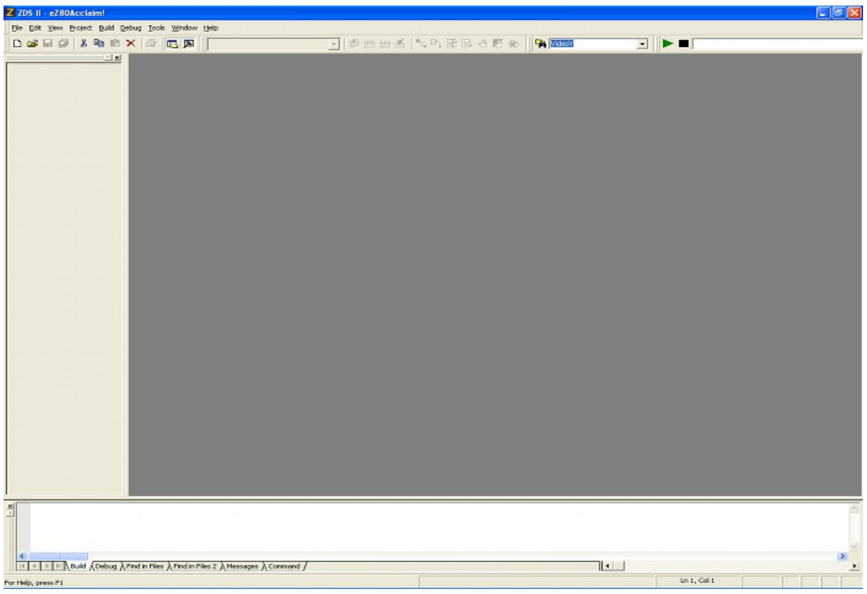
How to program in C under PROSE

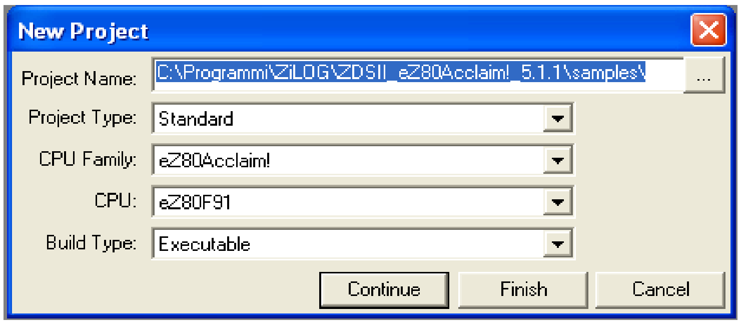
~

If you want to program in C under PROSE is necessary, first of all, download the development environment from Zilog's website. Registration is required but is free and the package to download is called: ZDS II eZ80Acclaim!

After installation, start the development environment and you will come to this screen:


ZDS II - eZ80Acclaim!
File Edit View Project Build Debug Tools Window Help
Build Debug Find in Files Find in Files 2 Messages Command
For Help, press F1
Ln 1, Col 1

We then create a small sample program. Let's go to File->New Project and this window will appear:



New Project

Project Name: C:\Programmi\ZiLOG\ZDSII_eZ80Acclaim!_5.1.1\samples\

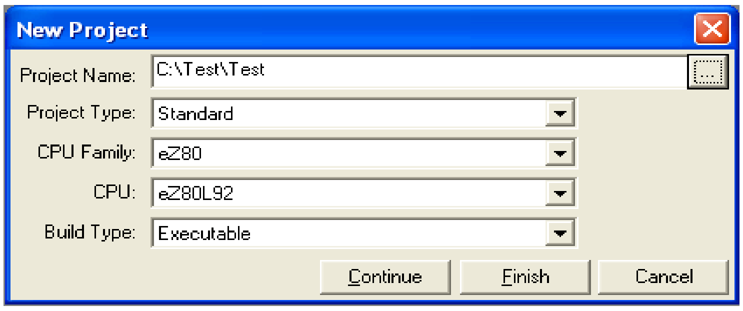
Project Type: Standard

CPU Family: eZ80Acclaim!

CPU: eZ80F91

Build Type: Executable

Select the folder to save the project and then set the various parameters:



New Project

Project Name: C:\Test\Test

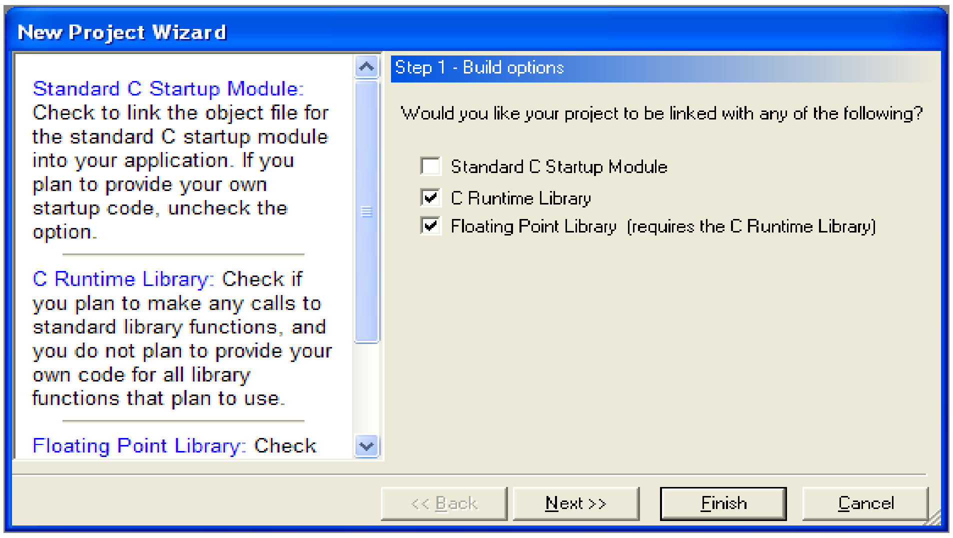
Project Type: Standard

CPU Family: eZ80

CPU: eZ80L92

Build Type: Executable

So Project Type = Standard, CPU Family = eZ80, CPU = eZ80L92 and Build Type = Executable, then press the Continue button. A window will appear for setting the build options. Here you HAVE to uncheck "Standard C Startup Module".


New Project Wizard

Standard C Startup Module: Check to link the object file for the standard C startup module into your application. If you plan to provide your own startup code, uncheck the option.

C Runtime Library: Check if you plan to make any calls to standard library functions, and you do not plan to provide your own code for all library functions that plan to use.

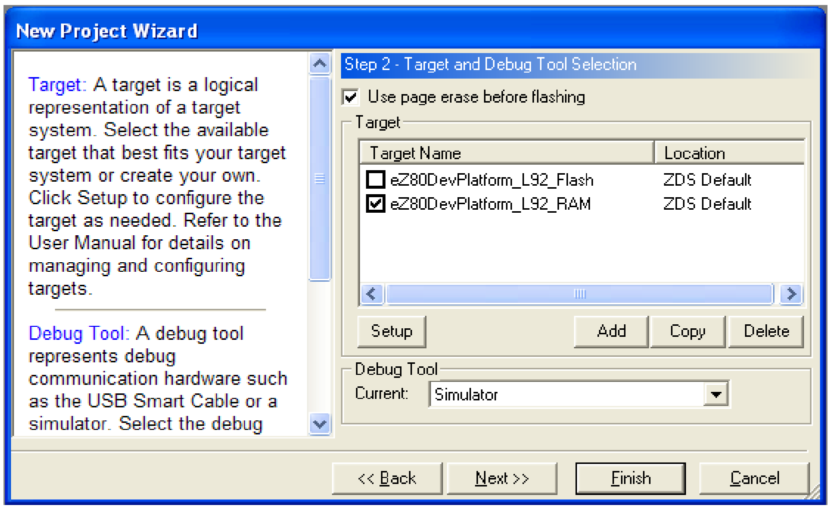
Floating Point Library: Check

Step 1 - Build options

Would you like your project to be linked with any of the following?

- [ ] Standard C Startup Module
- [✓] C Runtime Library
- [✓] Floating Point Library (requires the C Runtime Library)

Then click on Next button and the program asks you to select the Target and Debug Tool, you select "eZ80DevPlatform_L92_RAM.


New Project Wizard

Target: A target is a logical representation of a target system. Select the available target that best fits your target system or create your own. Click Setup to configure the target as needed. Refer to the User Manual for details on managing and configuring targets.

Debug Tool: A debug tool represents debug communication hardware such as the USB Smart Cable or a simulator. Select the debug

Step 2 - Target and Debug Tool Selection

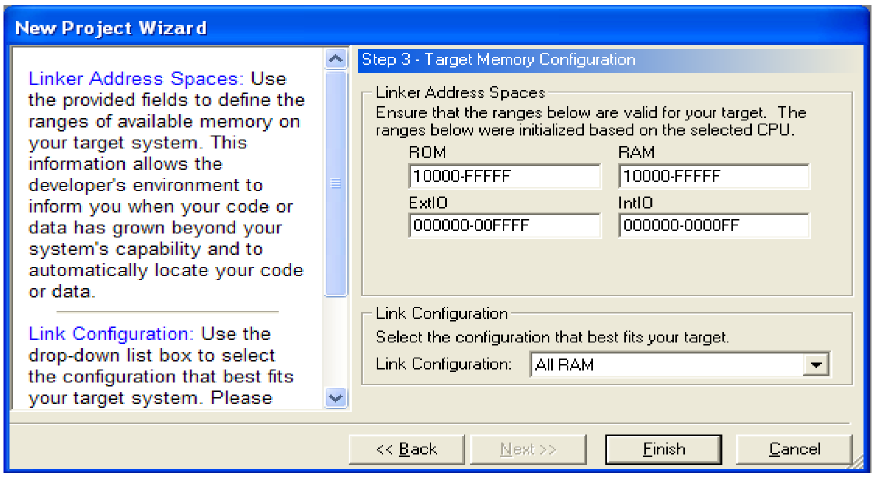
[✓] Use page erase before flashing

Target

Target Name	Location
[ ] eZ80DevPlatform_L92_Flash	ZDS Default
[✓] eZ80DevPlatform_L92_RAM	ZDS Default

Debug Tool
Current: Simulator

Then click on Next button. Now this part is important, you need to set the memory of our system. The best configuration is: Link Configuration = ALL RAM, ROM = 10000-FFFFF and RAM = 10000-FFFFF.


New Project Wizard

Linker Address Spaces: Use the provided fields to define the ranges of available memory on your target system. This information allows the developer's environment to inform you when your code or data has grown beyond your system's capability and to automatically locate your code or data.

Link Configuration: Use the drop-down list box to select the configuration that best fits your target system. Please

Step 3 - Target Memory Configuration

Linker Address Spaces
Ensure that the ranges below are valid for your target. The ranges below were initialized based on the selected CPU.

ROM: 10000-FFFFF
RAM: 10000-FFFFF
ExtIO: 000000-00FFFF
IntIO: 000000-0000FF

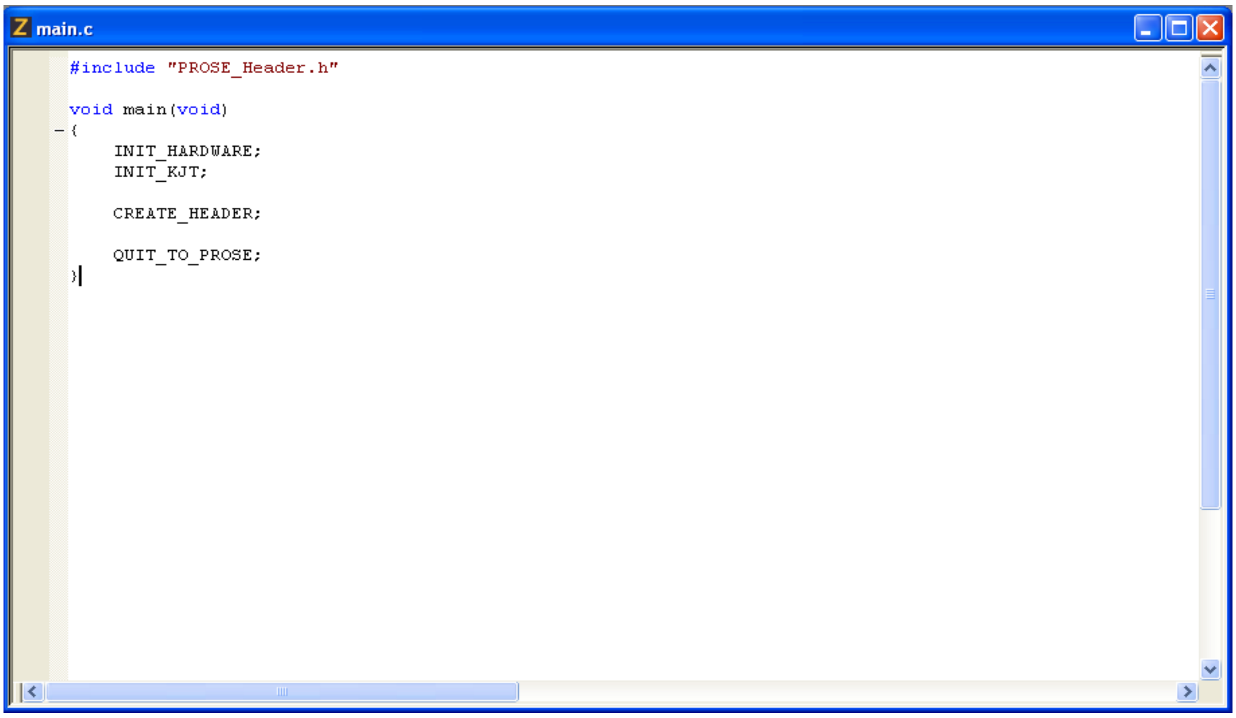
Link Configuration
Select the configuration that best fits your target.
Link Configuration: All RAM

Now press "Finish" to finish the first phase configuration.

Go to File and click "New File", an empty editing window will appear. Go to File and click Save and rename the file "main.c". Go "Standard Project Files", right click and select "Add files ti project" and select main.c. Now save the project.

Attached to this document are a header file called "PROSE_Header.h" that contains macros to interface with PROSE. Do not need to add it to your project if you copied it in the same folder of the main.c, which I suppose for this manual.

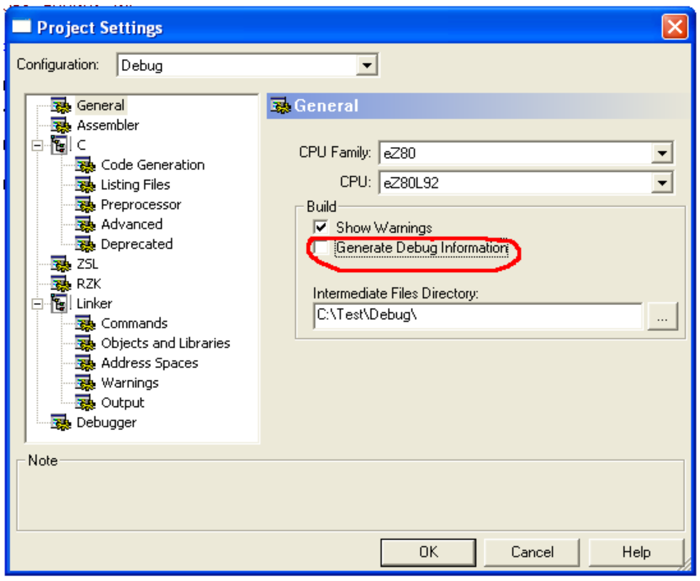
Here is the source of a minimal C program for PROSE:



```
#include "PROSE_Header.h"

void main(void)
{
    INIT_HARDWARE;
    INIT_KJT;

    CREATE_HEADER;

    QUIT_TO_PROSE;
}
```

As you can see, a fixed structure is required to generate a correct executable. First of all, the main must ALWAYS be the first function and then the first lines should be INIT_HARDWARE, INIT_KJT and CREATE_HEADER; your code goes **between** CREATE_HEADER and QUIT_TO_PROSE. Another important thing, in the main you **never** have to specify local variables in other functions or procedures you can use whatever you want.
Before completing this brief example, you have to configure some options of the project, so go on Project and click on Settings, a series of images you will see how to do:

General:



Configuration: Debug

General

CPU Family: eZ80

CPU: eZ80L92

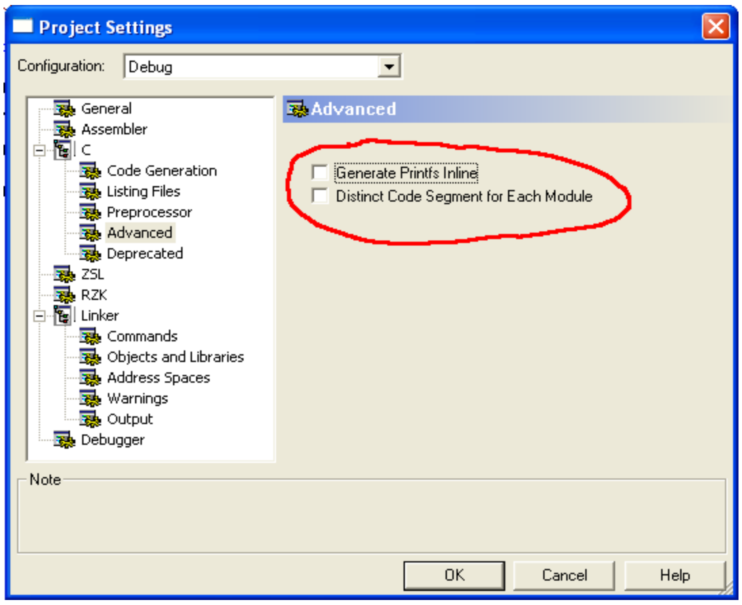
Build

[✓] Show Warnings

[ ] Generate Debug Information

Intermediate Files Directory: C:\Test\Debug\

Advanced:



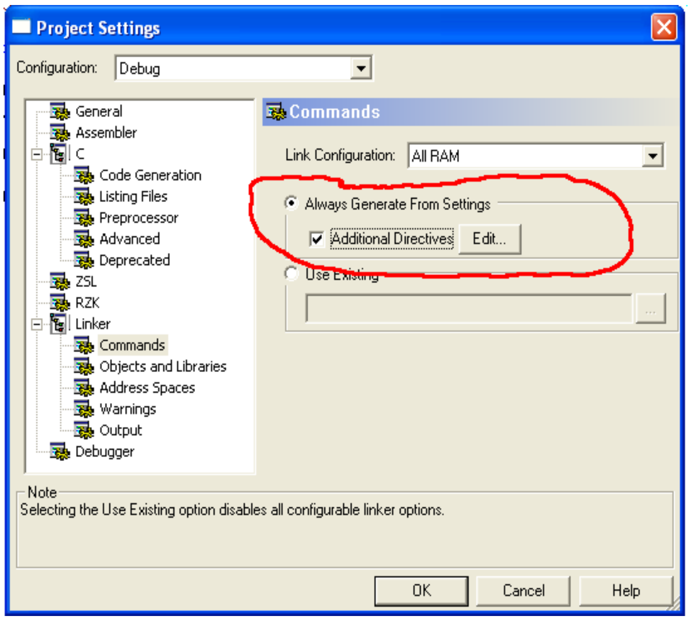
Configuration: Debug

Advanced

☐ Generate Printfs Inline

☐ Distinct Code Segment for Each Module

Commands:



Configuration: Debug

Commands

Link Configuration: All RAM

● Always Generate From Settings

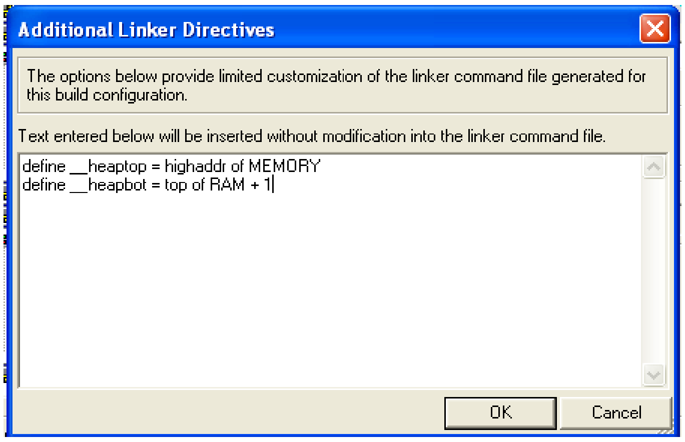
☑ Additional Directives

○ Use Existing

Note

Selecting the Use Existing option disables all configurable linker options.

Then you must specify the parameters for the heap, otherwise you can NOT use malloc, realloc ...



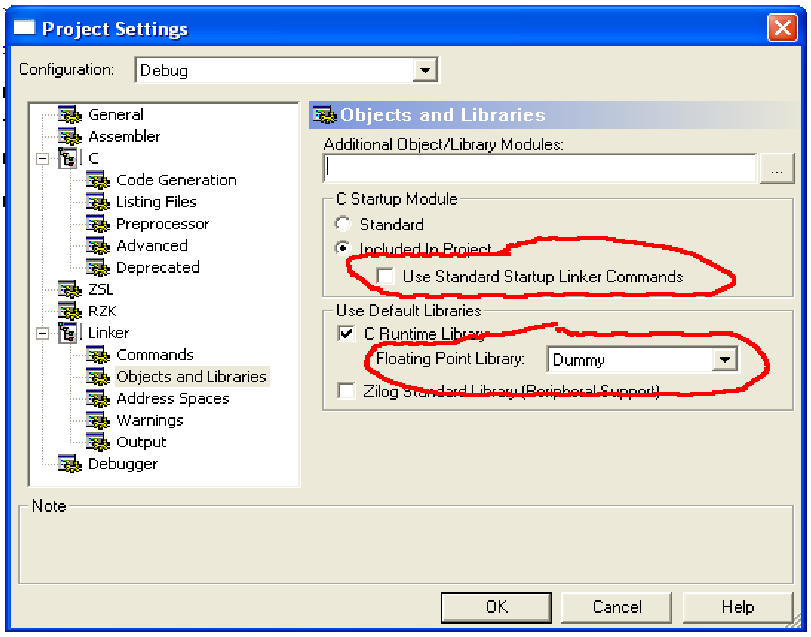
The options below provide limited customization of the linker command file generated for this build configuration.

Text entered below will be inserted without modification into the linker command file.

```
define __heaptop = highaddr of MEMORY
define __heapbot = top of RAM + 1
```

in the text box you must enter: *define __heaptop = highaddr of MEMORY* and *define* *__heapbot = top of RAM + 1*. Now you can use dinamic memory allocation.

Objects and libraries:


Project Settings

Configuration: Debug

General
Assembler
C
Code Generation
Listing Files
Preprocessor
Advanced
Deprecated
ZSL
RZK
Linker
Commands
Objects and Libraries
Address Spaces
Warnings
Output
Debugger

Objects and Libraries

Additional Object/Library Modules:

C Startup Module
( ) Standard
(●) Included In Project
[ ] Use Standard Startup Linker Commands

Use Default Libraries
[✓] C Runtime Library
Floating Point Library: Dummy
[ ] Zilog Standard Library (Peripheral Support)

Note

OK Cancel Help

Uncheck "Use standard Startup Linker Commands" and set the Floating Point Library as Dummy. Now click on OK and the sample projects will be compiled.

Now let's see how to place code and data to create a simple application. Suppose we want to create a function that prints a text on the screen, type ShowMessage("My Text");

First, we define it first the main:

```
#include "PROSE_Header.h"

char *TxtPnt;

void ShowMessage(const char *Txt);

void main(void)
{
    INIT_HARDWARE;
    INIT_KJT;

    CREATE_HEADER

    ShowMessage("Hello PROSE!!!\n\r");

    QUIT_TO_PROSE;
}

void ShowMessage(const char *Txt)
{
    TxtPnt = Txt;

    asm ("push ix");
    asm ("ld hl, (_TxtPnt)");
    asm ("ld a, kr_print_string");
    asm ("call.lil prose_kernal");
    asm ("pop ix");
}
```

Before you proceed, you must know that the development environment does not allow to pass local variables to inline assembly. So first we define a pointer that will host the local parameter of our function (char *TxtPnt).

Then we pre-declare our function and then defined it **after** the main.

The function ShowMessage executes these instructions :

1) Copy local parameter Txt in global pointer TxtPnt.

2) Before **any calls** to the Prose's kernal need to save the IX register.

3) Load in HL the value on pointer TxtPnt. Global variables in the source you wish to pass at the inline assembler are renamed by adding the character "_" before the name defined in the C source.

4) Set a to kr_print_string value and...

5) Call the kernel

6) Restore IX register.

You can also save the value of a register in a C global variable. Suppose that, after a call to the kernal, i want to save the value of the register DE in a global variable. You can do this:

```
int MyRegValue;

....
....
....

asm ("ld (_MyRegValue), DE);
```

Remember to always add the character "_" before the name of the variable and specify the variable in parentheses otherwise overwrite the **address** of a variable.

Of course, you have to use the program developed by Phil to convert the file from hex format to ezp.

This is all, always remember that the main must be the first call and can not have local variables. All functions / procedures should be inserted **after** the main. Before a call to the kernal, save IX register and after restore it.

Enzo Antonio Calogiuri.