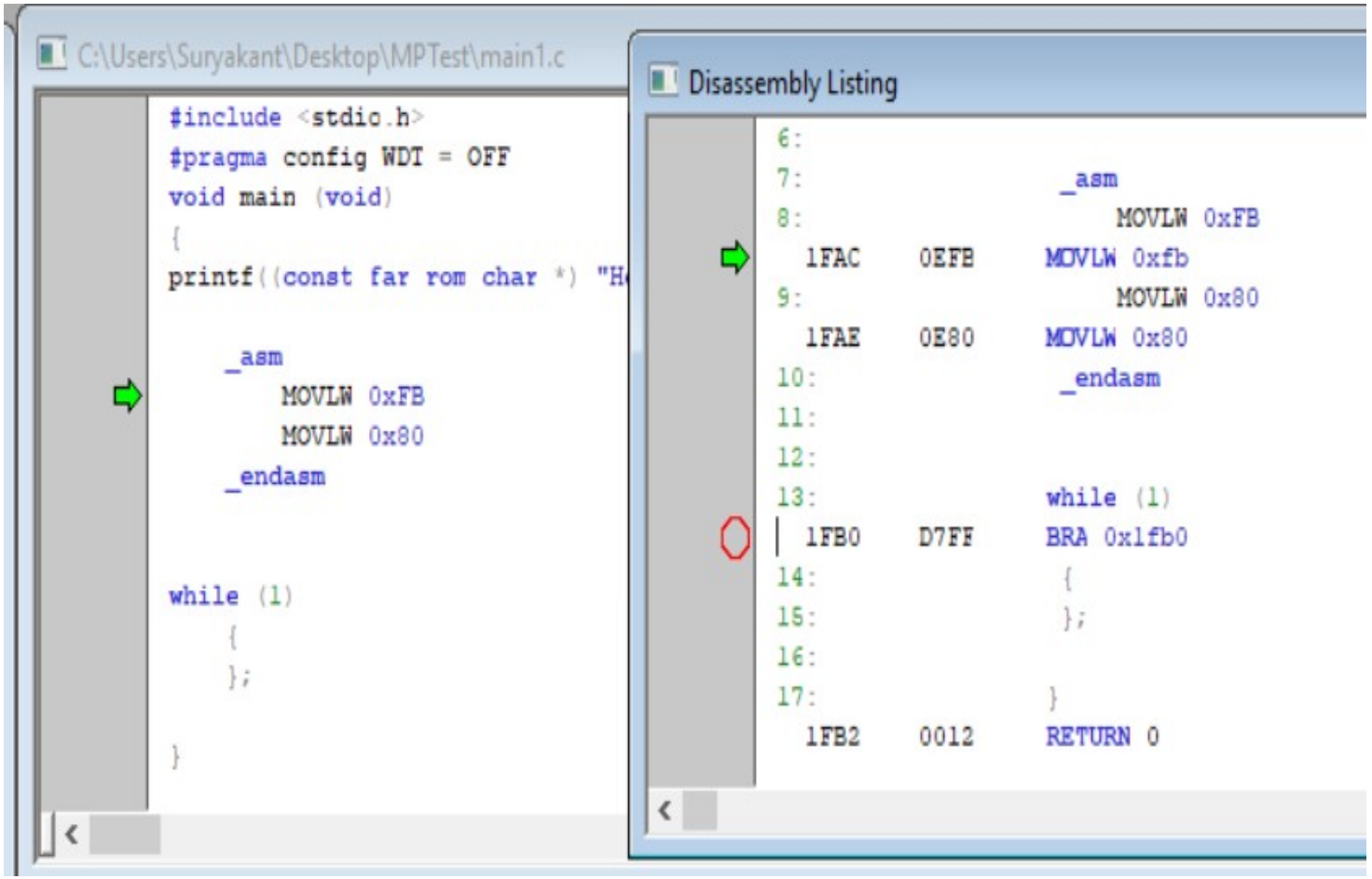
Assembly support in C program

- Mixed language programming using C-language with assembly language is supported by the C18 compiler.
  - Assembly blocks are surrounded with at **_asm** and a **_endasm** directives to the C18 compiler.
  - Assembly code can also be located in a separate **.asm** file.

Data Sizes

- The compiler normally operates on 8-bit bytes (char), 16-bit integers (int), and 32-bit floating-point (float) numbers.
- In the case of the PIC, 8-bit data should be used before resorting to 16-bit data.
  - Floats should only be used when absolutely necessary.

Using Assembly in C Code



C Runtime Environment

C Compiler sets up a runtime environment

- Allocates space for stack
- Initializes stack pointer
- Copies values from Flash/ROM to variables in RAM that were declared with initial values
- Clears uninitialized RAM
- Disables all interrupts
- Calls main() function (where your code starts)

C Runtime Environment

- Runtime environment setup code is automatically linked into application by most PIC® compiler suites
- *Usually comes from either:*
  - crt0.s / crt0.o (crt = C RunTime)
  - startup.asm / startup.o
- User modifiable if absolutely necessary

Compilation

Debug build of project `C:\Users\Suryakant\Desktop\MPTest\T1_4550.mcp' started.

Preprocessor symbol `__DEBUG' is defined.

Tue Jun 17 15:45:08 2019

Clean: Deleting intermediary and output files.

Clean: Done.

Executing: "C:\MCC18\bin\mcc18.exe" -p=18F4550 /i"C:\MCC18\h" "main1.c" -fo="C:\Users\Suryakant\Desktop\MPTest\main1.o" -D__DEBUG -Ou- -Ot- -Ob- -Op- -Or- -Od- -Opa-

Linking

Executing: "C:\MCC18\bin\mplink.exe" /l"C:\MCC18\lib" /k"C:\Users\Suryakant\Desktop\MPTest" "rm18f4550.lkr" "main1.o" /z__MPLAB_BUILD=1 /z__MPLAB_DEBUG=1 /o"C:\Users\Suryakant\Desktop\MPTest\T1_4550.cof" /M"C:\Users\Suryakant\Desktop\MPTest\T1_4550.map" /W

MPLINK 4.1, Linker
Copyright (c) 2006 Microchip Technology Inc.
Errors : 0

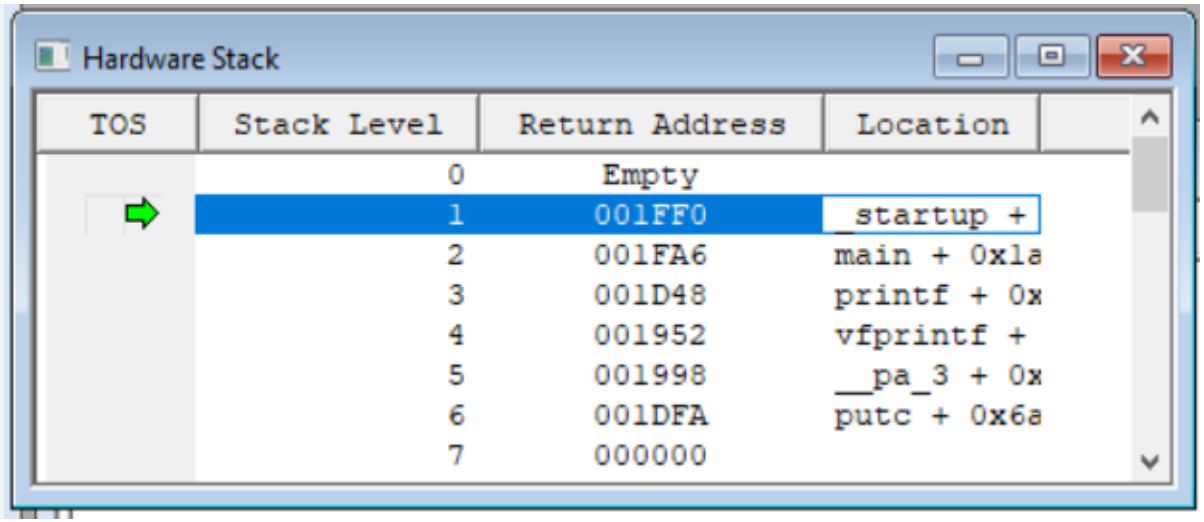
MP2HEX 4.1, COFF to HEX File Converter
Copyright (c) 2006 Microchip Technology Inc.
Errors : 0
Loaded C:\Users\Suryakant\Desktop\MPTest\T1_4550.cof.

Debug build of project `C:\Users\Suryakant\Desktop\MPTest\T1_4550.mcp' succeeded.
Preprocessor symbol `__DEBUG' is defined.
Tue Jun 17 15:45:09 2019

BUILD SUCCEEDED

Debugging Using IDE

- Simulating program execution – MPLAB SIM
  - Controlled Program execution (Single Stepping)
  - Breakpoi
  - Disassem
  - Watches
  - Stack Vie
  - SFRs



Debugging…

- **Debugger > Select Tool > MPLAB SIM**

  Debug toolbar opens
- **Debugger > Step Into → Debugger > Reset > Processor Reset**

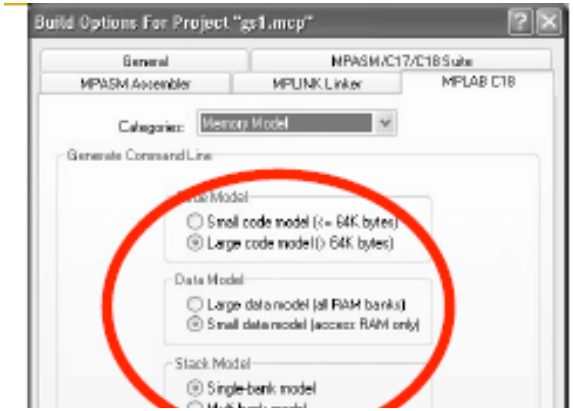
  Assembly code editor opens (disassembly)

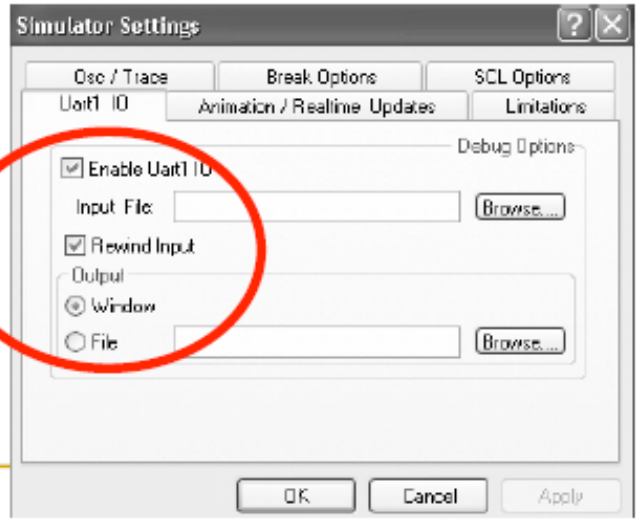
  Green arrow points to program start (main)
- **Step Into**

  Run program in trace mode (single step)

Debugging...

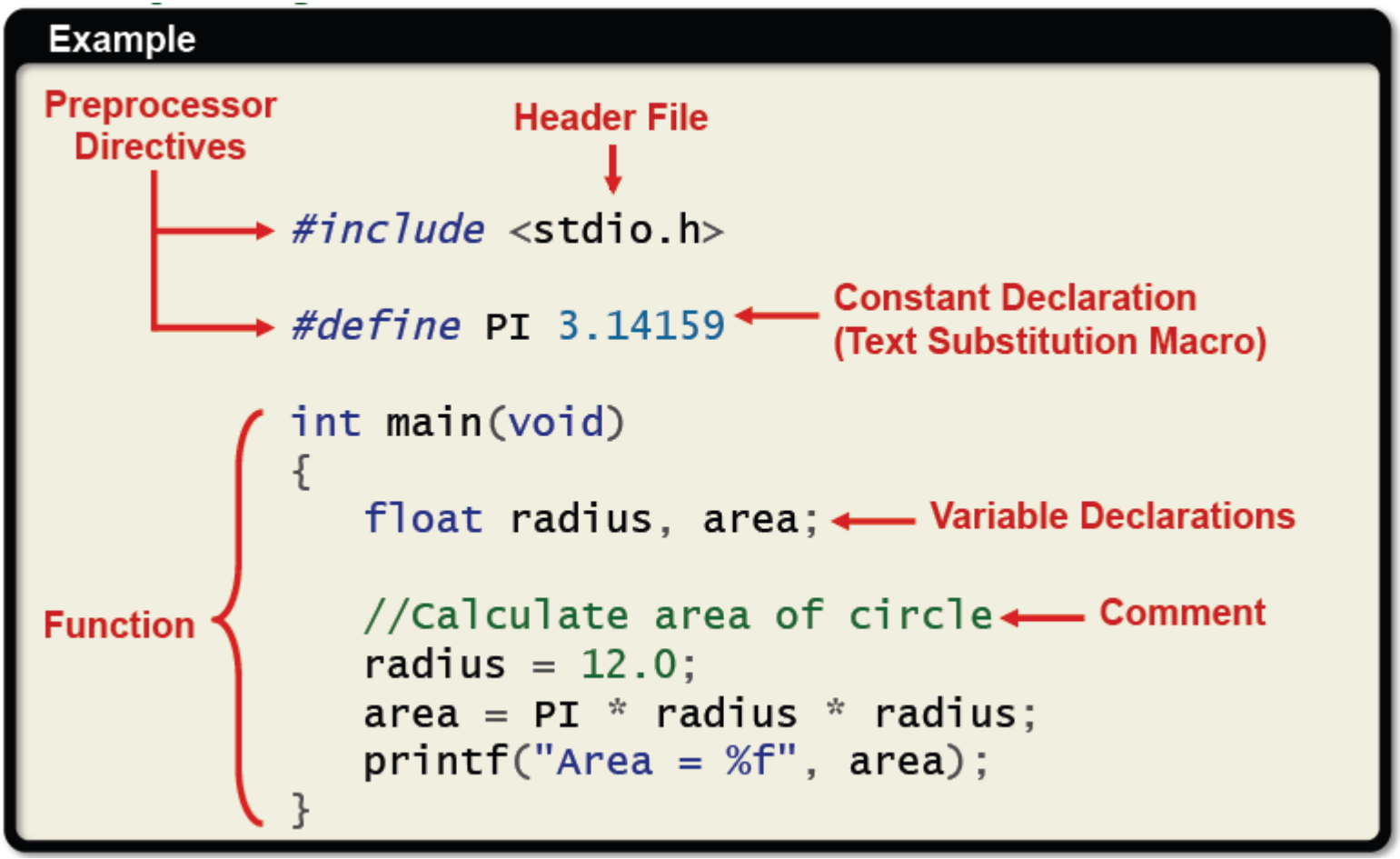
- Make and Build the project
- If you are using standard outputs:
  - Select *Debugger>Settings and*click on the **Uart1 IO tab.**
  - The box marked **Enable Uart1 IO** should be checked.
  - The **Output should** be set to **Window.**
- **Select large code model.**





Basics of C

(A Simple C Program)



Comments

Two kinds of comments may be used:

- Block Comment:- Eg: /* This is a comment */
- Single Line Comment:- Eg: // This is also a comment

```
/*************************************************
* Program: hello.c
* Author:  A good man
*************************************************/
#include <stdio.h>

/* Function: main() */
int main(void)
{
  printf("Hello, world!\n"); /* Display "Hello, world!" */
}
```

Variables and Data Types

(A Simple C Program)

Example

```
#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);
}
```

Data Types → `int` and `float`

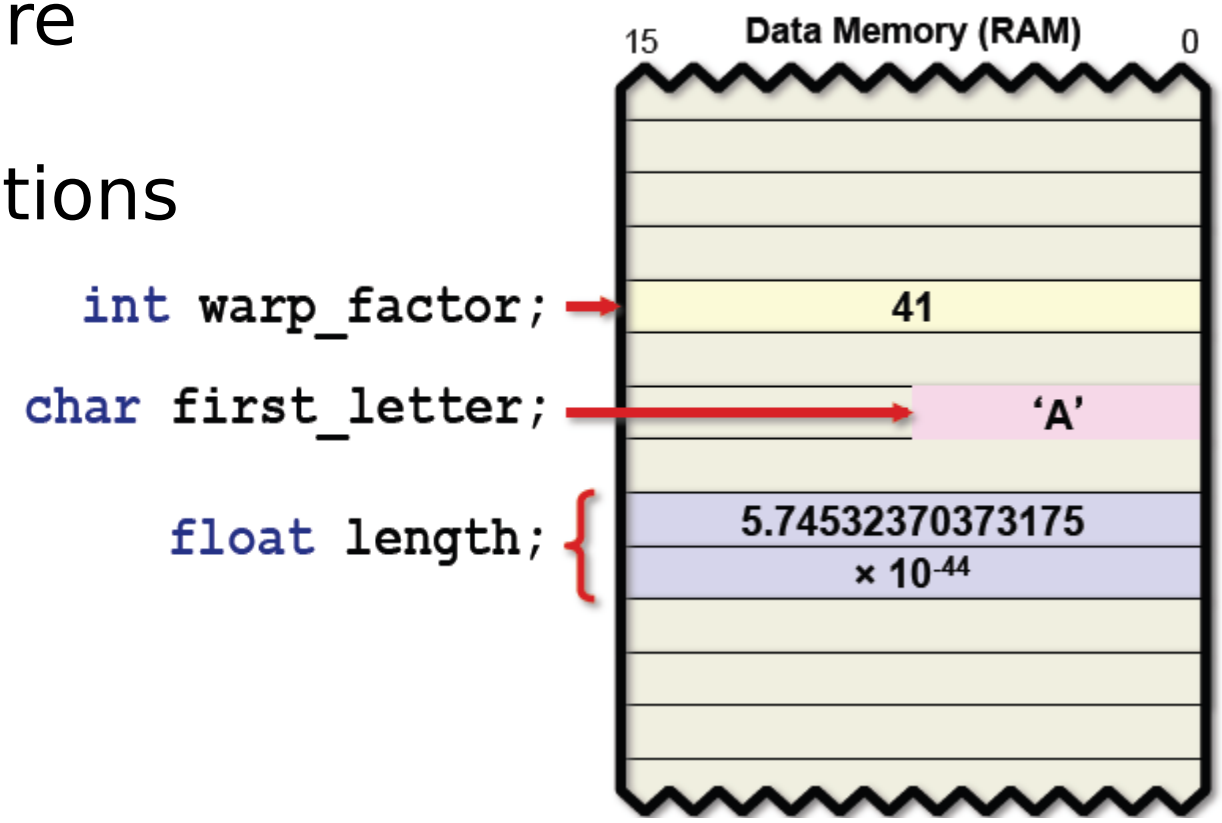
`float radius, area;` ← Variable Declarations

`area = PI * radius * radius;` ← Variables in use

`printf("Area = %f", area);` ← Variables in use

Variables

- **Variables** are names for storage locations in memory
- **Variable declaration consist of a unique identifier (name)**



Data Types

Type	Description	Bits
`char`	single character	8
`int`	integer	16
`float`	single precision floating point number	32
`double`	double precision floating point number	64

The size of an **int varies from compiler to compiler.**

- MPLAB-C30 **int is 16-bits**
- MPLAB-C18 **int is 16-bits**
- CCS PCB, PCM & PCH **int is 8-bits**
- Hi-Tech PICC **int is 16-bits**

Data Types Qualifiers
(Modifying data Types)

Qualifiers: unsigned, signed, short and long

Qualified Type	Min	Max	Bits
`unsigned char`	0	255	8
`char, signed char`	-128	127	8
`unsigned short int`	0	65535	16
`short int, signed short int`	-32768	32767	16
`unsigned int`	0	65535	16
`int, signed int`	-32768	32767	16
`unsigned long int`	0	$2^{32}-1$	32
`long int, signed long int`	-2^{31}	-2^{31}	32
`unsigned long long int`	0	$2^{64}-1$	64
`long long int, signed long long int`	-2^{31}	-2^{31}	64

Embedded C ?

- Additions to C-language for supporting program development for microcontrollers
- Supports
  - Multiple memory maps– Code, Data
  - Special Function Registers (SFRS)
  - Memory mapped IO
  - Interrupt support

Literal Constants

A literal is a constant, but a constant is not a literal

- #define **MAXINT 32767**
- const int **MAXINT = 32767;**
- Constants are labels that represent a literal
- **Literals** are values, often assigned to symbolic constants and variables

Literals or Literal Constants

- Are **"hard coded" values**
- May be numbers, characters, or strings
- May be represented in a number of formats (decimal, hexadecimal, binary, character, etc.)
- Always represent the same value (5 always represents the quantity five)

Constants

- Value that cannot be changed during program execution.
- Stored in the flash program memory
- Saves valuable space of RAM.
- Multiple types
  - Integer/Floats/Character/
  - String

    const char Message[] = "Press the LEFT button";
  - Enumerated constants

    enum Direction {UP, DOWN, LEFT, RIGHT};

Literal Constants

Example

```
unsigned int a;
unsigned int c;
#define b 2

void main(void)
{
   a = 5;
   c = a + b;
   printf("a=%d, b=%d, c=%d\n", a, b, c);
}
```

Literal

Literal

#pragma

- **#pragma** is used to declare directive; **config** directive allows configuring MCU operating modes; device dependent
- Config registers:
  - **#pragma config** WDT = OFF
- Code sections:
  - **#pragma code** _start_on_reset =0x000000
- Interrupt-directive:
  - **#pragma interrupt** *function-name*
  - **#pragma interruptlow** *function-name*

ROM directive

- "**rom**" directive tells the compiler to place data in program memory (*used for Look up tables*)
- Near ROM uses 16-bit address
- Far ROM uses 21-bit address

```
rom near char LUT1[][10]=
        { "Red", "color2", "Blue“ };
```

Data Memory / Registers

- **Data address space**

12 bit address ⇒ memory ≤ 2^{12}

= 4096 bytes = 4 KB

- **Memory partitioned into b**

Bank = 2^8 = 256 = **100h** regis

8 bit file address = **00h … FFh**

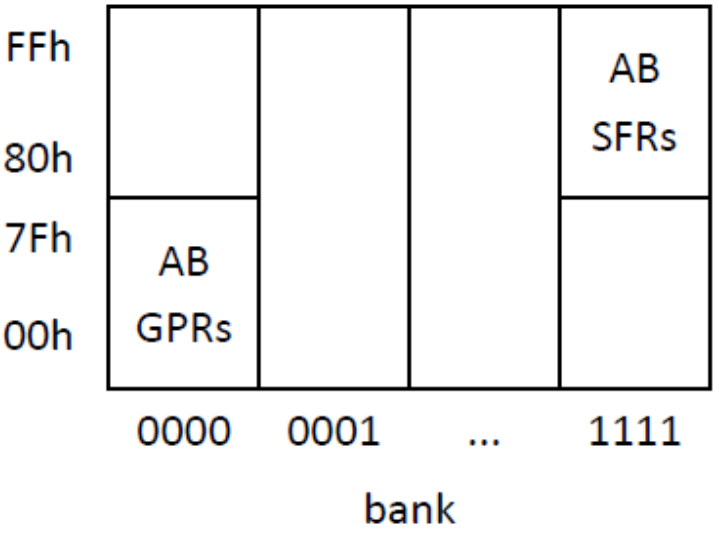
- **Banks in address space**

≤ 2^4 = 16 banks **0 … Fh**

2 to 16 banks implemented in d

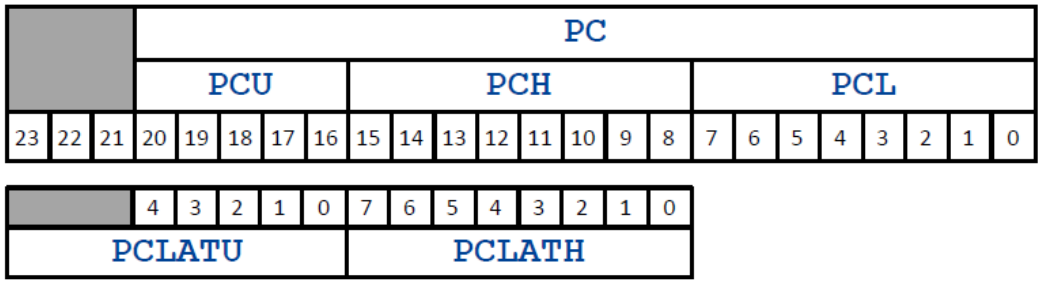
- **Bank select**

Bank Select Register (**BSR**)



Code Memory / Flash

- **Instruction width**
  16 bit instruction = 2 bytes
  PC points to byte (not instruction)
  PC ← PC + 2 on non-branch instruction
- **Address partitioning**
  PC width = 21 bits
  PIC18 has PCU:PCH:PCL
  21 bits = **PCU<20:16>:PCH<15:8>:PCL<7:0>**
  PCU<23:21> not used
- **PC access**
  PCL direct R/W
  PCH ← PCLATH
  PCU ← PCLATU



Programming using Functions

- Sequential list of commands specifying what should be done, can be compared to subroutines.
- Better to have a program consisting of large number of simple functions than of a few large functions.
- A function body usually consists of several commands being executed by the order they are specified.

Functions

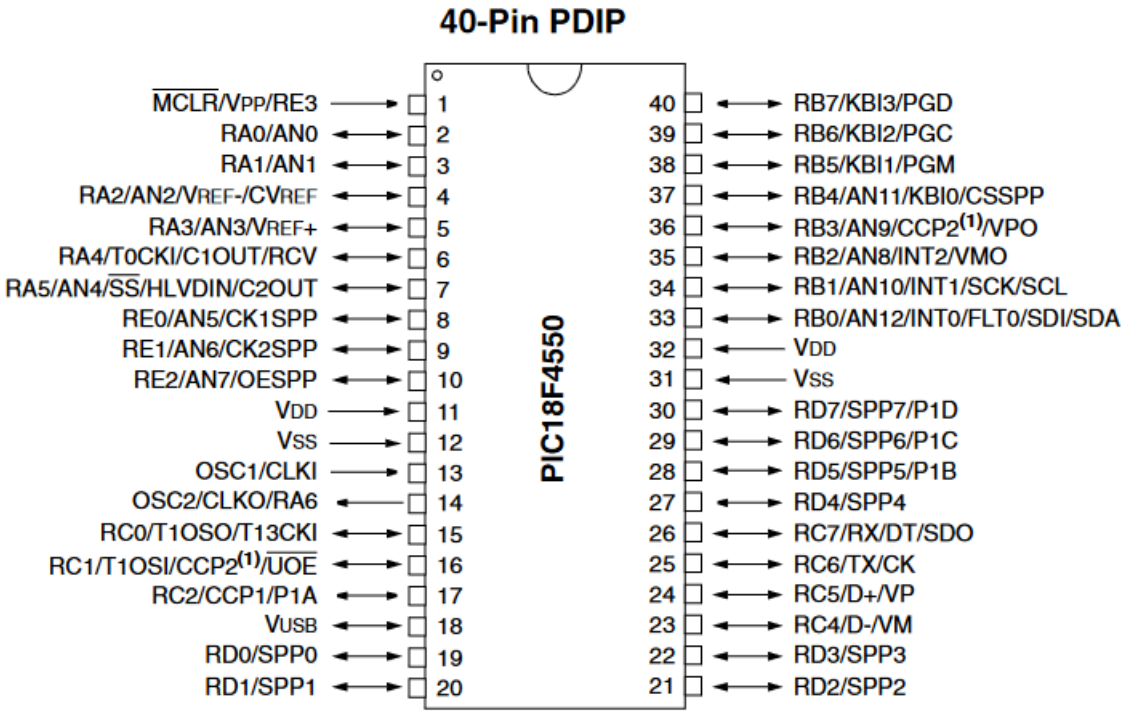
- Every function must be properly declared so as to be properly interpreted during the process of compilation.

  Declaration contains the following elements:

  - Function name
  - Function body
  - List of parameters
  - Declaration of parameters
  - Type of function result

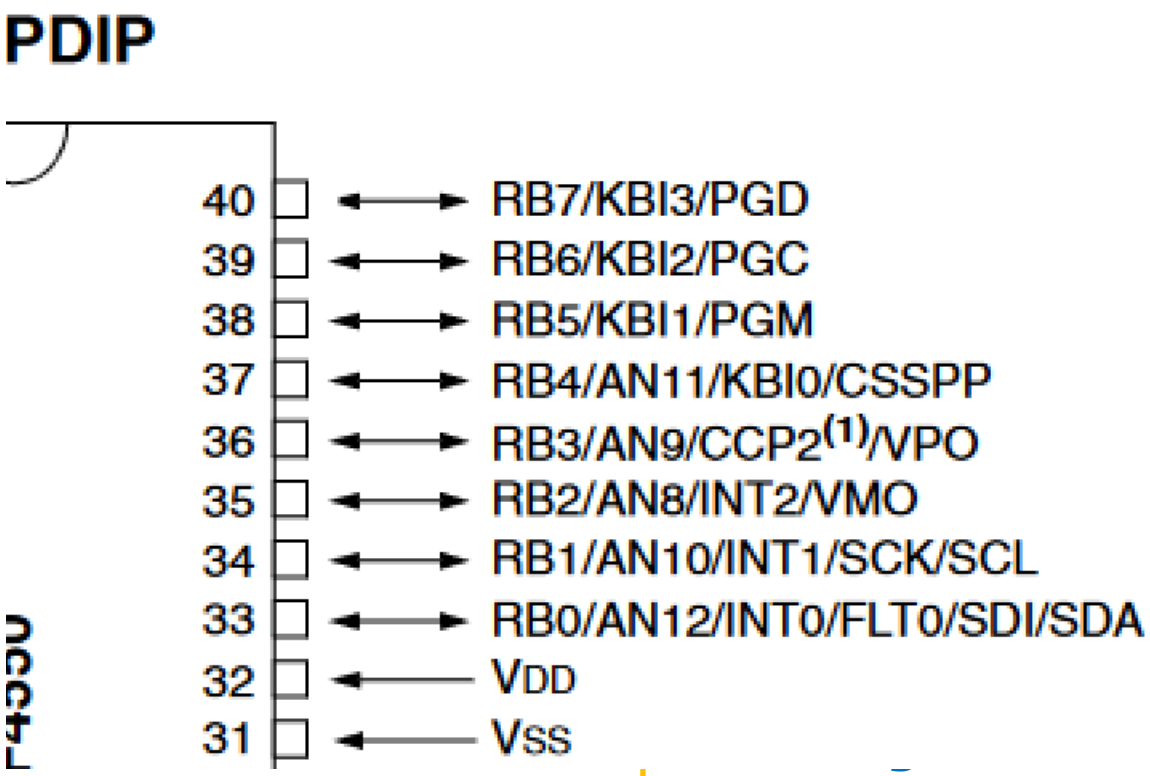
```
type_of_result    function_name(type argument1, type argument2,...)
{
Command;
Command;
...
}
```

PIC18F4550/4455



Port B bits structure (p18f4550.h)

```
extern volatile near union
{
  struct {
    unsigned RB0:1;
    unsigned RB1:1;
    unsigned RB2:1;
    unsigned RB3:1;
    unsigned RB4:1;
    unsigned RB5:1;
    unsigned RB6:1;
    unsigned RB7:1;
  };
  struct {
    unsigned INT0:1;
    unsigned :1;
    unsigned INT2:1;
  };
} PORTBbits;
```



LCD Interfacing

for the PIC18F using C18

LCD Operation

- LCD is finding widespread use replacing LEDs
  - The declining prices of LCD
  - The ability to display numbers, characters, and graphics
  - Incorporation of a refreshing controller into the LCD
    - Relieving the CPU of the task of refreshing the LCD
  - Ease of programming for characters and graphics

LCD Interface

Pin Descriptions for LCD

Pin	Symbol	I/O	Description
1	V_{SS}	--	Ground
2	V_{CC}	--	+5V power supply
3	V_{EE}	--	Power supply to control contrast
4	RS	I	RS = 0 to select command register, RS = 1 to select data register
5	R/W	I	R/W = 0 for write, R/W = 1 for read
6	E	I/O	Enable
7	DB0	I/O	The 8-bit data bus
8	DB1	I/O	The 8-bit data bus
9	DB2	I/O	The 8-bit data bus
10	DB3	I/O	The 8-bit data bus
11	DB4	I/O	The 8-bit data bus
12	DB5	I/O	The 8-bit data bus
13	DB6	I/O	The 8-bit data bus
14	DB7	I/O	The 8-bit data bus

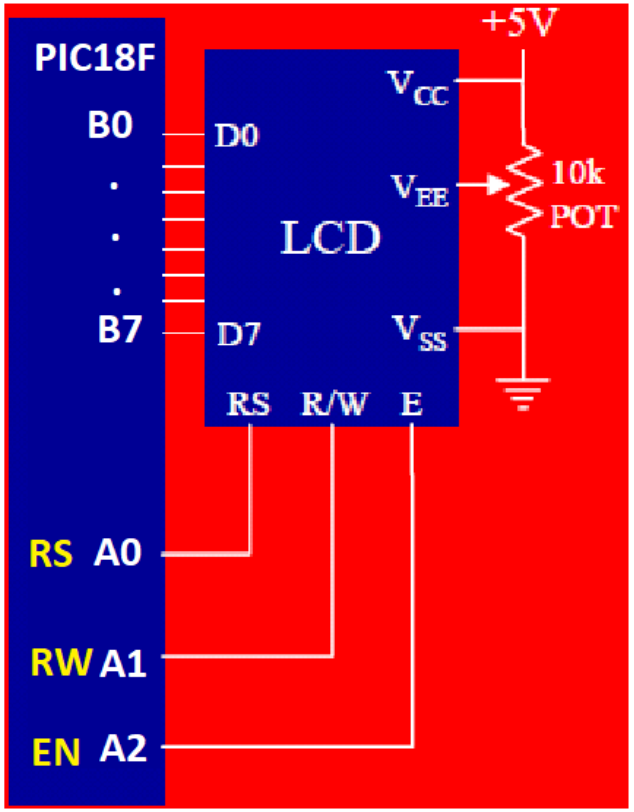
- Send displayed information or instruction command codes to the LCD
- Read the contents of the LCD's internal registers

used by the LCD to latch information presented to its data bus

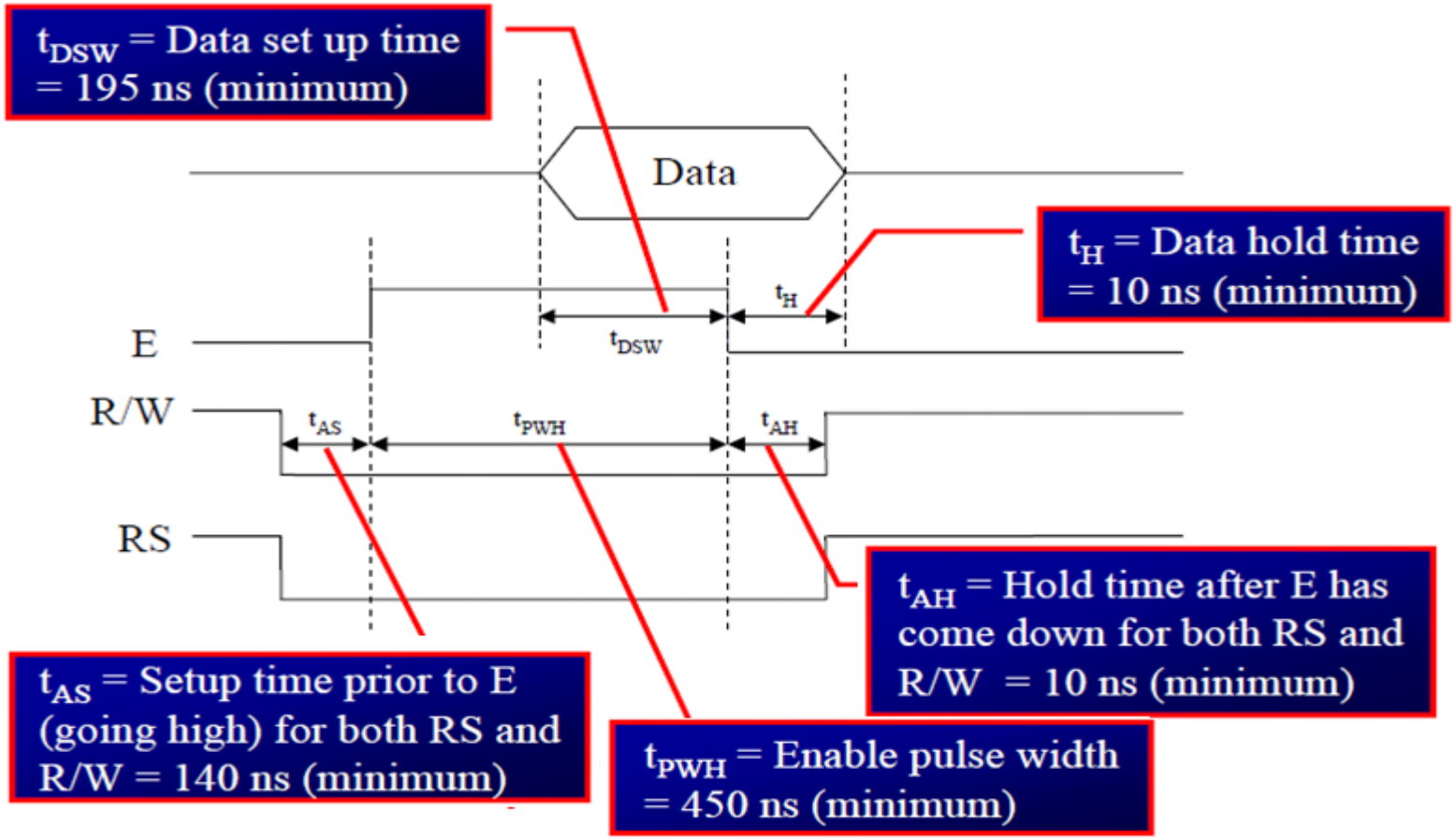
LCD Command Set

LCD Command Codes

Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning of 1st line
C0	Force cursor to beginning of 2nd line
38	2 lines and 5x7 matrix



LCD Timing for Write



LCD Write Operation

- To send
  - Command (From the Command Table) RS=0,
  - Data (ASCII code of Character to be displayed) RS=1
- Setup RW = 0
- Pulse the EN pin high to Low as per the timing for write cycle

LCD Data Write

- To put data at any position on the LCD screen
  - The following shows address locations and how they are accessed
    - AAAAAAA=000_0000 to 010_0111 for line1
    - AAAAAAA=100_0000 to 110_0111 for line2
      - The upper address range can go as high as 0100111 for the 40-character-wide LCD

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A	A	A	A	A	A	A

LCD API

- void LCD_Init(void);
- void LCD_disp(char);
- void LCD_ClearLine(void);
- void LCD_Clear(void);
- void LCD_dispstr(char *);
- void LCD_goto(int,int);

These are the functions exposed to other C files with a prototype in LCD.H file

- void lcd_sendcmd(char);
- void lcd_delay(char);
- void lcd_senddata(char);

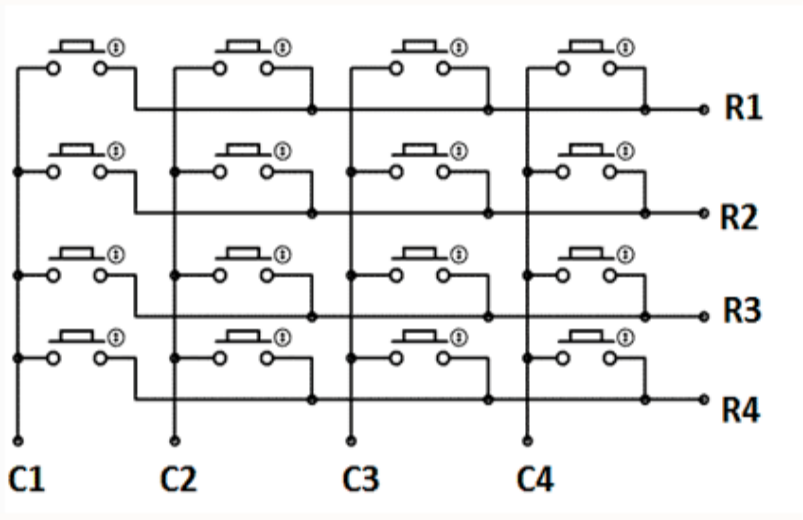
These are support functions to bridge functionality needed for LCD operation, exist only in LCD.C not part of LCD.h file

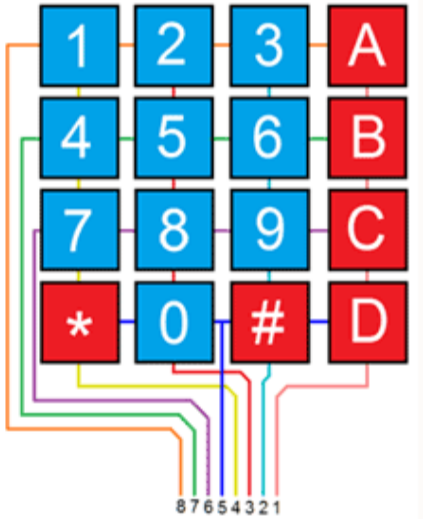
- void lcd_pulse_en(void);
- void lcd_en(char);
- void lcd_rs(char);
- void lcd_rw(char);

These are the functions modelled closely on hardware activity internal to LCD.c file

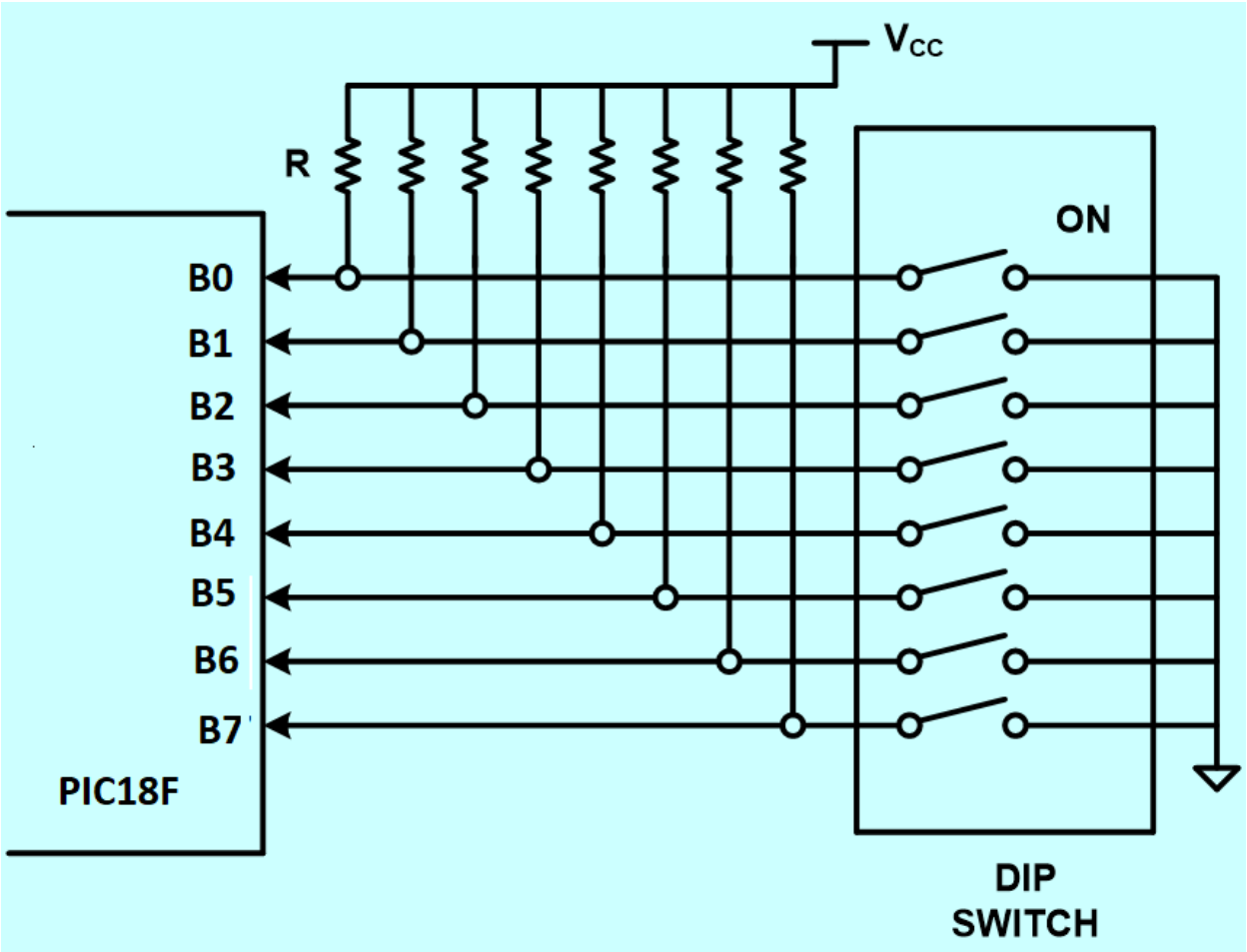
Keyboard Interfacing

for the PIC18F using C18





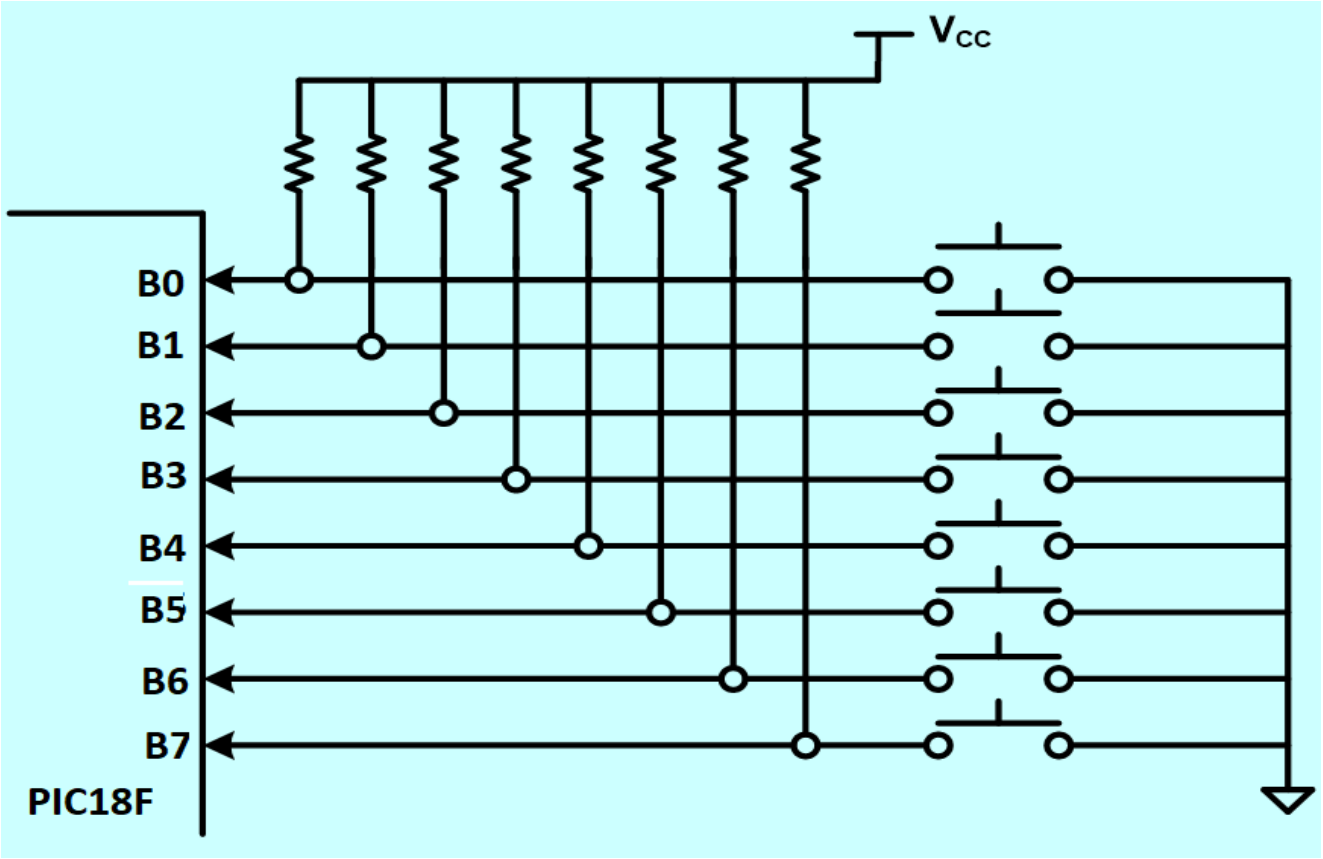
Interfacing Switches



Interfacing Keyboard

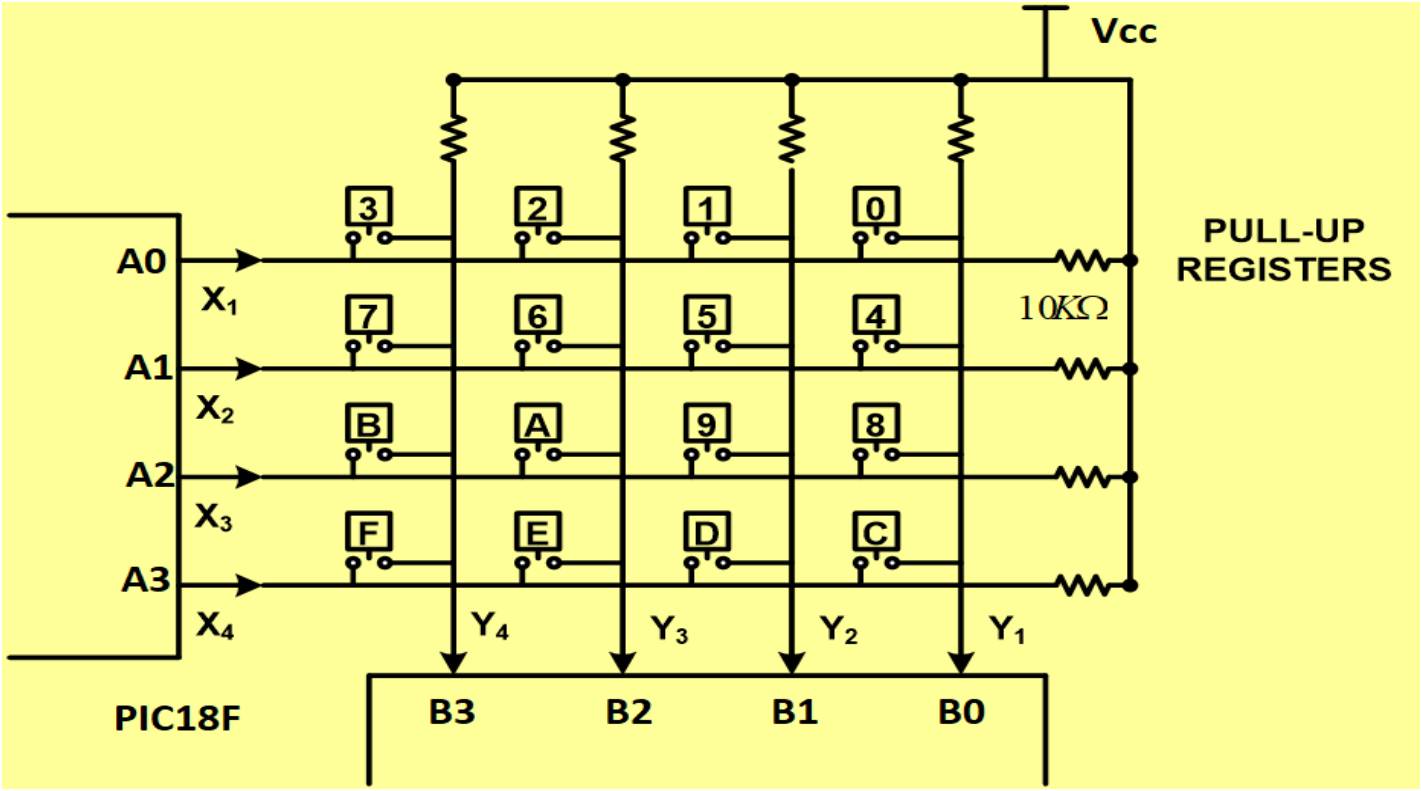
- Collection of keys interfaced to the microcontroller
- Arranged in the form of two dimensional matrix
- Matrix arrangement used for minimizing the number of port lines
- Junction of each row and column forms the key

Interfacing Keyboard...



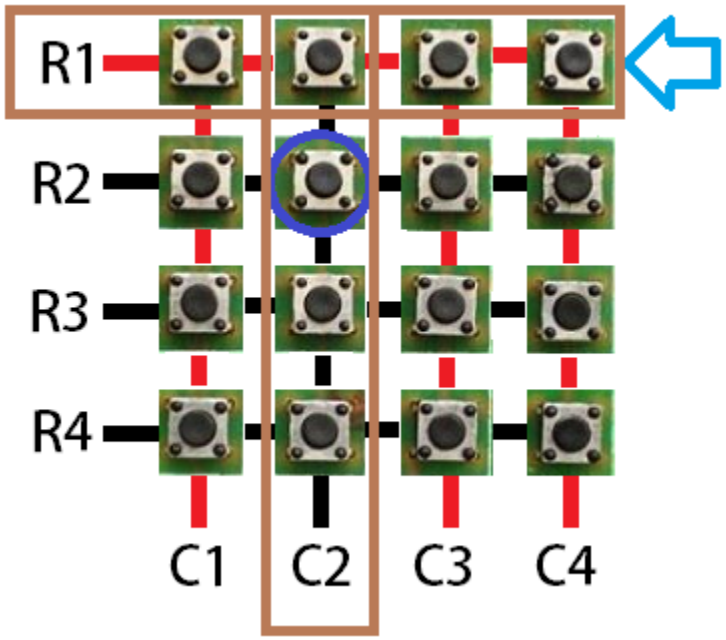
- One key per port line

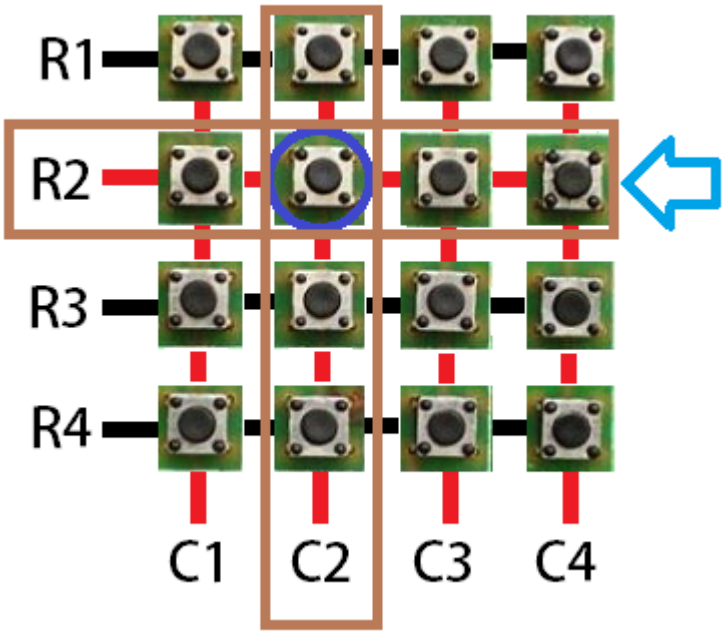
Interfacing Matrix Keyboard



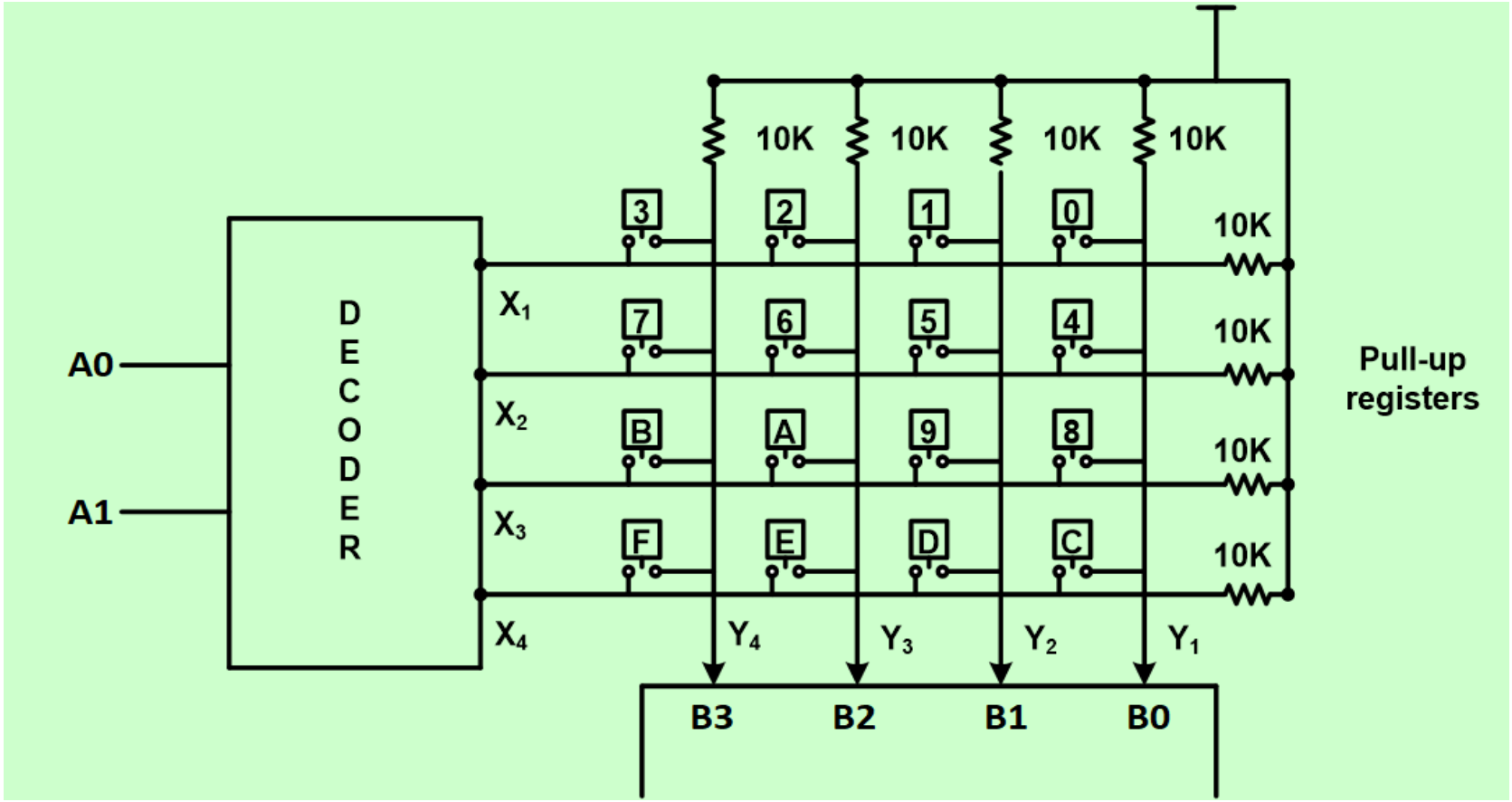
- Keys are organized in two-dimensional matrix to minimize the number of ports required for interfacing

Row Column Scanning





Interfacing Matrix Keyboard



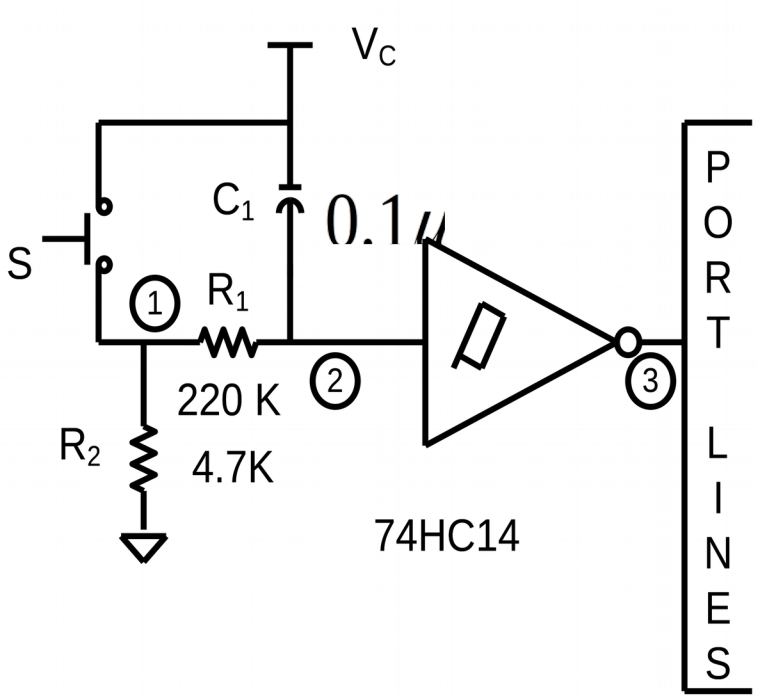
- Use of decoder further reduces the number of port lines required

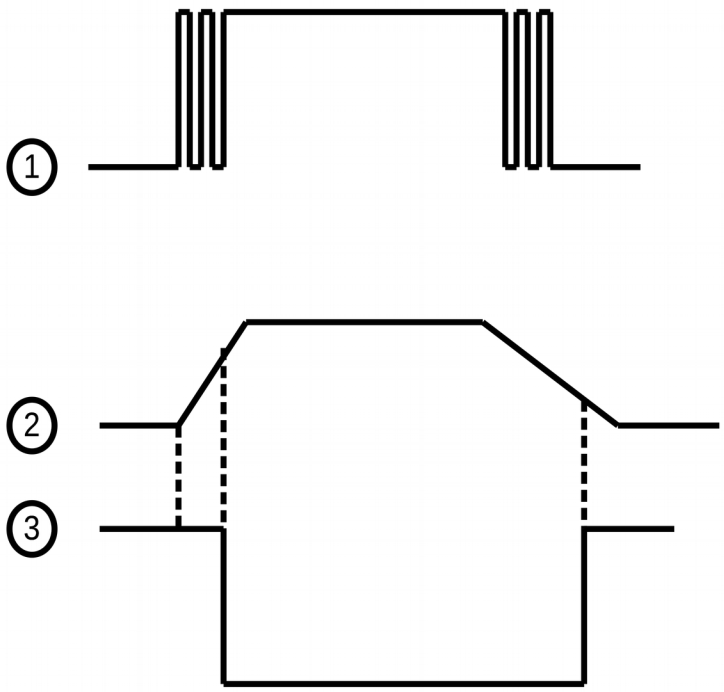
Key Issues in Keyboard Interfacing

- Key bounce can be overcome using Software/Hardware approach
- Keyboard Scanning
- Multiple Key Closure

- Minimize Hardware Requirement:
  - Use of Keyboard Encoder
- Minimize Software Overhead

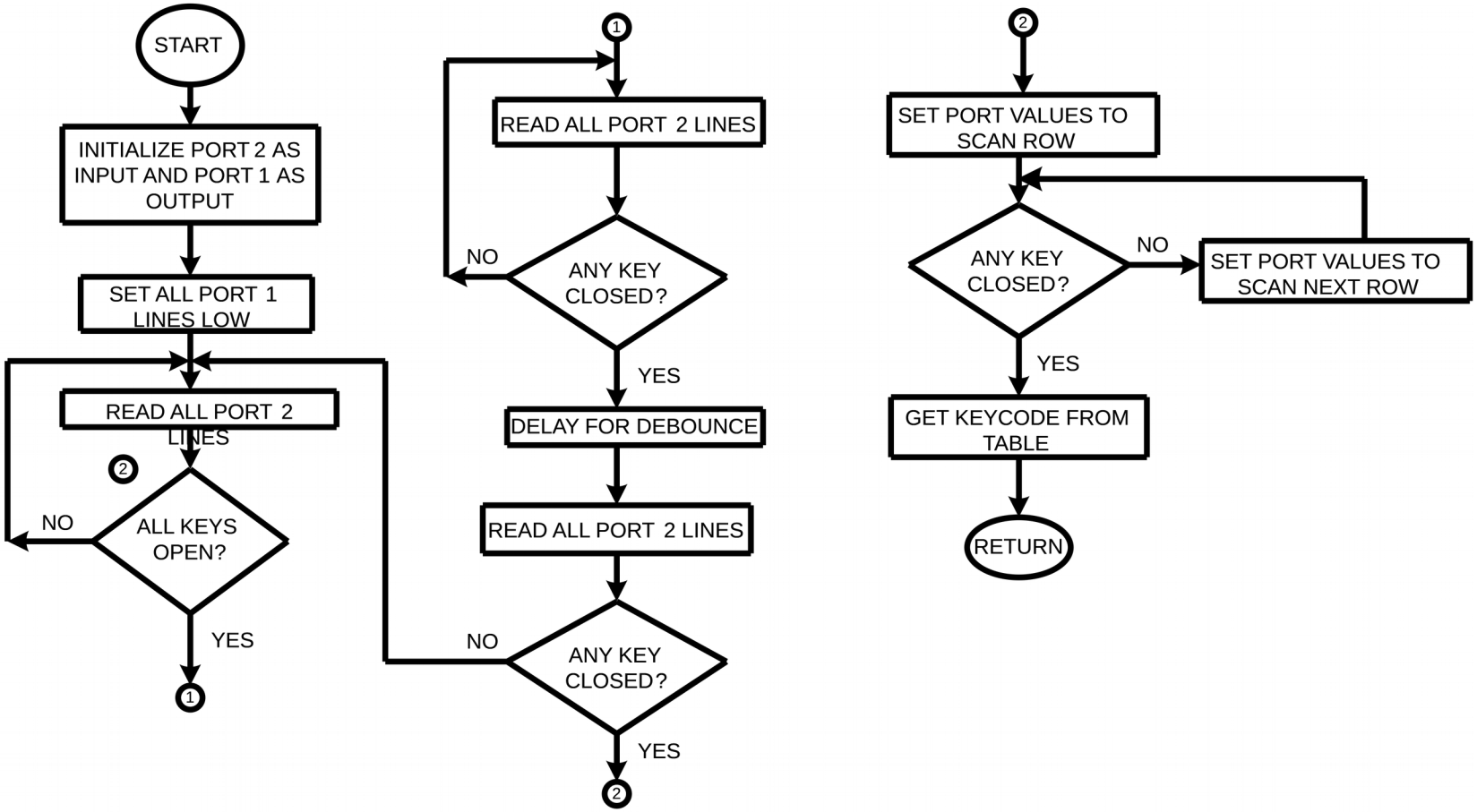
Key Bounce





- Hardware approach to overcome key-bounce

Keyboard Scanning



- Software approach for keyboard scanning

Keyboad Scanning

- **Hardware connections**
  - A0 - A3 connected to rows as output port
  - B0 - B3 connected to columns as input port
- **Top Level Routines**
  - boolean IsKeyHit(void)
  - char GetKey(void)
- **Keyboard API**

boolean IsKeyHit(void)		**char GetKey(void)**	**InitKeyBoard(void)**
void ScanKeys(void)			
void ActivateRow(void)	*char ScanCol(void)*	*DebounceDelay(void)*	*char Lin2Dec(char)*

Thank You