



**Middle East Technical University**  
**Department of Computer Engineering**

# **PIDE**

Emulator and Development Environment for

CEng Embedded System Card

## **USER MANUAL**

by

**SimSys Corporation**

## Table of Contents

1.	INTRODUCTION.....	3
2.	GRAPHICAL USER INTERFACE.....	4
2.1	GENERAL VIEW.....	4
2.2	MENUS .....	5
3.	ASM++ LANGUAGE FORMAT .....	8
3.1	GENERAL SPECIFICATIONS .....	8
3.2	IDENTIFIERS.....	8
3.3	RESERVED WORDS .....	9
3.4	DELIMITERS AND OPERATORS .....	9
3.5	VARIABLES.....	9
3.6	NUMBERS.....	10
3.7	EXPRESSIONS.....	10
3.8	STATEMENTS.....	11
3.9	COMMENTS.....	13
3.10	EXTENDED INSTRUCTION SET .....	13
4.	START USING PIDE .....	15

## **1. INTRODUCTION**

PIDE is a complete integrated development and simulation environment designed for CENG336 Embedded Systems Board by SimSys Corporation. This document gives information about the usage of the program, describes the graphical user interface and introduces a new language ASM++.

## 2. GRAPHICAL USER INTERFACE

### 2.1 GENERAL VIEW

Below in Figure 2.1, the GUI of the PIDE program, showing the menus, toolbars, tabs, workspace view and the status bar can be found. It shows the case with an opened project, and two opened asm files. The workspace view is also present on the left hand side. The program is able to handle multiple opened files using a tabbed view

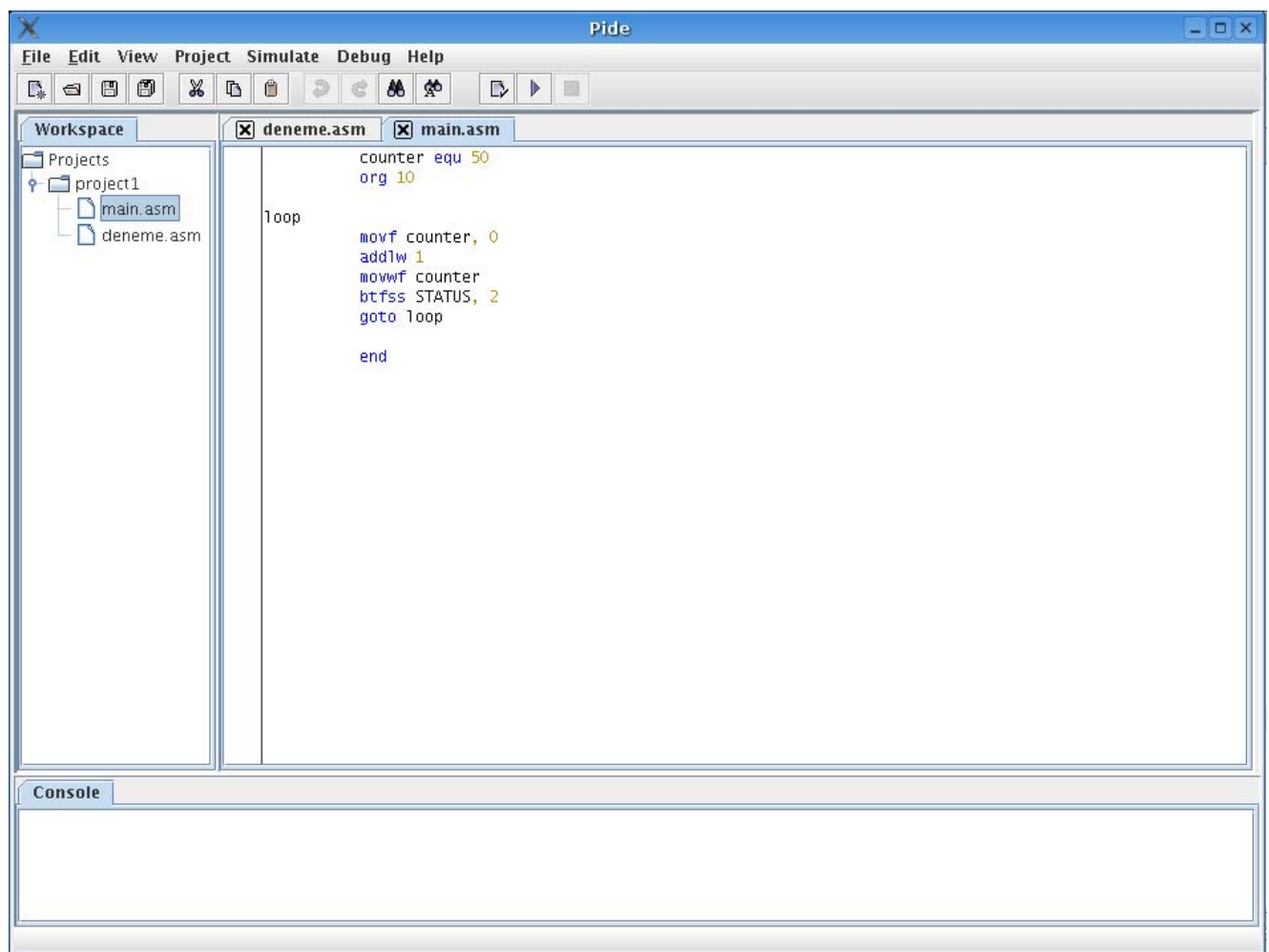


Figure 2.1

In Figure 2.2, the menu bar of the PIDE is shown. The menu items will be explained in detail in the following sections.



**Figure 2.2**

In Figure 2.3, the toolbar of the PIDE is shown. Here exist shortcuts of the frequently used operations in the menu bar.

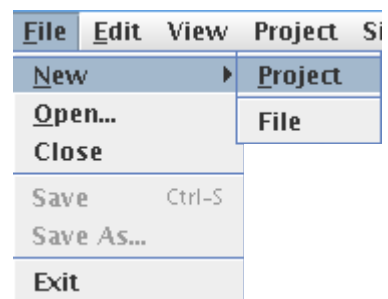


**Figure 2.3**

## 2.2 MENUS

### FILE MENU

<b>New</b>	<b>Project</b>	Create a new project.
	<b>ASM File</b>	Create a new ASM file.
	<b>ASM++ File</b>	Create a new ASM PlusPlus file.
<b>Open...</b>		Open an existing file.
<b>Close</b>		Close the current file.
<b>Save</b>		Save the current file.
<b>Save As...</b>		Save the current file with a different name or save to a different place.
<b>Exit</b>		Quit from the program.



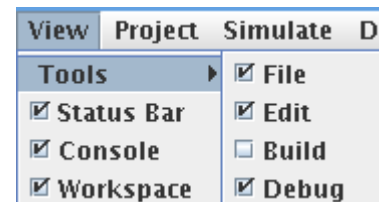
## EDIT MENU

<b>Undo</b>	Undo the last action.
<b>Redo</b>	Redo the last undo action.
<b>Cut</b>	Cut the selected item.
<b>Copy</b>	Copy the selected item.
<b>Paste</b>	Paste the last cut or copied item.
<b>Select All</b>	Select and highlight the whole text in the active tab.
<b>Find/ Replace</b>	Find and replace given word in the current file.



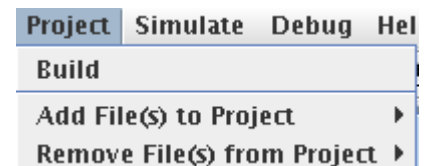
## VIEW MENU

<b>Tools</b>	Show/Hide the toolbars of File, Edit, Build and Debug menus.
<b>Status Bar</b>	Show/Hide the status bar.
<b>Console</b>	Show/Hide the console view.
<b>Workspace</b>	Show/Hide the workspace view.



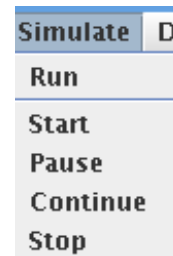
## PROJECT MENU

<b>Build</b>	Build the current project.
<b>Add File to Project</b>	Add a new file to the current project.
<b>Remove File from Project</b>	Remove a file from the current project.



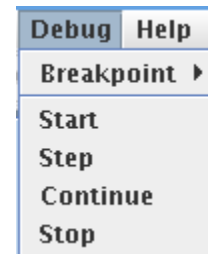
### SIMULATE MENU

<b>Run</b>	Open the simulator window.
<b>Start</b>	Start the simulation.
<b>Pause</b>	Pause the simulation.
<b>Continue</b>	Continue the simulation.
<b>Stop</b>	Stop the simulation.



### DEBUG MENU

<b>Breakpoint</b>	Add or Remove breakpoints.
<b>Start</b>	Start the debugging process.
<b>Step</b>	Execute one instruction.
<b>Continue</b>	Execute until the next breakpoint.
<b>Stop</b>	Stop the debugging process



### HELP MENU

<b>About PIDE...</b>	Show brief information about the program.
----------------------	---



### **3. ASM++ LANGUAGE FORMAT**

ASM++ is simply an improvement on assembly language, including some new instructions and introducing some high level language concepts such as control and repetitive structures and variable definitions. The name of the language is ASM++ (ASM plus plus), and the file extension is ".asmpp".

#### **3.1 GENERAL SPECIFICATIONS**

- ASM and ASM++ in PIDE are case sensitive. Instructions and other reserved words (if, else, while, etc.) should be written in lower-case letters.
- PIC 16F877 constants (STATUS, PORTA, etc.) are recognized and must be used in upper-case.
- White spaces (space character, tab character and end-of-line) serve to separate tokens; otherwise, they are ignored.
- No token can extend past end-of-line.
- Spaces may not appear inside any token except character and string literals.
- A comment begins with two forward slashes (as in C++) or with a semicolon (as in assembly language) and extends to end of line.
- There cannot be more than one statement in a line.
- No semicolons exist at the end of statements. (In fact, that does not matter since, after a semicolon, the rest of the line is considered as comment.)

#### **3.2 IDENTIFIERS**

Identifiers start with a letter and contain letters and digits.



### 3.3 RESERVED WORDS

The following keywords are reserved in ASM++ in addition to the instructions in ASM:

addff	subff	addwff	subwff	swapff	
iorwff	andwff	xorwff	movff		
if	else	for	while	do	var

### 3.4 DELIMITERS AND OPERATORS

One-character delimiters:           : ; , ( ) EOF

One-character operators:           ! < = > '

Two-character delimiters:           //

Two-character operators:           == != >= <= && ||

### 3.5 VARIABLES

Variables are introduced by the declaration of the form:

**var** *var\_name* *var\_address*

This declaration reserves the given var\_address for that variable. It is the user's responsibility to use the variable in the correct bank.

Example:

var var1 0x121

### 3.6 NUMBERS

A constant consists of a sequence of one or more digits in decimal, binary or hexadecimal format. Example usage is shown below.

Binary number: Starts with a "B" or "b" (which stands for binary) and contains digits 1, 0 enclosed in single quotes.

Ex. B`10011101` or b`10011101`

Hexadecimal number: Starts with "0x" and contains 0-9 and A-F.

Ex. 0x45AF

Decimal number: Contains digits 0-9.

Ex. 45 or 127

### 3.7 EXPRESSIONS

In ASM++, expressions are defined as below:

<expr> : <label> == <label> | <label> != <label> |  
          <label> > <label> | <label> < <label> |  
          <label> >= <label> | <label> <= <label>

<label> : <variable> | <number>

<variable> is the variable declared using **var** keyword,

<number> is the number represented in binary, decimal or hexadecimal format.  
The numbers defined with **equ** keyword are also included here.

With these expressions, when the name of the variable is used alone, its address will be considered. If the value of the variable is intended to be used, a star '\*' should be prefixed to the variable name. An example usage is below.

```
Ex.      var myvar1 0x025
          var myvar2 0x064
          myvar1 == *myvar2    →    checks if 'the content of the address 0x064'
                                   is equal to '0x025'
```

### 3.8 STATEMENTS

- **Assignment statement**

"=" is the assignment operator.

For example:

```
var a 0x021
```

```
var b 0x022
```

a = '0x0C4' → Puts the number '0x0C4' into the address 0x021

b = a      →      Copies the content of address 0x021 to the address 0x022

- **If statement**

An if-statement can be used alone or together with an else-statement. The syntax of an if-else statement is as follows:

```

if (expression)
    goto hede
else
    goto hodo

```

endif

- **Loop Statements**

The compiler supports **while**, **do-while** and **for** loops. The curly braces are compulsory regardless of the number of statements inside the loop. The syntaxes of the loop statements are as follows:

```
while (hede)
```

```
{
```

```
.....
```

```
.....
```

```
}
```

```
for (expr1; expr2; expr3)
```

```
{
```

```
.....
```

```
}
```

```
do
```

```
{
```

```
.....
```

```
}
```

```
while (hede)
```

### 3.9 COMMENTS

The comments are specified by a semicolon or two forward slashes. It will comment out the characters until the end of line.

### 3.10 EXTENDED INSTRUCTION SET

ASM++ provides a bunch of new instructions together with the basic PIC instruction set. Using these new instructions, it will be possible to do arithmetic operations between two file registers without using the working register WREG in between.

These instructions are:

**addff v1 v2** : (V1  $\leftarrow$  V1 + V2)

Adds the value of v2 to v1, and writes the result back to v1.

**subff v1 v2** : (V1  $\leftarrow$  V1 - V2)

Subtracts the value of v2 from v1, and writes the result back to v1.

**addwff** : (W  $\leftarrow$  V1 + V2)

Adds the value of v2 to v1, and writes the result to WREG.

**subwff** : (W  $\leftarrow$  V1 - V2)

Subtracts the value of v2 from v1, and writes the result to WREG.

**swapff** : (Temp  $\leftarrow$  V1, V1  $\leftarrow$  V2, V2  $\leftarrow$  Temp)

Swaps the values of v1 and v2.

**iorwff** : (V1  $\leftarrow$  V1 OR V2)

Takes the OR of v1 and v2, and writes the result to WREG.

**andwff** : (V1  $\leftarrow$  V1 AND V2)

Takes the AND of v1 and v2, and writes the result to WREG.

**xorwff** : (V1  $\leftarrow$  V1 XOR V2)

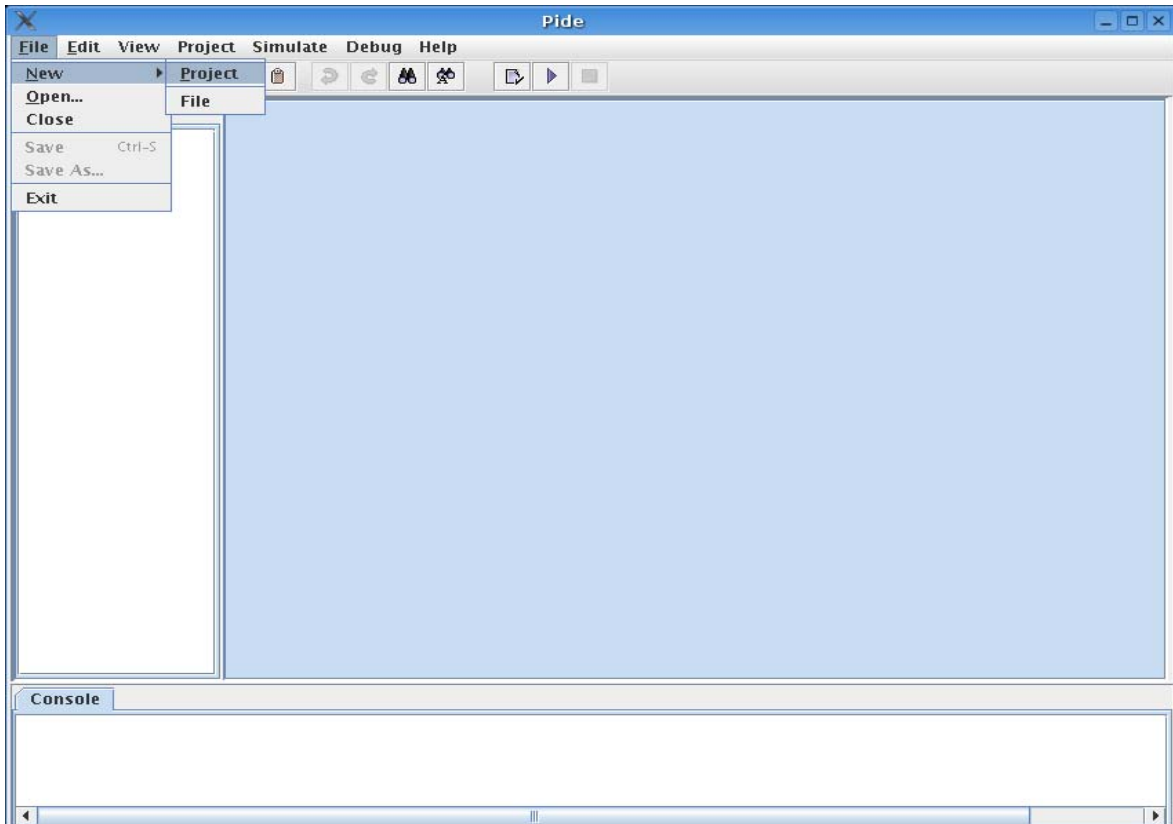
Takes the XOR of v1 and v2, and writes the result to WREG.

**movff** : (V2  $\leftarrow$  V1)

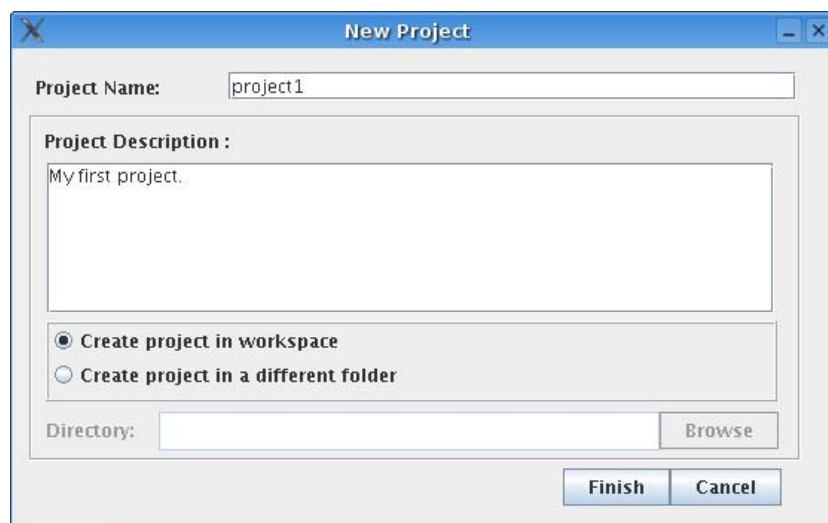
Copies the value of v1 to v2.

## 4. START USING PIDE

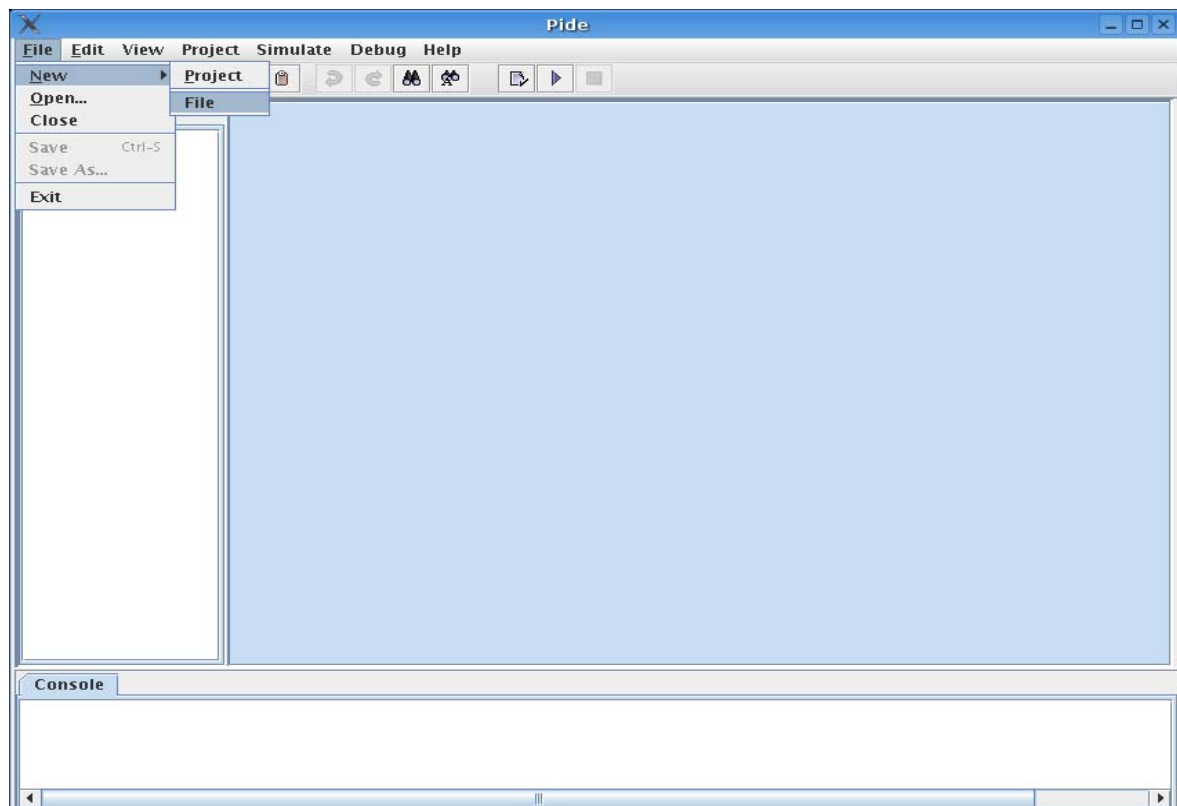
- First, create a new project through *File* → *New* → *Project*.



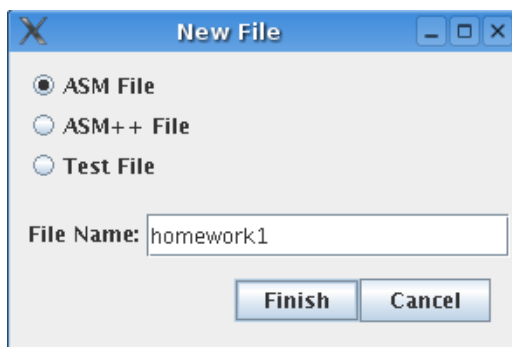
- Enter the project name and description, and choose whether to create the project in workspace or in a different folder.



- Create a new file through *File* → *New* → *File*.



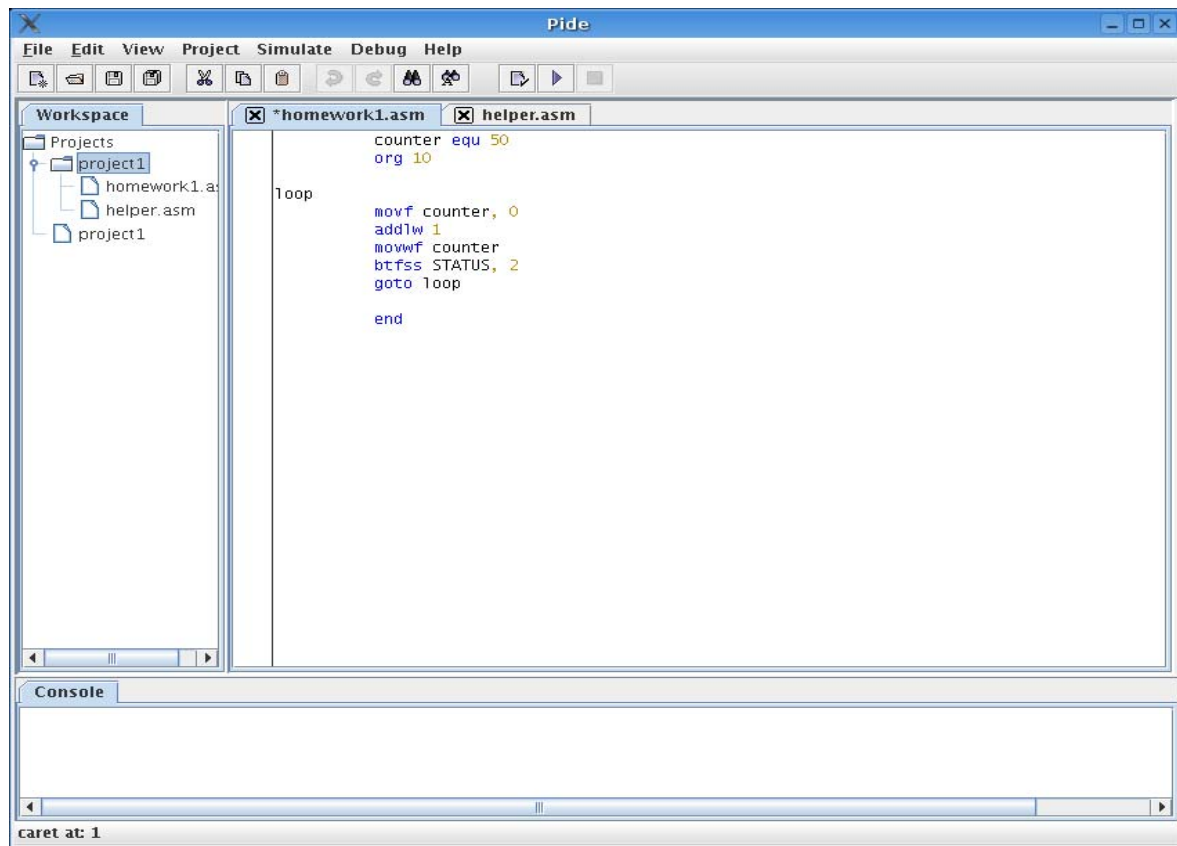
- Choose ASM or ASM++ file and enter a name.



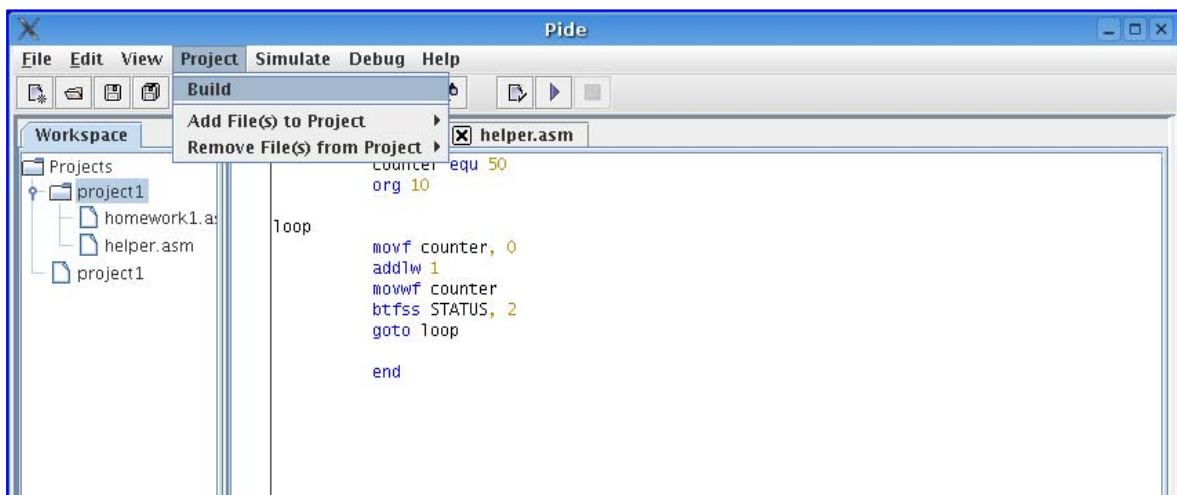
- Multiple files can be created which will be shown in a tabbed manner. A closed file can be opened by double-clicking or by right-clicking on its name on the workspace pane.



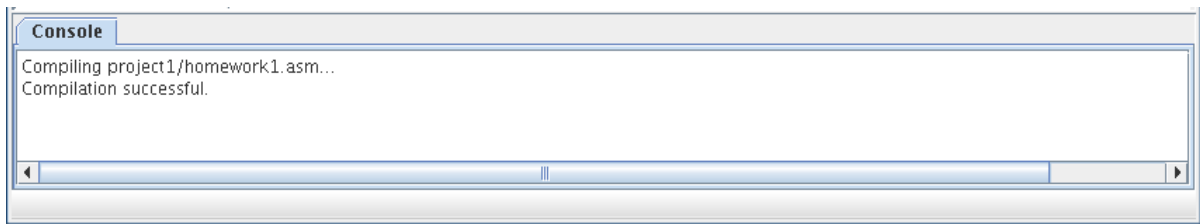
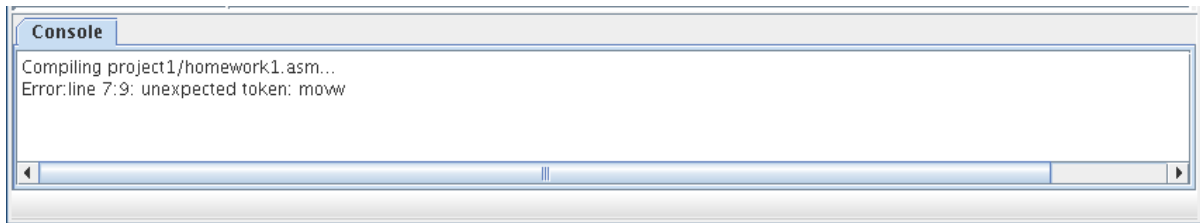
- Write the source code, which will automatically be colored.



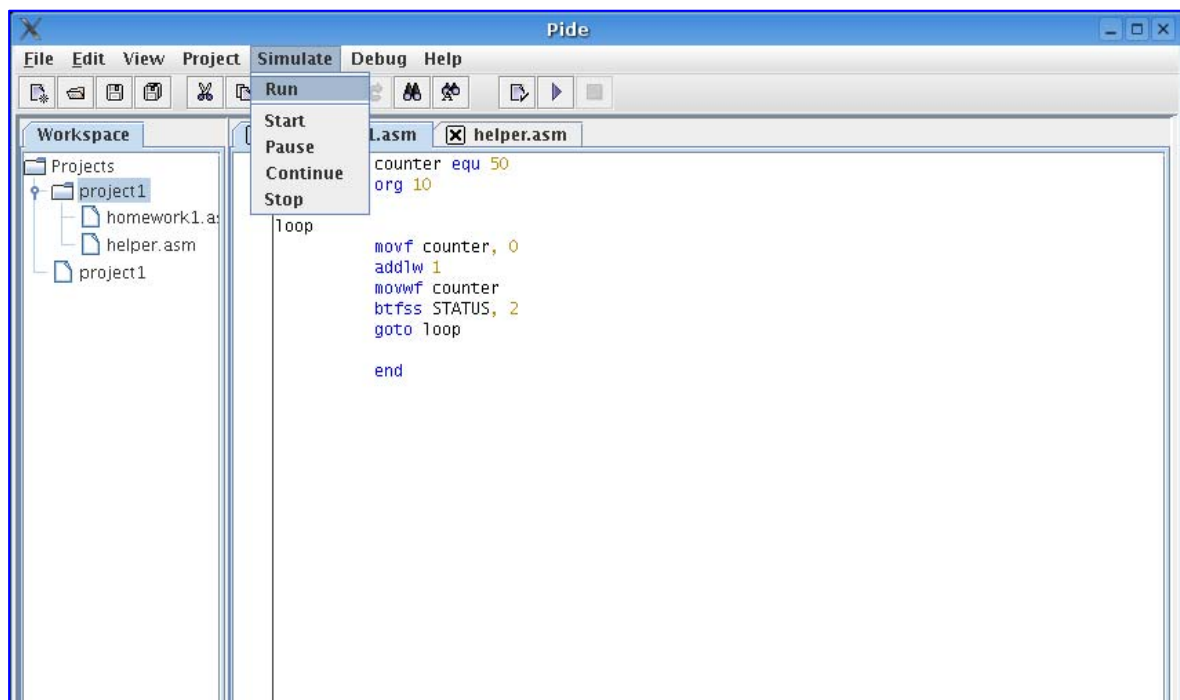
- Compile the file through *Project* → *Build*.



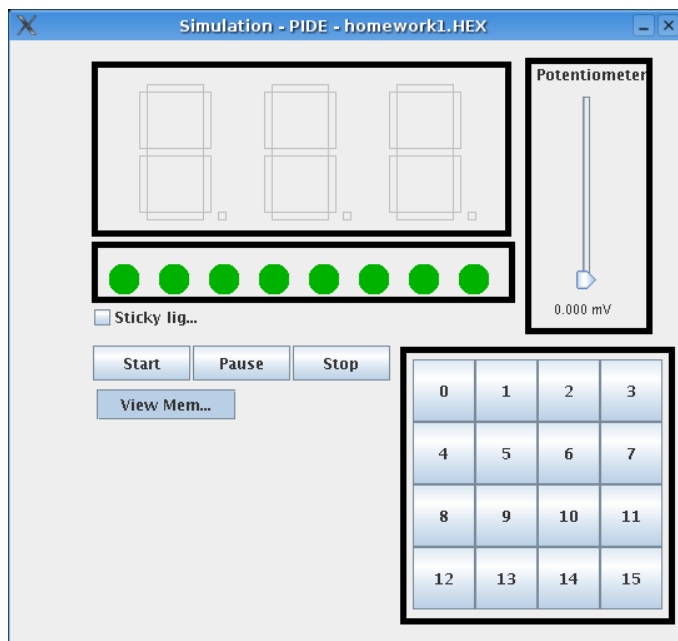
- If the file contains errors, it can be seen on the console pane. Otherwise, "Compilation Successful" message will be seen.



- Simulate the code through *Simulate* → *Run*.



- The board view and the memory view are opened. *Start/Pause/Stop* the simulation as intended. Console pane shows the simulated instructions.



Memory							
Address	Value	Address	Value	Address	Value	Address	Value
001h TMR0	0	081h OPTIO...	0	101h TMR0	0	181h OPTIO...	0
002h PCL	0	082h PCL	0	102h PCL	0	182h PCL	0
003h STATUS	0	083h STATUS	0	103h STATUS	0	183h STATUS	0
004h FSR	0	084h FSR	0	104h FSR	0	184h FSR	0
005h PORTA	0	085h TRISA	0	105h	UNIMPLEME...	185h	UNIMPLEME...
006h PORTB	0	086h TRISB	0	106h PORTB	0	186h TRISB	0
007h PORTC	0	087h TRISC	0	107h	UNIMPLEME...	187h	UNIMPLEME...
008h PORTD	0	088h TRISD	0	108h	UNIMPLEME...	188h	UNIMPLEME...
009h PORTE	0	089h TRISE	0	109h	UNIMPLEME...	189h	UNIMPLEME...
00Ah PCLATH	0	08Ah PCLATH	0	10Ah PCLATH	0	18Ah PCLATH	0
00Bh INTCON	0	08Bh INTCON	0	10Bh INTCON	0	18Bh INTCON	0
00Ch PIR1	0	08Ch PIE1	0	10Ch EEDATA	0	18Ch EECON1	0
00Dh PIR2	0	08Dh PIE2	0	10Dh EEADR	0	18Dh EECON2	0
00Eh TMR1L	0	08Eh PCON	0	10Eh EEDATH	0	18Eh	RESERVED
Address	Value	Address	Value	Address	Value	Address	Value
020h	0	0a0h	0	120h	0	1a0h	0
021h	0	0a1h	0	121h	0	1a1h	0
022h	0	0a2h	0	122h	0	1a2h	0
023h	0	0a3h	0	123h	0	1a3h	0
024h	0	0a4h	0	124h	0	1a4h	0
025h	0	0a5h	0	125h	0	1a5h	0
026h	0	0a6h	0	126h	0	1a6h	0
027h	0	0a7h	0	127h	0	1a7h	0
028h	0	0a8h	0	128h	0	1a8h	0
029h	0	0a9h	0	129h	0	1a9h	0
02ah	0	0aah	0	12ah	0	1aah	0
02bh	0	0abh	0	12bh	0	1abh	0
02ch	0	0ach	0	12ch	0	1ach	0