Labo n° 5
Microprocessor Architectures [ELEC-H-473]
dsPIC33: simulation 1/2

2015–2016

Purpose

- Use a simulator and to understand its utility.
- Understand the instruction set and the assembly language, starting from the code produced by a C compiler. See the link between high-level language instructions and assembly/machine code.
- Discover the limitations of small micro-controllers.

Useful Documents

A set of useful documents can be found in the network share "Labo/ELEC-H-473/dsPIC33":
- Introduction to MPLAB
- MPLAB C30 C Compiler User's guide
- dsPIC30F/33F Programmer's Reference Manual
- dsPIC33FJXXXGPX06/X08/X10 Data Sheet
- dsPIC30F/33F CPU reference manual

1 Architecture analysis

1.1 CPU architecture

Question 1. Detail the architecture of the CPU used in this microcontroller. Focus on the CPU, not on the peripherals.

Question 2. Explain the difference between a microprocessor and a microcontroller.

Now, focus on the CPU.

1.2 Memory Addressing modes

Question 3. In the "Programmer's Reference Manual", identify the various addressing modes

Question 4. Give an example of instruction illustrating each addressing mode and instructions using multiple addressing modes, explain how they work.

Question 5. What is the size of an instruction? Is this coherent with the CPU architecture presented in the datasheet?

Question 6. Examine the notion of "format" of the instruction and show using examples that the size of the instruction introduces limitations in the range of each addressing mode and in the size of constants.

2 MPLAB IDE – Simulation workspace

You can configure your environment to suit your needs:
- Launch the MPLAB IDE.
- Open the workspace **"Simul_dsPIC1.mcw"** (File>Open Workspace) which already includes:
  - the choice of the microcontroller.
  - the configuration of the debugger in the *simulation mode* which enables you to execute and debug the code in your development environment, *without* any microcontroller hardware.

- Display the following windows in your IDE
  - C source code
  - Output for note, warning and error messages
  - Disassembly listing (right-click in the window and check "Symbolic Disassembly")
  - Memory Usage: amount of memory used for program and data
  - Stopwatch: number of cycles required to execute each instruction
  - Watch window to observe the registers and variables
  - File Register: to dump the Data Memory
- Save your workspace under another name (File>Save Workspace as) to be able to reuse it any time.

3 Project organisation

Using the MPLAB IDE, you now will compile some code for the `dsPIC33` using the C30 compiler and analyse it.

3.1 Header Files

Notice the presence of the file `p33FJ256GP710.h` in the header files. Double-click on it and read its content.

Question 7. Look at lines related to the I/O port `PORTA`, for example. Explain the utility of this header file.

Question 8. Explain the use of the adjective "volatile" in this header file.

3.2 Types of the variables

Add the source file `initvar.c` to the project and read it.

Question 9. What are the various types of variables accepted by the compiler and their min-max range? (cf MPLAB C30 C Compiler User guide). Convert the min-max range in decimal.

Question 10. Are the integer types signed or unsigned if you do not specify it explicitly?

Variable types definition ambiguity

The size of certain types of variable can change from one processor to another; for example the default size of the `int` standard can be puzzling because it depends on the size of the data bus (but not always):

- for 8-bit µP it can be 8 bits, but is often 16 bits
- for 16-bit µP it is 16 bits
- for 32-bit µP it can be 16 bits, or 32 bits

Moreover, the fact that a `char` or an `int` is `signed` or `unsigned` by default depends on the compiler and on the options of the compiler. It is always a good practice to improve the *portability* of the code *i.e.* to minimize the modifications when you want to migrate to another processor. For this reason it is wise to gather *everything which depends on the* µ*P* in a few files, so that only those well-identified files have to be modified for another processor.
In this lab, you will use the header file `Datatypes.h` to redefine the types of the variables to avoid any ambiguity.

Question 11. Add `Datatypes.h` in the header files of your project, explain the improvement given by these new definitions.

3.3 Variables initialisation

- Replace the file `initvar.c` by `variables.c`.
- Make the line numbers appear in the C source (right-click on window->Properties->C File Type ->line numbers)

– Click on “Build all” to compile and link every file of the project. Ignore the warnings for the moment.

REM : the assembly code begins with the “`lnk`” instruction which will allocate a ”stack frame” for the local variables of the function `main()`. Refer to 4.7.4 of the “dsPIC30F/33F Programmer’s Reference Manual_70157B.pdf” for the explanation of stack frames and the use of `W15` as a Stack Pointer and `W14` as a Frame Pointer.

Question 12. Observe in the assembly code the initialization of the different variables; advance in the program using breakpoints and step-by-step mode (both in C and in ASM) and observe the state of the variables in the “Watch” window (trick: add `W14` to the Watch Window); you can change the properties of the Watch Window to observe the values of the variables in different bases.

Question 13. Where are the different variables stored?

Question 14. Describe how variables are initialized? What are the difference between the different types and the addressing mode(s) .

Question 15. Observe the number of instructions necessary to initialize each variable, use the “Stopwatch” window.

Question 16. Observe attentively the values of the various variables in the “Watch” window. Do you have any comment?

Question 17. Observe attentively the values of the various variables in the “File register” window. In particular see how the `INT32U` and `INT64U` are stored and the mental effort required if you want to analyse long variables in memory. Are all variables initialized correctly?

Question 18. Pay attention to the warnings appearing in the “Output” window, are they justified?

Question 19. The code defines a global variable `glob1`. Where is it defined? initialised?

Question 20. Explain how to place this variable in a register (the grammar can vary depending on the compiler, but the keyword `register` is always used).

3.4 Variables initialisation, be careful

Replace the file `variables.c` by `variables2.c`.

Question 21. Observe the warnings and the actual results in the watch window. What is the problem? What is the difference between signed and unsigned variables ?

3.5 Variable assignations

Replace `variables.c` by `assign.c` and compile.

Question 22. Observe and comment the way variables are accessed compared to `variables.c`.

Question 23. Observe the way assignations work. Explain any unexpected result.