# MIDDLE EAST TECHNICAL UNIVERSITY
# COMPUTER ENGINEERING DEPARTMENT

## CENG 490

# System Requirement Specification
# And
# Analysis Report

## By
## ResolveSoft

**Project:**

Emulator and Development Environment for CEng Embedded System Card

**Members:**


**Adem HALİMOĞLU**

**Hayri ERDENER**

**Ulaş TUTAK**

**Fatih Mehmet DOĞU**

# 1 INTRODUCTION

## 1.1 Purpose of this document

Purpose of this document is to describe the product that will be produced for senior project course. In this document we are going to try to designate our goals and expectations about our software project, namely DEVEMB. We are going to try to shed light upon our tentative agenda, our future undertakings and details of our intended project.

## 1.2 Scope of this document

This document will derive the requirements for DEVEMB Emulator and Development Environment for PIC Development Board. In this document we defined our functional requirements, interface requirements, performance requirements and our preliminary design and constraints on these. This document also includes our tentative agenda and proposed development methods and environments.

The requirements to DEVEMB as delineated in this report shall be met by the DEVEMB design and implementation. Detailed plans to achieve these goals are to be devised later and described in separate documents. In this paper we did not concentrate on cost and effort estimation. Effort estimation can not change anything since this project must be done in a limited time and also cost estimation is unnecessary since most of the hardware is supplied by the university and there is no one working for salary in our team.

## 1.3 Problem definition

In DEVEMB project, we are going to develop a software development environment, compiler and emulator for a specific embedded system card which is PIC demo board that is designed for CENG 336 Embedded Systems course. By using DEVEMB, users will be able to compile,

debug and test their programs in the virtual card emulated by software. Since, testing and simulation software is not integrated, testing is very difficult. Therefore, DEVEMB will be a software which integrate the simulation and testing.

### 1.4 Business Context

In this project we are developing software which will be used in industry and education. Therefore, our customers will be students, engineers and technicians. Our product has to work correctly and fast. The interface has to be easy to understand and by using our product one should develop his/her program rapidly. Also, our product should include all the necessary tools in order to program 16F877 PICmicro microcontroller.

## 2 GENERAL DESCRIPTIONS

### 2.1 Product Functions

DEVEMB an integrated toolset for the development of embedded applications for Microchip's PIC 16F877 microcontrollers. DEVEMB runs as a 32-bit application on Linux, is easy to use and includes software components for fast application development debugging. DEVEMB serves as a software emulator, compiler and development environment for CEng Card which is used in CEng336 course. Users will be able to compile, upload and debug their programs to the card. Also they will be able to test their programs in the virtual card emulated by software.

### 2.2 Similar System Information

In this section we would like to inform you about some other products functioning similar to our product.

#### 2.2.1 MPLAB IDE

MPLAB is an Integrated Development Environment for Microchip PIC MCU families.
It is a 32 bit Windows application. Like many other IDEs, it consists of a project manager, a text editor, an assembler, and a simulator. Similar to IDEs targeting personal computers, in MPLAB too, user writes programs in the editor, organizes files into projects, compiles assembles-links files into executables and runs them (albeit on a simulator in the case of MPLAB). Debugging is done mainly through watching memory locations, stepping and tracing the code and measuring times spent on various code sections.

MPLAB also may integrate with other products from the developer such as PIC C compilers, ICE emulator, etc.

### 2.2.2 The MPLAB C18 Compiler

It is a free-standing, optimizing ANSI C compiler for the PIC18 PICmicro microcontrollers (MCU). The compiler deviates from the ANSI standard X3.159-1989 only where the standard conflicts with efficient PICmicro MCU support. The compiler is a 32-bit Windows console application and is fully compatible with Microchip's MPLAB IDE, allowing source level debugging with the MPLAB ICE in-circuit emulator, the MPLAB ICD 2 in-circuit debugger or the MPLAB SIM simulator.

### 2.2.3 HI-TECH PICC (1)

HI-TECH PICC is the leading C compiler for the Microchip PICmicro 10/12/14/16/17 series of microcontrollers. HI-TECH PICC makes full use of specific PIC features and using an intelligent optimizer, can generate high-quality code easily rivalling hand-written assembler. Automatic handling of page and bank selection frees the programmer from the trivial details of assembler code. HI-TECH PICC Compiler Features:

- ANSI C - full featured and portable
- Reliable - mature, field-proven technology
- Multiple C optimization levels
- An optimizing assembler
- Full linker, with overlaying of local variables to minimize RAM usage
- Comprehensive C library with all source code provided
- Includes support for 24-bit and 32-bit IEEE floating point and 32-bit long data types
- Mixed C and assembler programming
- Unlimited number of source files
- Listings showing generated assembler
- Compatible - integrates into the MPLAB IDE, MPLAB ICD and most 3rd-party development tools
- Runs on multiple platforms: Windows, Linux, UNIX, Mac OS X, Solaris

### 2.2.4 GPSIM (2) (3)

gpsim is a full-featured software simulator for Microchip PIC microcontrollers distributed under the GNU General Public License (see the COPYING section).
gpsim has been designed to be as accurate as possible. Accuracy includes the entire PIC - from the core to the I/O pins and including ALL of the internal peripherals. Thus it's possible to create stimuli and tie them to the I/O pins and test the PIC the same PIC the same way you would in the real world.
gpsim has been designed to be as fast as possible. Real time simulation speeds of 20Mhz pics are possible.
gpsim has been designed to be as useful as possible. The standard simulation paradigm including breakpoints, single stepping, disassembling, memory inspect & change, and so on has been implemented. In addition, gpsim supports many debugging features that

are only available with in-circuit emulators. For example, a continuous trace buffer tracks every action of the simulator (whether you want it or not). Also, it's possible to set read and write break points on values (e.g. break if a speci_c value is read from or written to a register).
gpsim can be operated either in a command line only mode or in a graphical mode. The command line mode is similar to gdb. Users can set break points and control the simulation. The graphical mode provides a more intuitive debugging interface.
gpsim also supports external modules. Users can extend gpsim for their custom debugging applications.

### **2.2.5** ProgPIC2 - PIC-Programmer-Software (4)

This is a PIC Programmer Program having features:
* For Win95/98/NT/ME/2000/XP
* Programs Flash -Devices
like the PIC16F87x, PIC16F62x
(New PICs can be added in the INI-File)
* In-system programming
* Connects via parallel- or com-port, many circuits available
* Low voltage programming possible
* Write , readback and verify Flash- and EEProm-Memory
* Loads files with Intel Format

### 2.3 user characteristics

The user of our product will be programmers. The users want to develop their program rapidly and they also want to test and debug their source code without losing much time. Therefore, our program should work fast and include all the necessary tools for PIC 16F877 microcontroller.

### 2.4 User Problem Statement

For a programmer, working with an assembly language is almost painful and by this method sophisticated projects are hard to realize. Moreover assembly programs, being succinct and cryptic, are not very accessible. Beside these, by copying the program to the hardware for every testing is a waste of time. The development of on-chip debuggers, and on-chip BASIC interpreters by an industry in which the profit margins are already slim, is a strong indicator of a need in this field.

MPLAB is very widely used by PIC developers because its freeware and produced and distributed by the developer of the PIC MCU and thus accepted to be the most standard development tool. It also supports the widest variety of PIC families. However, it does not run on Unix-like operating systems, to which category most free operating systems belong.

Beside these, it is not open source, thus not suitable for adapting to other uses and does not lend itself readily to interfacing with other applications. Moreover, trying to determine how a peripheral (e.g. an LCD) will behave in certain situations by tracing the code and watching numerical values are very cumbersome to say the least. Also, simulating more sophisticated features of PIC MCU, such as analog/digital conversion or serial communication, in MPLAB is either very hard or just totally impossible.

By using DEVEMB, developer will be able to compile and debug their programs and test their programs in the virtual card emulated by software and upload them to the PDB, quickly. The products told in above part used for development for PIC micro-controller units in the market, compilers, debuggers and simulators. However, testing is cumbersome in those software, because development environment and simulation software is not integrated. For instance, when using MPLAB IDE, one can not examine the output of LCD, whereas, when simulating a program on ISIS, basic debugging facilities such as tracing are not available to the programmer. Likewise, each of the other products suffer some deficiency, be it limited capabilities, limited licenses or high prices. Being compatible with Unix OS and containing all necessary tolls for PIC 16F877 programming, DEVEMB will be a solution to those problems.

### 2.5 User Objectives

Our system have to be:

- Easily installed
- Easily understood
- Easy to use
- Precise

### 2.6 General Constraints

Our system would need to be:

- Real-time
- Portable
- Quite fast

## 3 TEAM ORGANIZATION

The team organization selected for the project is controlled decentralized with Hayri ERDENER being the project leader. This type of an organization is selected since our project is related with various concepts and these concepts are partially new to us so vast researches are needed. The project leader assigns specific search areas to the members and communications which are made are both horizontal and vertical. All decisions are taken with the opinions of group members and since the team consists of four members all decisions can

be taken without ambiguity. In addition to these all members have the same rights for a decision to agree or disagree.

# 4 LITERATURE SURVEY

## 4.1 GENERAL INFORMATION ABOUT PIC MICROCONTROLLER

Integrated circuits development has made it possible to store hundreds of thousands of transistors into one chip. This provided the production of microprocessors. The first computers were constructed by adding external peripherals such as memory, input-output lines, timers and other. Increasing the volume of the package resulted in creation of integrated circuits. These integrated circuits contained both processor and peripherals. This is how the first microcontroller invented.

PIC is a family of **RISC** microcontrollers made by Microchip Technology, derived from the PIC1650 originally developed by General Instrument's Microelectronics Division. The original PIC was built to be used with General Instrument's new 16-bit CPU, the CP1600. While generally a good CPU, the CP1600 had poor I/O performance, and the 8-bit PIC was developed in 1975 to improve performance of the overall system by offloading I/O tasks from the CPU. The PIC used simple microcode stored in ROM to perform its tasks, and it is a RISC design that runs one instruction per cycle (4 oscillator cycles).

In 1985 the PIC was upgraded with EPROM to produce a programmable channel controller, and today a huge variety of PICs are available with various on-board peripherals (serial communication modules, UARTs, motor control kernels, etc.) and program memory from 512 words to 32k words.

## 4.2 Microcontrollers versus Microprocessors

A microcontroller a microprocessor is not the same. The most important difference is in functionality. In order to use a microprocessor, other components such as memory, or divisions for receiving and sending data must be added. Microprocessor is the heart of the computer but microcontroller is designed to be all of those components in one. No other external components are needed for an application of microcontroller because all necessary peripherals are already embedded into it. Thus, time and space needed to construct those devices are saved.

### 4.3 Units of PIC 16F877

PIC 16F877 Microcontroller has the following units:

#### 4.3.1 Memory unit

Memory is the unit whose function is storing data. I can be simply thought as a chiffonier**.** The drawers can be thought as memory locations and we can think that each drawer is marked so that we know the adresses, and any of their contents will then be easily accessible.

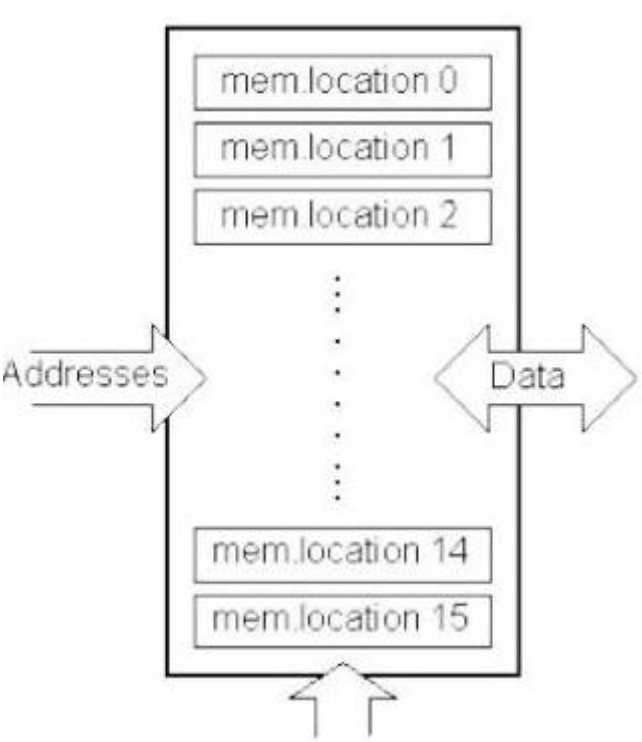



Figure 4.1 R/W (Determines whether the data is read or written)

Memory components are exactly like in the figureXXX.1. For a certain input we get the contents of a certain addressed memory location. Addressing and memory location are two concepts in those operations. Memory is the collection of all memory locations, and addressing is just selecting one of them. “This means that we need to select the desired memory location on one hand, and on the other hand we need to wait for the contents of that location. Beside reading from a memory location, memory must also provide for writing onto it.” This is done by supplying an additional line called control line. We will symbolize this line as R/W (read/write). For a control line is:

if r/w=1, reading is done.
if r/w=0 then writing is done on the memory location.

#### 4.3.2 Central Processing Unit

Central Processing Unit (CPU) is the unit that interprets instructions and processes data contained in computer programs. Its memory locations are called registers.

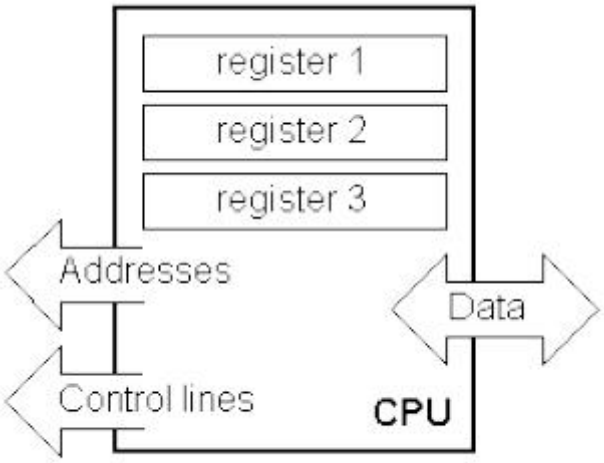



Figure 4.2

The role of the register is to help in performing various operations. It is a temprorary storage for the operands. Memory and CPU are interconnected, and any exchange of data is hindered, as well as its functionality. If we wish to add the contents of two memory locations and return the result to memory, we would need a bus between CPU and memory.

### 4.3.3 Bus

A bus is a group of wires. There are two types of buses, namely address and data bus. The data bus consists of as many lines as needed to adress the whole memory. The data bus is as wide as data. Function of the address bus is to transmit address from CPU memory, and the function of the data bus is to connect all blocks inside the microcontroller.

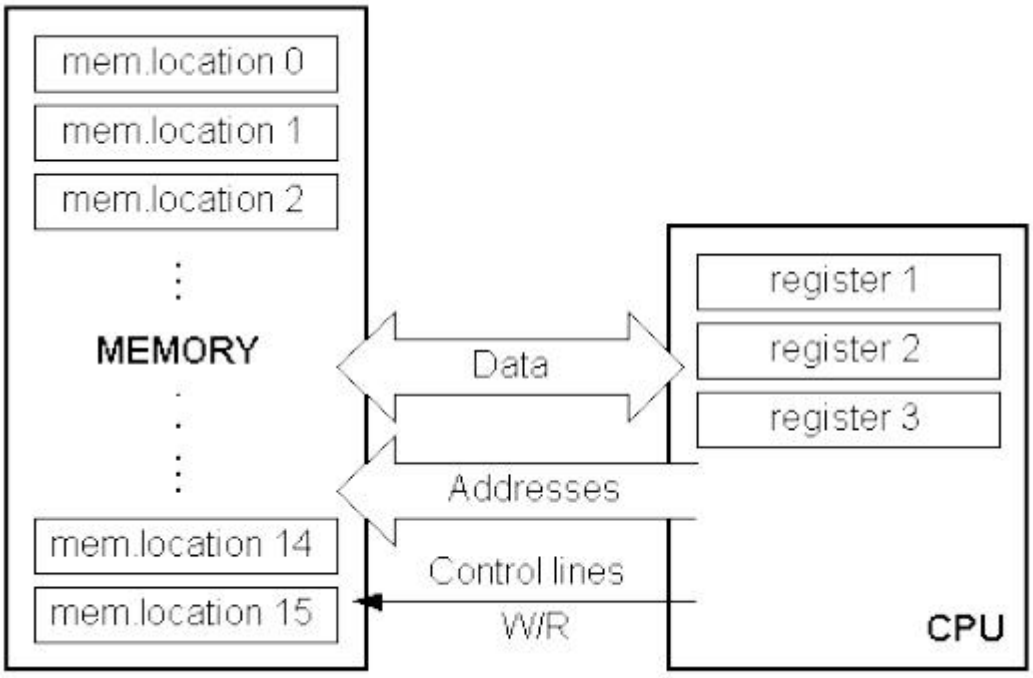



Figure 4.3

We have a unit which is capable of working by itself and to communicate with this device we need to add a block which contains several memory locations whose one end is connected to the data bus, and the other end has been connected to the output lines on the microcontroller. Those locations are called ports.

### 4.3.4 Input-output unit

There are different types of ports, namely, input, output or bidirectional ports. When working with ports, first, we have to choose which port we need to work with, and then communicate from the port.

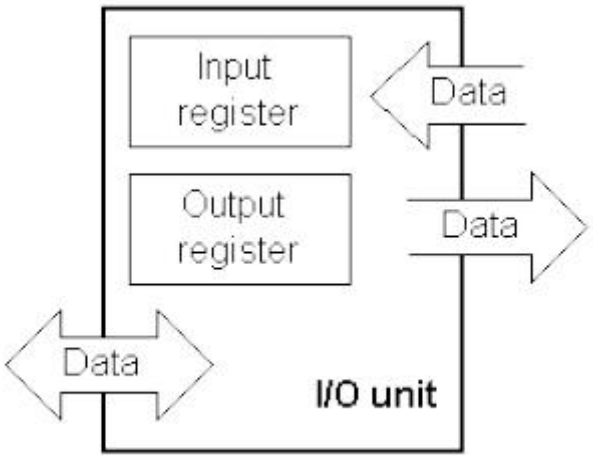



Figure 4.4

When working with a port, it acts like a memory location. Anything is being written into it or read from it, and it could be noticed on the pins of the microcontroller.

### 4.3.5 Serial communication

The way of communicating told above has its drawbacks. One of the vital drawbacks is the number of lines which transfer data. Also, one should consider the transfer of data to a distant location? One need an economic solution for sending wide data to a distant place. Without decreasing the functionality the number of lines should be reduced. Suppose we are performing operations with three lines only, and one of the lines is used for sending data, the other for receiving data, and the third one is used as a reference line for both the input and the output side. In order to work as in this method one have to set the rules of data exchange. These rules are called **protocol**. Protocol must be defined before. Therefore, any misunderstanding between the sides that are communicating with each other would not arise.

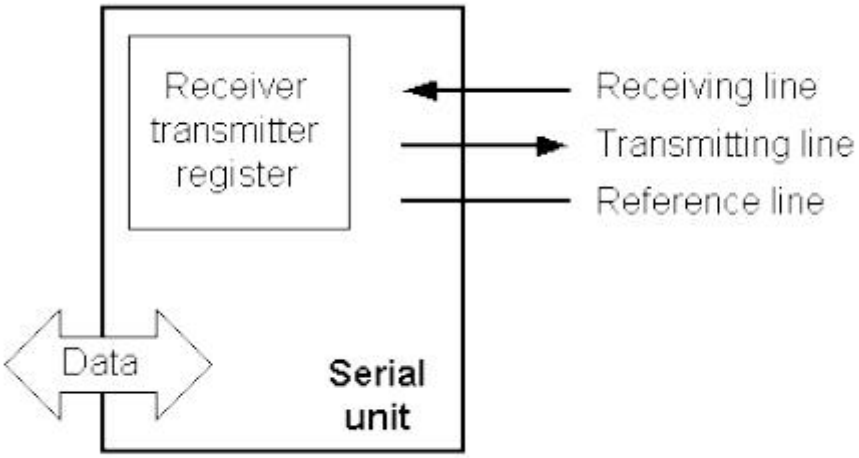



Figure 4.5

As we have separate lines for receiving and sending data, it is possible to receive and send data at the same time. This way of communication is called a serial communication block. Unlike the parallel transmission, data moves, in this method, bit by bit, or in a series of bits what defines the term serial communication comes from. After the reception of data we need to read it from the receiving location and store it in memory as opposed to sending where the process is reversed. Data goes from memory through the bus to the sending location, and then to the receiving unit according to the protocol.

### 4.3.6 Timer unit

Since we have the serial communication explained, we can receive, send and process data with the help of the timer block. Timer unit is significant since it gives information about time, duration, protocol etc. The basic unit of the timer is a counter which is in fact a register whose numeric value increments by one in even intervals freely, so that by taking its value during periods T1 and T2 and calculating their difference we can determine how much time has passed. This is a very vital part of the microcontroller.

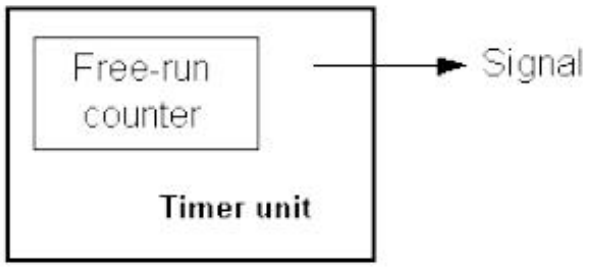



Figure 4.6

### 4.3.7 Watchdog

“One more thing is requiring our attention is a flawless functioning of the microcontroller during its run-time.” Assume that, as a result of some interference,which often does occur in industry, our microcontroller stops executing the program, or it starts working incorrectly. When this happens with a computer, one reset it and it will keep working again. However, there is no reset option for microcontroller. To find solution to this problem, we need to a block called watchdog. In fact, this block is another counter. In case that program gets work abnormally the counter alone will reset the microcontroller upon achieving its maximum value. This will result in reexecuting the program, and correctly this time. That is an important element of every program to be reliable without interference of human.


Figure 4.7

### 4.3.8 Analog to Digital Converter

As the peripheral signals usually are substantially different from the ones that microcontroller can understand (zero and one), they have to be converted into a pattern which can be understood by a microcontroller. “This task is performed by a block for analog to digital conversion ( ADC ). This block is responsible for converting an information about some analog value to a binary number and for follow it through to a CPU block so that CPU block can further process it.”

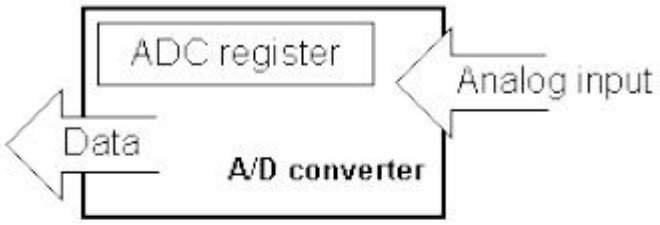



Figure 4.8

The picture below shows what a microcontroller looks like:

Figure 4.9

### 4.4 Coding For PICs

PICs use a RISC instruction set, which varies in length from about 35 instructions for the low end PICs to about 70 instructions for the high-end PICs. The instruction set includes instructions to perform a variety of operations on the accumulator and a constant or the accumulator and a memory location, as well as for conditionally executing code and jumping or calling other parts of the program and returning from them, and specific hardware features like interrupts and one low-power mode called sleep. Microchip provides a freeware IDE package called **MPLAB**, that also includes a software **simulator** as well as an **assembler**.

Third parties make **C** and **BASIC** language **compiler**s for PICs. Microchip also sells compilers for the high-end PICs ("C18" for the 18F series and "C30" for the dsPICs). They also make available for download a "student edition/demo" version of C18 or C30 which disables some optimiser features after a timeout period. Open-source compilers for the C, Pascal, JAL, BASIC, and for the Forth programming language, PicForth, have also been released.

**GPUTILS** is an Open Source collection of tools, distributed under the **GNU** General Public License. GPUTILS includes an assembler and **linker** and works on **Linux**, Mac OS X, OS/2 and Microsoft Windows. GPSIM is an **Open Source** simulator for the **PIC microcontrollers** featuring hardware modules that simulate specific devices that might be connected to them, like LCDs.

### 4.5 Programming PICs

Devices called "programmers" are traditionally used to get program code into the target PIC. Most PICs that Microchip sells nowadays have **ICSP (In Circuit Serial Programming)** and/or **LVP (Low Voltage Programming)** capabilities, allowing the PIC to be programmed while it is sitting in the target circuit. ICSP programming is performed using the RB6 and RB7 pins for clock and data, while a high voltage (12V) is present on the Vpp/MCLR pin. Low voltage programming allows for the elimination of the extra voltage rail in the programmer but comes at the cost of an IO pin and can therefore be disabled (once disabled it can only be re-enabled using high voltage programming). There are many programmers for PIC microcontrollers, ranging from the extremely simple designs that rely on the communications software for taking care of all the communication details to complex designs that can verify the device at several supply voltages and can do much of the work in the hardware. Many of these complex programmers use a pre-programmed PIC themselves to send the programming commands to the PIC that is to be programmed.

Many of the higher end flash based PICs can also write to their own program memory. Demo boards are available with a small bootloader factory programmed that can be used to load user programs over an interface such as **RS-232** or **USB**.

### 4.6 RS232 Protocol

RS232 is an asynchronous serial communications protocol, widely used on computers. Asynchronous means it doesn't have any separate synchronizing clock signal, so it has to synchronize itself to the incoming data; it does this by the use of 'START' and 'STOP' signals. The signal itself is slightly unusual for computers, as rather than the normal 0V to 5V range, it uses +12V to -12V. Current usage of RS232 states that, data is transmitted in groups or characters of 7 or 8 bits. Each character is preceded by a start bit that must be 0 and is followed by at least one stop bit that must be a one. And also we have parity bit and it is optional. The parity may be odd, even or may not be present.

## 4.7 PIC16F877 DEVICE OVERVIEW (5)

. 28/40/44 pin packages
. 14bit core - 35 instructions
. 200ns instruction time (Tclk = 20MHz)
. 8,092 14bit FLASH program memory
. 368 8bit data memory or registers (“File registers”)
. 256 8bit EEPROM (nonvolatile) data registers
. 8 level hardware stack
. Interrupt capability (up to 14 sources)
. 33 pin I/O (for 40 pin package)
. 3 Timer/Counter modules
    . Timer0: 8-bit
    . Timer1: 16-bit
    . Timer2: 8-bit
. Two Capture, Compare, PWM modules
    . Capture: 16-bit
    . Compare: 16-bit
    . PWM: max. resolution is 10-bit
. 10-bit 8 channel Analog-to-Digital Converter
. Synchronous Serial Port (SSP) with SPI and I2C
. Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
. Parallel Slave Port (PSP) 8-bit

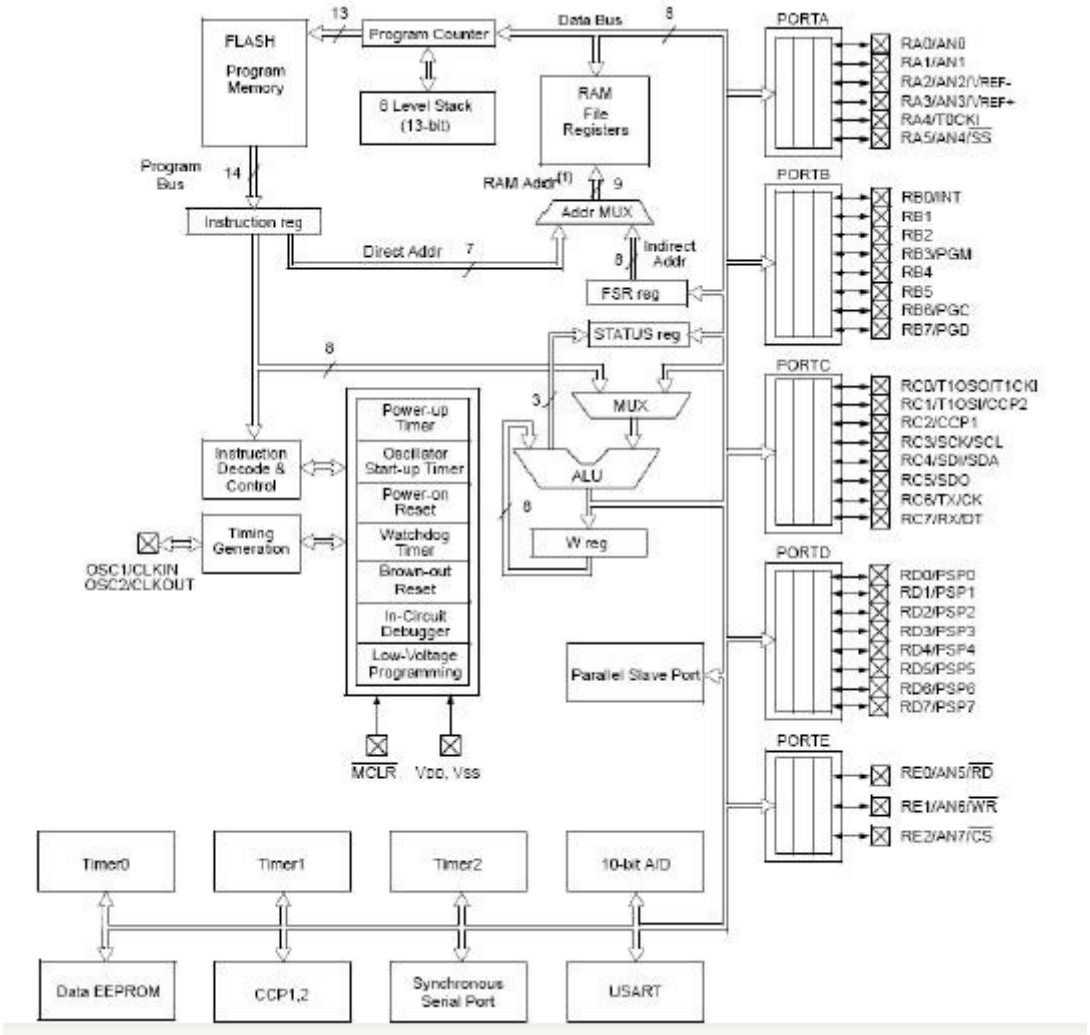



Figure 4.10 (6)

**4.8 Compilers (7)**

A compiler is a computer program (or set of programs) that translates text written in a computer language (the source language) into another computer language (the target language). The original sequence is usually called the source code and the output called object code. Commonly the output has a form suitable for processing by other programs (e.g., a linker), but it may be a human readable text file.

The most common reason for wanting to translate source code is to create an executable program. The name "compiler" is primarily used for programs that translate source code from a high level language to a lower level language (e.g., assembly language or machine language). A program that translates between high-level languages is usually called a language translator, source to source translator, or language converter. A language rewriter is usually a program that translates the form of expressions without a change of language.

A compiler is likely to perform many or all of the following operations: lexing(lexical analysis), preprocessing, parsing, semantic analysis, code optimizations, and code generation.

### 4.8.1 Lexical analysis

Lexical analysis is the processing of an input sequence of characters (such as the source code of a computer program) to produce, as output, a sequence of symbols called "lexical tokens", or just "tokens". For example, lexers for many programming languages convert the character sequence 123 abc into two tokens: 123 and abc (whitespace is not a token in most languages). The purpose of producing these tokens is usually to forward them as input to another program, such as a parser.

A lexical analyzer, or lexer for short, can be thought of having two stages, namely a scanner and an evaluator. (These are often integrated, for efficiency reasons, so they operate in parallel.)

The first stage, the scanner, is usually based on a finite state machine. It has encoded within it information on the possible sequences of characters that can be contained within any of the tokens it handles (individual instances of these character sequences are known as lexemes). For instance, an *integer* token may contain any sequence of numerical digit characters. In many cases the first non-whitespace character can be used to deduce the kind of token that follows, the input characters are then processed one at a time until reaching a character that is not in the set of characters acceptable for that token (this is known as the maximal munch rule). In some languages the lexeme creation rules are more complicated and may involve backtracking over previously read characters.

A lexeme, however, is only a string of characters known to be of a certain type. In order to construct a token, the lexical analyzer needs a second stage, the evaluator, which goes over the characters of the lexeme to produce a *value*. The lexeme's type combined with its value is what properly constitutes a token, which can be given to a parser. (Some tokens such as parentheses do not really have values, and so the evaluator function for these can return nothing. The evaluators for integers, identifiers, and strings can be considerably more complex. Sometimes evaluators can suppress a lexeme entirely, concealing it from the parser, which is useful for whitespace and comments.)

Though it is possible and sometimes necessary to write a lexer by hand, lexers are often generated by automated tools. These tools generally accept regular expressions that describe the tokens allowed in the input stream. Each regular expression is associated with a production in the lexical grammar of the programming language that evaluates the lexemes matching the regular expression. These tools may generate source code that can be compiled and executed or construct a state table for a finite state machine (which is plugged into template code for compilation and execution).

Regular expressions compactly represent patterns that the characters in lexemes might follow. For example, for an English-based language, a NAME token might be any English alphabetical character or an underscore, followed by any number of instances of any ASCII alphanumeric character or an underscore. This could be represented compactly by the string [a-zA-Z_][a-zA-Z_0-9]*. This means "any character a-z, A-Z or _, followed by 0 or more of a-z, A-Z, _ or 0-9".

Regular expressions and the finite state machines they generate are not powerful enough to handle recursive patterns, such as "*n* opening parentheses, followed by a statement, followed by *n* closing parentheses." They are not capable of keeping count, and verifying that *n* is the

same on both sides — unless you have a finite set of permissible values for *n*. It takes a full-fledged parser to recognize such patterns in their full generality. A parser can push parentheses on a stack and then try to pop them off and see if the stack is empty at the end.

The Lex programming tool and its compiler is designed to generate code for fast lexical analysers based on a formal description of the lexical syntax. It is not generally considered sufficient for applications with a complicated set of lexical rules and severe performance requirements; for instance, the GNU Compiler Collection uses hand-written lexers.

### 4.8.2 Preprocessor

Preprocessor is a program that processes its input data (source code) to produce output that is used as input to another compilation phase.  The amount and kind of processing done depends on the nature of the preprocessor; some preprocessors are only capable of performing relatively simple textual substitutions and macro expansions.

### 4.8.3 Parsing

parsing is the process of analyzing an input sequence (read from a file or a keyboard, for example) in order to determine its grammatical structure with respect to a given formal grammar. It is formally named syntax analysis. A parser is a computer program that carries out this task. The name is analogous with the usage in grammar and linguistics. The term parseable is generally applied to text or data which can be parsed.

Parsing transforms input text into a data structure, usually a tree, which is suitable for later processing and which captures the implied hierarchy of the input. Generally, parsers operate in two stages, first identifying the meaningful tokens in the input, and then building a parse tree from those tokens.

Parsers have simple grammars with few exceptions. Parsers for programming languages tend to be based on context-free grammars because fast and efficient parsers can be written for them. However, context-free grammars are limited in their expressiveness because they can describe only a limited set of languages. Informally, the reason is that the memory of such a language is limited. The grammar cannot remember the presence of a construct over an arbitrarily long input; this is necessary for a language in which, for example, a name must be declared before it may be referenced. More powerful grammars, however, cannot be parsed efficiently. Thus, it is a common strategy to create a relaxed parser for a context-free grammar which accepts a superset of the desired language constructs (that is, it accepts some invalid constructs); later, the unwanted constructs can be filtered out. It is usually easy to define a context-free grammar which includes all desired language constructs; on the other hand, it is often impossible to create a context-free grammar which admits *only* the desirable constructs. Parsers are usually not written by hand but are generated by parser generators.

Semantic Analysis

In computer science, **semantic analysis** is a pass by a compiler that adds semantical information to the parse tree and performs certain checks based on this information. It logically follows the parsing phase, in which the parse tree is generated, and logically precedes the code generation phase, in which executable code is generated. (In a compiler implementation, it may be possible to fold different phases into one pass). Typical examples

of semantical information that is added and checked is typing information (type checking) and the binding of variables and function names to their definitions (object binding). Sometimes also some early code optimization is done in this phase.

For this phase the compiler usually maintains so-called *symbolic tables* in which it stores what each symbol (variable names, function names, etc.) refers to.

### 4.8.4 Code Generation

**code generation** is the process by which a compiler's *code generator* converts a syntactically-correct program into a series of instructions that could be executed by a machine. Sophisticated compilers may use several cascaded code generation stages to fully compile code; this is due to the fact that algorithms for code optimization are more readily applicable in an intermediate code form, and also facilitates a single compiler that can target multiple architectures as only the final code generation stage (the *backend*) would need to change from target to target.

The input to the code generator stage typically consists of a parse tree, abstract syntax tree, or intermediate language code (often in three address code form). Since the target machine may be a physical machine such as a microprocessor, or an abstract machine such as a virtual machine or an intermediate language, (human-readable code), the output of code generator could be machine code, assembly code, code for an abstract machine (like JVM), or anything between.

In a more general sense, code generation is used to produce programs in some automatic manner, reducing the need for human programmers to write code manually. Code generations can be done either at runtime, including load time, or compiler time. Just-in-time compilers are an example of a code generator that produce native or nearly native code from byte-code or the like when programs are loaded onto the compilers. On the other hand, a compiler-compiler, (yacc, for example) almost always generates code at compiler time. A preprocessor is an example of the simplest code generator, which produces target code from the source code by replacing predefined keywords.

When code generation occurs at runtime, it is important that it is efficient in space and time. For example, when regular expressions are interpreted and used to generate code at runtime, a non-determistic FSA instead of deterministic one is often generated because usually the former can be created more quickly and occupies less memory space than the latter. Despite it generally generating less efficient code, code generation at runtime can take the advantage of being done at runtime. Some people cite this fact to note that a JIT compiler can generate more efficient code than a compiler invoked before runtime, since it is more knowledgeable about the context and the execution path of the program than when the compiler generates code at compile time.

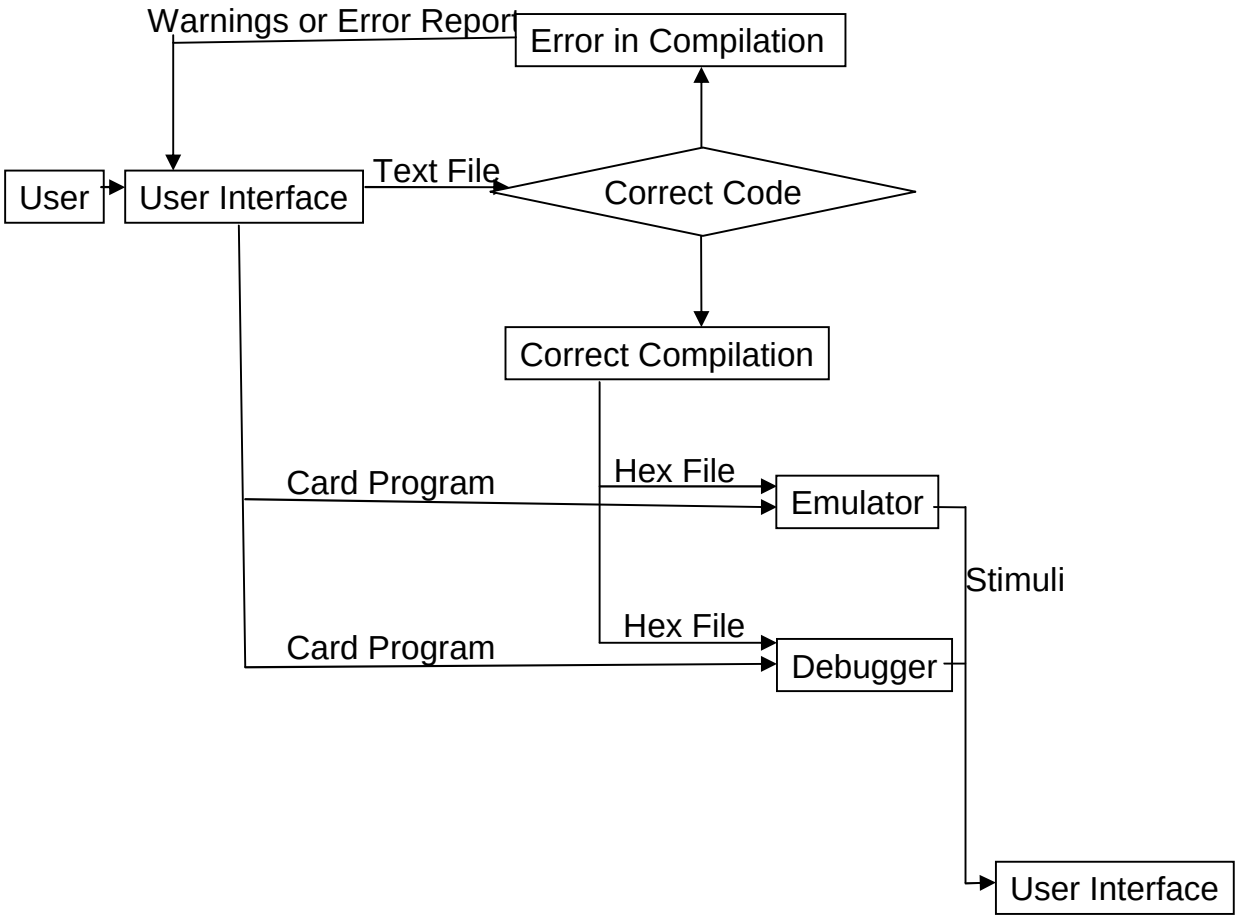
In addition to basic conversion from intermediate representation into machine instructions, code generator also tries to use faster instructions, use fewer instructions, exploit available fast registers and avoid redundant computations.

- Instruction selection. With the diverse instructions supported in a target machine, instruction selection deal with the problem of which instructions to use
- Instruction scheduling. In what order to run the instructions.

- Register allocation. The speed gap between processors and memory is partially bridged by the registers in processors. How to put useful variables into registers has a great impact of the final performance.

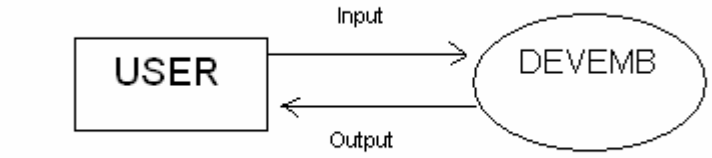
The usual method is a state machine, or weak artificial intelligence scheme that selects and combines templates for computations.

## 5 FLOWCHART

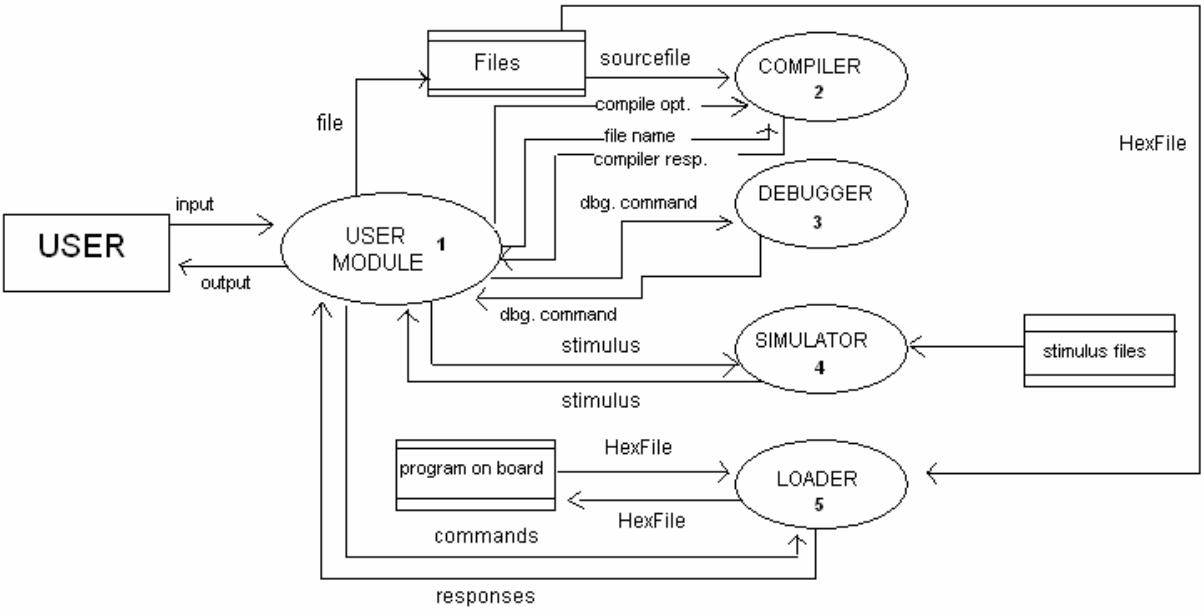


# 6 DATA FLOW DIAGRAMS

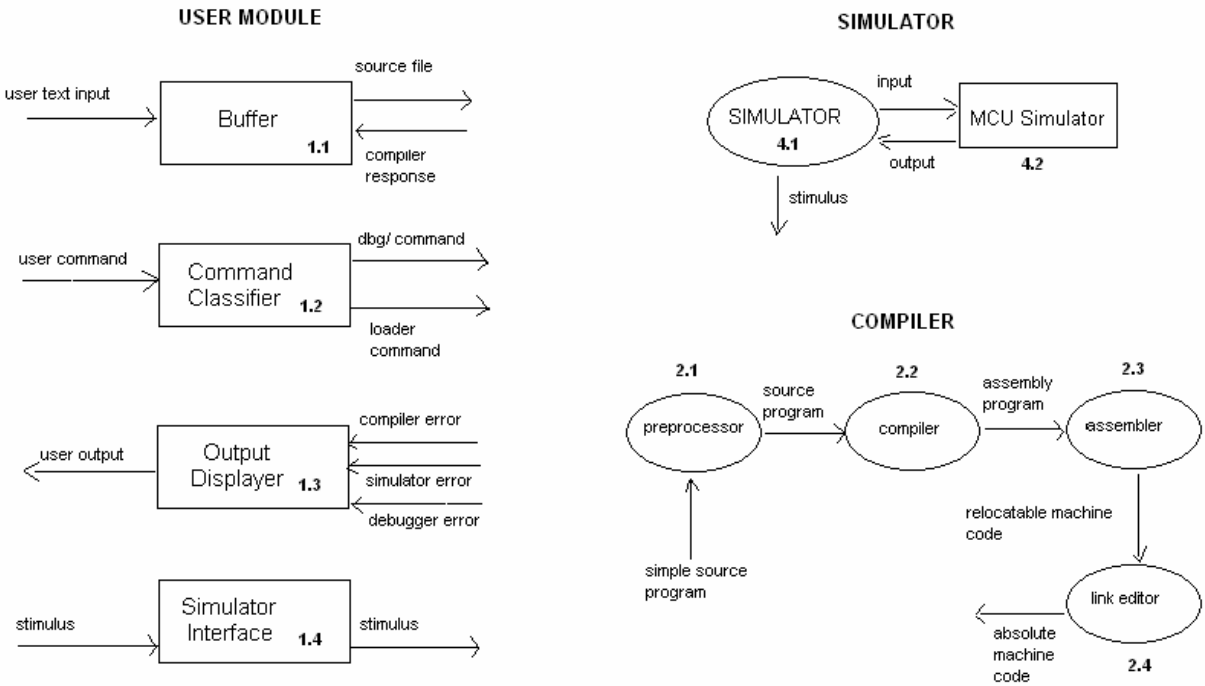
## DFD- Level 0




## DFD- Level 1




## DFD- Level 2



## 7 DATA DICTIONARY

| **Name** | User Input |
|---|---|
| **Where used / How used** | Input to >>**User Module** |
| **Description:**<br>The basic idea of this data is to provide user to to do some register related coding like in mplab.Text editor is going to be used.User command goes to command classifier to classify the commands. | |

| **Name** | User Output |
|---|---|
| **Where used / How used** | Output from << **User Module** |
| **Description:**<br>The basic idea of this data is to show the the user what is the output of Project . | |

| **Name** | File |
|---|---|
| **Where used / How used** | Input to >> **File Data Storage**<br>Output from << **User Module** |
| **Description:**<br>This data as known includes all sort of files like source and hex files. | |

| **Name** | Compile Option |
|---|---|
| **Where used / How used** | Input to >> **Compiler**<br>Output from << **User Module** |
| **Description:**<br>This is a simple compile option like –c in gcc -c file.c .But this will be a simple option file name because it will be enough. | |

| **Name** | File Name |
|---|---|
| **Where used / How used** | Output from << **User Module**<br>Input to >> **Compiler** |
| **Description:**<br>This name will be used for compile.File data storage will give the rest of file to compiler. | |

| **Name** | Compiler Response |
|---|---|
| **Where used / How used** | Input to >> **User Module**<br>Output from << **Compiler** |
| **Description:**<br>Compiler sends the data to user module whether its true or not or the absolute machine code to send the user as an output or use it. | |

| **Name** | Debbuger Command |
|---|---|
| **Where used / How used** | Input to >> **User Module**<br>Output from << **Debugger**<br>**or vice versa** |
| **Description:**<br>Debugger command is a command that specify and arrange the input and output of debugger. | |

| **Name** | Stimulus |
|---|---|
| **Where used / How used** | Input to >> **User Module**<br>Output from << **Simulator**<br>**or vice versa** |
| **Description:**<br>Stimulus is a file that includes the input and output files.In stimulus there is a text file that arranges the job that will be done step by step per clock time. | |

| **Name** | Commands |
|---|---|
| **Where used / How used** | Input to >> **Loader**<br>Output from << **User Module** |
| **Description:**<br>These are some specific commands that comes to loader as loader commands.Loader is going to show the results to the screen after that.This request comes from user. | |

| Name | Responses |
|---|---|
| Where used / How used | Input to >> **User Module**<br>Output from << **Loader** |
| **Description:**<br>This prints the screen whether it is loaded or not.Loader will arrange it .An error message shown in output displayer may come. | |

## 8 PROCESS SPECIFICATIONS

**1 User Module:**

This part is the most important part of our project.It is simply known as the mplab.exe.It means it has the same characteristics of mplab.There are also some parts that construct it.This is the outher layer that also communicates with the user an taking inputs.

**1.1 Buffer:**

This process has the same characteristics with a simple text editor.User text input is its input and it creates a source file that goes to compiler.Compiler will run the rest.

**1.2 Command Classifier:**

It classifies the command that comes from the user and sends these instructions to debugger or loader .It is a bridge like process that connects the debugger and loader to user module.

**1.3 Output Displayer:**

It displays the output .It can be also an error message that comes from other process which connects to it.It displays the message on screen.

**1.4 Simulator Interface:**

It is an interface that takes the the stimulus files and the user will be able to see the changes .It is the interface of Simulator so ;the Simulator deals with the code part and you can see changes from screen.

**2 Compiler :**

It is the part that we all have to know /It compiles the code that you write in.It takes the file from file data storage unit then gives it to user module back as an absolute machine code.

**2.1 Preprocessor:**

It translates the simple source code that we write and store to file data storage unit to source program that compiler understands.It is preparation to compile.

**2.2 Compiler:**

It translates the source program to assembly program.

**2.3 Assembler:**

It translates the assembly program to relocatable machine code.

**2.4 Link-editor:**

Last step of compilation.It translates the relocatable machine code to absolute machine code that will be sent to User Module.

**3 Debugger:**

It is the process that helps to run the code step by step to find errors.

**4 Simulator:**

Simulator is connected to the User Module .It learns when to do simulation from User Module then it sends the output.

**4.1 Simulator:**

This is the main part of the simulation it sends an gets data from MCU Simulator and do the rest.

**4.2 MCU Simulator class:**

This process is a simple class that is designed to help the main part it takes the instructions and give the exact data that Simulator needs.

**5 Loader:**

It gets and gives the hex files to the board .Also its main job is to take user commands that came and load the data to screen.

# 9 REQUIREMENTS

## 9.1 Interface Requirements

As for all softwares, graphical user interface is very important for our project too since easy to use, easy to learn and adaptation is crucial for all softwares. Therefore, we will make the user interface clear and understandable. Moreover, graphical user interface will meet all the users' needs as far as it is possible. Hence, while we were deciding our tool's features, we took into consideration both the inexperienced and professionals. As a conclusion, we considered the user satisfaction as the primary goal of our project.

### 9.1.1 General IDE Overview

- IDE will be based on multiple document interface concept because several sheets can be used to write code and simulate them.
- Project title header will contain the name of the application and the current active sheet.
- Menu bar has to facilitate to access to all system features of the application.
- Tool bar has to contain the symbols of the most frequently used features.
- There will be a text editor to write code.
- Status bar will display the processor information and cursor position.

### 9.1.2 Menu Bar

Menu bar will contain the following menus:

- File
- Edit
- View
- Project
- Debugger
- Macros
- Window
- Help

#### 9.1.2.1 File Menu

- New: This option enables the user to create a new file.
- Open: This option enable the user to open an existing file.
- Close: This option enable the user to close the current file.
- Save: This option enable the user to save the current file.
- Save As: This option enable the user to save the current file with a different name.
- Recent Files: This option enable the user to view and open the most recently used files.
- Print: This option enable the user to print the current file.
- Exit: This option enables the user to exit from the program.

#### 9.1.2.2 Edit Menu

- Undo: This option enable the user to go to the situation before the changes made recently.
- Redo: This option enable the user to make the change which is canceled again.
- Cut: This option enable the user to cut the selected part.
- Copy: This option enable the user to copy the selected part.
- Paste: This option enable the user to paste the last cut or copied part.
- Select All: This option enable the user to select everything in the current sheet.

- Find: This option enable the user to find the desired word or phrase in the current sheet.
- Replace: This option enable the user to replace the desired word or phrase with another desired word or phrase.
- Properties: This option enables the user to change the background color, font and size of the text, etc.

#### 9.1.2.3 View Menu

- Toolbars: This option enables the user to specify the tool bar by selecting.
- Status Bar: This option enables the user to show or hide.

#### 9.1.2.4 Project Menu

- Project Wizard: This option enables the user to create a new project.
- New: This option enables the user to create a new project.
- Open: This option enable the user to open an existing project.
- Close: This option enable the user to close the current project.
- Quickbuild: This option enables the user to compile its code.
- Clean: This option enable the user to delete all files in the project.
- Build All: This option enable the user to compile all files in the project.
- Make: This option enable the user to link all files in the project.
- Save Project: This option enable the user to save the current project.
- Save Project As: This option enable the user to save the current project with a different name.
- Add Files to Project: This option enable the user to add files from other projects to the current project.
- Add New Files to Project: This option enable the user to add a new file.
- Remove File From Project: This option enable the user to remove file from the current project.

#### 9.1.2.5 Debugger Menu

- Clear Memory: This option enable the user to clear the selected memory (All Memory, Program Memory)
- Run: This option enable the user to execute the code.

- Animate: This option enable the user to see the program execution step by step automatically.
- Halt: This option enable the user to stop the running animation.
- Step Into: This option enable the user to step into the desired code segment.
- Step Over: This option enable the user to step over the unwanted code segment.
- Step Out: This option enable the user to step out from the unwanted code segment.
- Reset: This option enable the user to restart the debugger.
- Breakpoints: This option enables the user to put breakpoint to the desired place to stop the program.
- Stopwatch: This option enables the user to measure the time.
- Stimulus Controller: This option enables the user to draw a scenario the program will execute accordingly.

#### 9.1.2.6 Macros Menu

- Record Macro: This option enable the user to save the current project as a macro in order to support reusage.
- Stop Recording: This option enable the user to cancel the recording of a macro.
- Open Macro: This option enable the user to load previously recorded macro.
- Exit Macros: This option enables the user to exit from the macros.

#### 9.1.2.7 Window Menu

- Close All: This option enable the user to close all open windows.
- Cascade: This option enables the user to cascade all the open windows.
- Tile Horizontally: This option enable the user to spread all the open windows horizontally.
- Tile Vertically: This option enable the user to spread all the open windows vertically.

Moreover, all opened windows' names are shown at the bottom of the Window Menu and user can reach any window from this window list by clicking them.

#### 9.1.2.8 Help Menu

- About: This option enables the user to learn some information about our product. Maybe they can connect our web site by this option.
- Custom Help: This option enables the user to see the tutorials about usage of our product.

### 9.1.3 Tool Bar

Tool bar contains symbols of the most frequently used actions and these are the followings:

- New: This icon is the shortcut of the create a new file operation.
- Open: This icon is the shortcut of the open an existing file operation.
- Save: This icon is the shortcut of save the current file operation.
- Cut: This icon is the shortcut of cut the selected part operation.
- Copy: This icon is the shortcut of copy the selected part operation.
- Paste: This icon is the shortcut of paste the last cut or copied part operation.
- Print: This icon is the shortcut of print the current file operation.
- Find: This icon is the shortcut of the desired word or phrase in the current sheet operation.
- Help: This icon is the shortcut of the Help Menu.
- New Project: This icon is the shortcut of the create a new project operation.
- Open Project: This icon is the shortcut of the open an existing project operation.
- Save Project: This icon is the shortcut of save the current project operation.
- Make: This icon is the shortcut of the link all files in the project operation.
- Build All: This icon is the shortcut of the compile all files in the project operation.
- Run: This icon is the shortcut of execute the code operation.
- Halt: This icon is the shortcut of stop the running animation operation.
- Animate: This icon is the shortcut of see the program execution step by step automatically operation.
- Step Into: This icon is the shortcut of step into the desired code segment operation.
- Step Over: This icon is the shortcut of step over the unwanted code segment operation.
- Step Out: This icon is the shortcut of step out from the unwanted code segment operation.
- Reset: This icon is the shortcut of restart the debugger operation.

### 9.1.4 Text Editor

This is used for writing code and it will be an empty text sheet.

### 9.1.5 Console

- Console view will display warnings (if there are any) after building of the project.
- Console view will display errors (if there are any) after building of the project.
- Console view will display messages (if there are any) after building of the project.

### 9.1.6 Status Bar

- Status Bar will display the processor type information.
- Status Bar will display the cursor information in terms of line and column numbers.

### 9.1.7 Hot Keys

There exists some common hot keys for different kinds of programs that users often familiar from many programs such as Ctrl-S, Ctrl-C, Ctrl-V, etc. Therefore, we will add these hot key functions to our product. These are the followings:

- Ctrl-S: Save
- Ctrl-C: Copy
- Ctrl-X: Cut
- Ctrl-V: Paste
- Shift-Insert: Paste
- Ctrl-Z: Undo
- Ctrl-Y: Redo

## 9.1 Emulator Overview

Our emulator will be a terminal emulator. At the beginning, user will see some default information on the terminal and as program runs, these information change accordingly. Besides, all actions can be done with commands. At the top of the terminal some counters will be seen. These counters are:

total: ic = 00000000 sim time = 0.00 ns

delta: ic = 00000000 sim time = 0.00 ns

stopw: ic = 00000000 sim time = 0.00 ns

stackdepth = 0 max stackdepth = 0:0000 0000 0000 0000 0000 0000 0000 0000

W = 00 IP = 0000 Status = 18 (bank = 0 NZ ND NC) CONF = 3FFF

0000 3FFF ADDLW FF

### 9.2.1 Registers Part

Registers part will take place below the counters part. In this part registers and their contents in hexadecimal format will be demonstrated. They will be sorted according to the register numbers 0 to 1F0.

Since, this is the terminal emulator all actions can be made by commands and for these commands we need to some keys. Therefore, for our emulator there will some special purpose keys and these will be seen at the top of the Registers part. These special keys are ↑, ↓, PgUp, PgDn, Home, End, <Ctrl-Tab>.

- ↑ : This key will be used to scroll the Registers part up line by line.
- ↓ : This key will be used to scroll the Registers part down line by line.
- PgUp : This key will be used to scroll the Registers part up page by page.
- PgDn : This key will be used to scroll the Registers part down page by page.
- Home : This key will be used to see the top of the Registers part.
- End : This key will be used to see the bottom of the Registers part.
- <Ctrl-Tab> : This key will be used to pass to the other parts.

### 9.2.2 EEPROM

EEPROM part will take place in our emulator below the Registers part. This part contains EEPROM registers and their contents, this part gene. They are also sorted according to their register numbers 0 to F0. Again this part has control keys and these will be demonstrated at the top of the EEPROM part. These keys are ↑, ↓, PgUp, PgDn, Home, End, <Ctrl-Tab>. Since, I have written these keys' functionalities, I do not want to write them again.

### 9.2.3 Command Window

Command Window part will be below the EEPROM part. At the bottom of this part of course command line will take place. This part also will have control keys and they will be seen at the top of the Command Window part. These keys are: ←, →, Home, End, Del, Esc, F3, <Ctrl-Tab>. Functionalities of these keys are:

- ← : This key is used for to go to the previous character in the command line.
- → : This key is used for to go to the next character in the command line.
- Home : This key is used for to go to the start of the command line.

- End : This key is used for to go to end of command line.
- Del : This key is used for to delete the last inserted character.
- Esc : This key is used for to escape.
- F3 : This key is used for to see the last command inserted into command line.
- <Ctrl-Tab> : This is used for to pass to the other parts.

If "H" is inserted into command line, help topics will be seen in the main part of Command Window. These help topics are: ENTEREeprom, READUsestris, ALLRegs, STIMulus, DIR, RETurn, BRegisters, SAve, Fill, Proceede, #, CHECKPoint, RESET, REGBreakpoints, REGDisplay, THRottle, WATchregs, EXecute, WRite, Breakpoints, Trace, Output, CHECKStack, SOFTUart, SHOWregs, FEEprom, MACro, WAke, TTrace, Registers, Help, Input, TRACETrue, STOPWatch, CALC, HEXScreen, EEEprom, OSCope, FRequency, Name, Unassemble, Quit, View, PAUSE, SETPorts, CONFig, BROwse, HEX, I2C, EEprom, Enter, Go, ;, Keys.

## 9.3 Non-Functional Requirements

### 9.3.1 Usability

Without considering how the software is good and powerful, if it not user friendly, it is not successful. Our observations in the current product range show us that most of these programs' user interface does not satisfy the users' needs. Therefore, our primary goal in this project is to create the best user interface as far as it is possible. So, our interface should be easy to understand and easy to use. To realize the user satisfaction, the interface items will be well designed and located effectively that the user can easily access everything without spending more effort. Moreover, menus and tool bar will be clear and the user will be able to access some of the menu items with hot keys. Besides, to facilitate the adaptation to our product and solving problems easily, we plan to put a tutorial about our product.

### 9.3.2 Reliability

Our product will be used by students, academicians and professionals in the industry. Hence, our product should be cleaned from the bugs as much as we can because any corruption which yields data loss will affect users.

### 9.3.3 Portability

Our topic is a general topic and our product will be used by students, academicians and professionals in the industry. Namely, our product has a wide range of usage area because of this some users can use Windows, some users can use Linux and some user can use other operating systems. Therefore, our product should be run almost all computer platforms. Since we will develop our product in Java, it will have high portability. It will run in almost all computers having Java runtime environment.

### 9.3.4 Performance

Since time is very important for people, speed of our product becomes an important issue in our design. In general, wrong programming methodologies that are used in the applications

slow down the programs, not the complex algorithms used in the applications. Therefore, we will try not to do this mistake. The usage of system resources will be reduced as much as possible to increase the performance of our design. User can run other applications easily while our program working.

## 10 DESIGN CONSTRAINTS

The programming language will be Java so that we can use the object oriented concepts and reusability will be achieved. Our main graphics library will be Open GL and we will use 3D Max for graphical design part.

## 11 REFERENCES

1 http://www.htsoft.com/products/picccompiler.php
2 http://www.gnupic.org/
3 http://www.dattalo.com/gnupic/gpsim.pdf
4 http://www.speedy-bl.com/pic16fxxx-e.htm
5 lecture notes
6 lab manual
7 http://en.wikipedia.org

**references are shown in paranthesis near the related titles