Pearson BTEC Level 5 Higher Nationals in Engineering (RQF)

Unit 46: Embedded Systems

Unit Workbook 3

in a series of 4 for this unit

Learning Outcome 3

Coding, Simulation, Test, Debug

Contents

The Development Cycle

Integrated Development Environment

An Integrated Development Environment (IDE) provides a full suite of tools to a software engineer to facilitate the development of software applications, including code compilation to load onto a microcontroller chip.

The IDE will usually offer a source code editor, which could be written in Basic, C or Assembly Language, automation tools, code debugging features and possibly simulation facilities.

Microchip Technology Inc. offer a desktop-based IDE known as MPLABX and a browser-based IDE called MPLAB Xpress, both of which offer the features mentioned above, plus an abundance of others. These two IDE's will feature prominently in this workbook.

Assembler and High-Level Languages

A high-level language is one in which a programmer may write computer code that is more readily understandable for the human reader. Some examples of high-level languages are given below.

- Python
- C
- C++
- Arduino
- Assembly
- Rust
- C#
- Verilog
- VHDL
- LabView
- Elixir
- Ada
- D
- Forth
- TCL
- LadderLogic
- Erlang

In this workbook we will be considering C as our high-level language of choice, aimed at PIC microcontrollers produced by Microchip Technology Inc.

Assembly Language is as close as you can get to the language of memory chips and software registers (known as Machine Code, consisting of sequences of 1's and 0's). Assembly Language is far more difficult for the inexperienced programmer to understand and is only beneficial (and more efficient) when the reader has a deep understanding of the internal registers/ports available within the microcontroller. In this workbook, we will look at the Assembly Language pertinent to PIC microcontrollers.

Compilers

As previously mentioned, we have High-level Languages and Assembly Language. Whichever language we choose to write code in, a Compiler is a software application that will turn that code into Machine Language to be executed on the microcontroller chip.

Simulators

A simulator will mimic the real-life functionality of the microcontroller chip. Most simulators are proprietary, written by chip manufacturers. Some examples are;

- MPLABX
- PBLAB Xpress
- PICSimLab
- TINA
- gpsim
- Proteus

Debugging

Put simply, debugging is the process employed to find errors or bugs within computer code. Most debuggers allow the programmer to step into, step out of, step over code, or blocks of code, using watch windows to observe the specific effects of those lines or blocks of code on the actual functionality of the program.

Code Structure: Flow Diagrams and Pseudo Code

A Flow Diagram for a basic household refrigerator is shown in Figure 1.

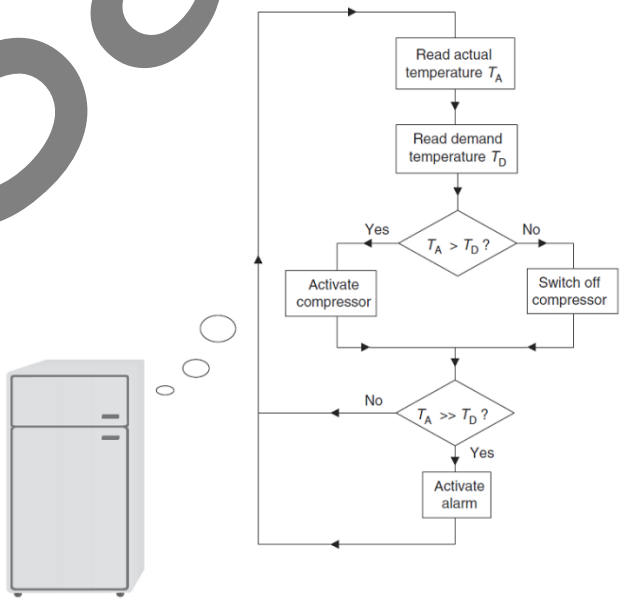


Figure 1: *Flow Diagram for a basic refrigerator*

Here we see that the rectangular boxes contain commands, whereas the diamond shapes are decision elements. Of course, we wish the refrigerator to function continually; hence the code must operate in a continual loop, executing (hopefully) forever or until the power is switched off.

The internal temperature is measured and compared to the desired temperature set by the human operator. Should the measured temperature be greater than the set temperature, the fridge is too warm, so the compressor is activated. Another temperature comparison is then made; if the actual temperature is no longer above the set temperature, then the compressor is de-activated. We then proceed back to the top of the Flow Diagram and repeat the whole process.

The Flow Diagram is useful, as it helps us plan out the actual high-level or Assembler code structure.

Pseudo Code, on the other hand, is a technique whereby no particular programming language is selected, but general ideas about how the code should look/appear are featured. A section of pseudo code is featured in Figure 2.

```
state = 0

while (1)
        read user settings (water temp, etc.) and modify as needed

        if (lid_open = 0)           // lid down
                switch (state)
                        case 0:         // wait for start button

                                if (start_button_pressed)
                                        close drain
                                        turn on Fill lamp
                                        turn on Wash lamp
                                        fill tub        // start
fill
                                        state = 1

                        case 1:         // wait for tub to fill
                                        fill tub            // in case
it was paused
                                if (tub_full)
                                        start timer
                                        turn off Fill lamp  // Wash
still on
                                                start wash cycle
                                        state = 2

                        case 2:         // timed wash cycle
                                        if (timer_paused)   // in case
lid was open
                                                resume timer
                        resume wash cycle
```

Figure 2: *An example of possible pseudo code for a refrigerator*

Review of the C Programming Language

Language Structure

The C programming language is well suited to developing embedded systems with microcontrollers. One of the advanced features of C is its use of Pointers. Pointers allow advanced manipulation of semiconductor registers within the chip, both in terms of their address (location) and contents (1's and 0's). Pointers, and their use, are not covered in this workbook. Still, you are encouraged to investigate their power should you wish to focus more on microcontroller technology as a career objective.

The C language itself is quite succinct, and therefore relatively easy to learn and apply. C is a procedural language, 'procedural', meaning that it facilitates step-by-step instructions very efficiently, as microcontrollers put into practice very well. For example, a microcontroller within a washing machine will be aware of multiple sensors connected to its inputs, such as; weight of clothing load, required temperature, current temperature, drum pressure, actual drum speed, required drum speed, internal humidity etc. In terms of its output pins, the microcontroller controls motors, heaters, fluid valves, timing functions, user displays etc. The human operator may select various wash/dry cycles based upon pre-programmed software algorithms.

The C language can be used in this procedural way extremely efficiently, hence its popularity. C tends to be considered the lowest level high-level language, meaning that it is closer to the digital language of high and low (1 and 0) than other languages – although Assembly Language is always the level directly above Machine Code.

Data Types

The table below shows the data types available in the C programming language.

Data Type	Description
char	Smallest 8-bit character (integer)
signed char	As char, but within the range -127 to +127
unsigned char	Like char, but no sign, 0 to 255 at least
short	A signed integer, -32k to +32k
unsigned short	0 to 65k
int	Integer. -32k to +32k
unsigned	Integer. 0 to 64k
long	Integer. -2 billion to +2 billion
unsigned long	Integer. 0 to 4 billion
long long	Integer. -9 quintillion to +9 quintillion
unsigned long long	Integer. 0 to 18 quintillion
float	Floating point decimal (32 bits max)
double	Floating point decimal (64 bits max)
long double	Floating point decimal (up to 128 bits max)

Program Flow

The C source code produced by a human is first of all sent to a Preprocessor, which converts directives mentioned at the top of the C source code into actual values. Once this is done, the preprocessor will generate a larger version of the C source code.

The expanded C source code is then directed to a Compiler, which essentially converts the C code into Assembly Language.

Compiled code is then sent to an Assembler, which essentially assembles the code and transforms it into Object Code.

The object code is subsequently directed to a Linker, which will link the code to a library of files, including header (*.h) files.

Finally, the linked code is converted into Machine Code (a *.exe file consisting only of 1's and 0's).

Looping

There are three commonly used looping statements used by computer programmers.

while

A 'while' loop will execute a group of statements cyclically, only while a certain condition is true. For example…

while (condition) {

    statements;

}

Each time the 'condition' is true, the statements will execute. Once the last statement in the group has executed, the program goes back and tests for the condition once more. Note here that the condition is tested ***before*** the sequence of statements is executed. This could mean that the sequence of statements are never executed if the 'condition' remains persistently false.

do while

A 'do while' loop will also execute a group of statements, but in this case the group of statements is executed at least once before the 'condition' is tested for. For example…

do {

    statements;

} while (condition);

for

In a 'for' loop we may initialise a variable, then test for a condition, then modify a variable, all before executing a group of statements. As long as the 'condition' is true, then the group of statements will be executed. For example…

```
for (initialisation; condition; modification)
{
statements;
}
```

Branching

Branching out of a sequence of code may be performed in a number of ways, the most popular methods as follows.

if

The 'if' statement can be used with a single statement or a group of statements. For example…

```
if (condition)
{
        statement(s);
}
```

if-else

The 'if-else' statement has two blocks of statements and tests whether a condition is true. If that is the case, then the first block of statements is executed. Should the condition not be true, then the second block of statements is executed. For example…

```
if (condition)
{
        statement(s)_1;
}
else
{
        statement(s)_2;
}
```

switch-case

A switch statement is useful when a number of nested 'if' statements become too complicated and cumbersome. A variable's value is tested, and a number of 'case' statements are used to determine whether there is a match with a series of possibilities and the variable's value. If none of the case tests are true then a 'default' statement(s) is executed. For example…

```
switch (age) {
        case 16:  printf("You are 16");
        case 17:  printf("You are 17");
        case 18:  printf("You are 18");
        case 19:  printf("You are 19");
        case 20:  printf("You are 20");
        case 21:  printf("You are 21");
        default:
                printf("You are not aged between 16 and 21");
}
```

Conditional Testing

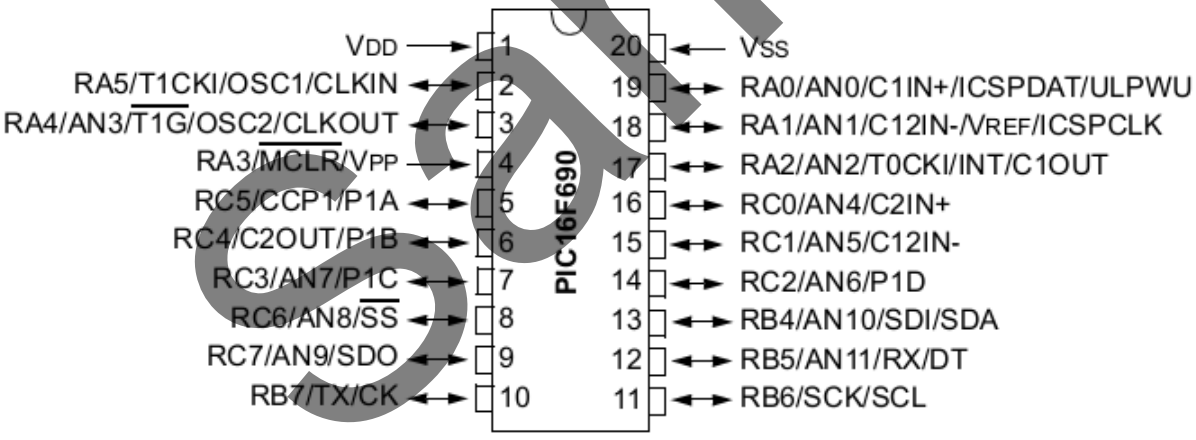
Let's assume that variable A is 0 and variable B is 1. The conditional tests below may be used.

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.

<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Essential Guide - PIC16F690 Microcontroller

Pinout and Block Diagram



Device	Program Memory	Data Memory		I/O	10-bit A/D (ch)	Comparators	Timers 8/16-bit	SSP	ECCP+	EUSART
	Flash (words)	SRAM (bytes)	EEPROM (bytes)							
PIC16F690	4096	256	256	18	12	2	2/1	Yes	Yes	Yes