

St. name:

St. number:

## Assignment week 1.2:

### What language does the PIC32 speak?

#### Requirements

- A laptop or desktop computer
- MPLAB X IDE v4.0x and MPLAB XC32 C Compiler v1.4x
- Our N@Tschool entry

#### Objectives

- Understand the transfer from a programming language to instructions for a CPU.
- Understand the principle and usefulness of a stack and the stack pointer.
- Disassemble a small piece of C-code to understand how a few simple C-statements are converted to instructions for the MIPS M4K CPU core of the PIC32 microcontroller.

#### Introduction

From next week onwards, we are going to write programs for the Basys MX3 platform in the C-language. As we know from the previous lectures, the CPU sequentially handles instructions, and those instructions are stored into program memory in the form of zeroes and ones. Figure 1 shows a schematic overview of the transition from C-code to coded instructions in the microcontroller. The conversion is done by a so-called compiler, in our case the MPLAB XC32 C Compiler v1.4x.

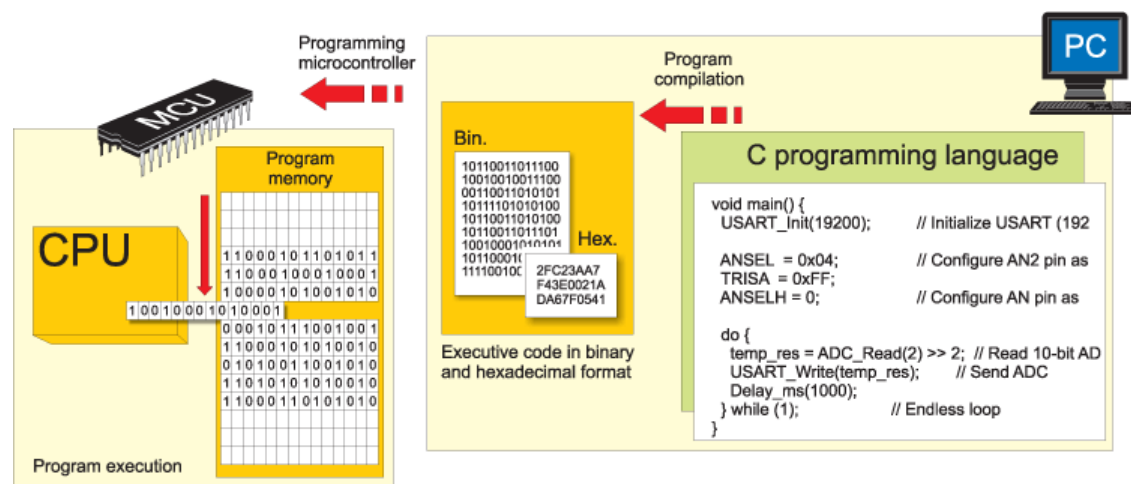


Figure 1: An overview of the transition from C-code to a program in a microcontroller, taken from [1]

#### Assembly language

The programming language that is most closely related to machine code is the assembly language. Using assembly language, a programmer can write the instructions that he wants to

St. name:

St. number:

have performed by the processor in human-readable statements. Figure 2 shows how the abstraction level in writing code varies for the most common languages in embedded systems. You can see that assembly is closest to the machine code, and that the C language is more abstract than assembly. As abstraction levels increase, the compiler becomes more complex, and the recyclability of code also increases. The statements in assembly language are very closely related to the instructions of the processor, and a design made using a graphical programming environment is often easily portable to many different processors.

## Increasing Levels of Software Abstraction

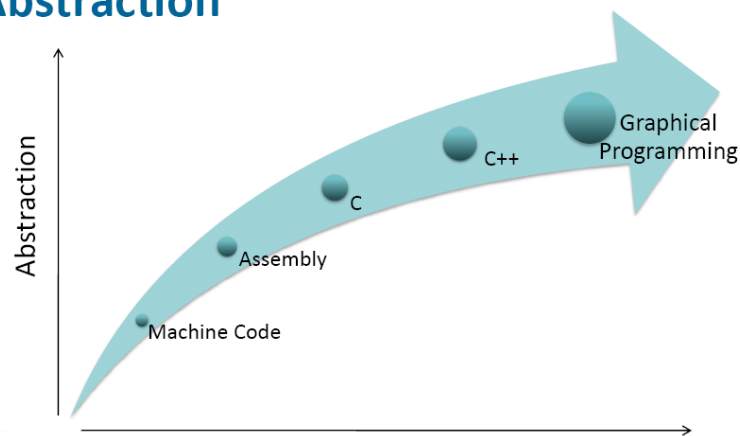


Figure 2: Overview of increasing levels of software abstraction, using a selection of programming languages

### Disassembly

A disassembler takes a piece of machine code and translates it into assembly code. As humans have difficulty finding patterns in zeroes and ones, the disassembler translates the instructions into words and decimal or hexadecimal values. We will use the disassembler of MPLAB to see how our C-code is converted to machine language for the PIC32 microcontroller.

### Registers of the MIPS M4K core

When we look at the disassembly code, we find operations, such as `addui`, `sw` and `jr`. We can also differentiate register names, such as `v0` and `s8`. The MIPS M4K core of the PIC32 microcontroller has 32 general purpose registers (how many bits do we require to code that amount of registers?). The designer of the MIPS core suggests to use the 32 registers in a specific way, which is given in Stack and **stack pointer**

When compiling a programming language, most of the times a so-called stack is used for implementing function calls. A stack is a piece of volatile memory that is reserved for keeping track of e.g. function arguments and return values. It is a last-in first-out (LIFO) queue. Read the explanation on the following website for a good understanding of stack use:

[https://www.youtube.com/watch?v=\\_8-ht2AKyH4](https://www.youtube.com/watch?v=_8-ht2AKyH4)

The stack pointer thus holds the address of the top of the stack, and is stored in register 29 of the MIPS general purpose registers, `sp`

St. name:

St. number:

---

**Table 1.** In the disassembly listing we will find registers with the same names like *zero*, *sp* and *ra*, and starting with *v* and *s*, which can be found in the table. The PIC32 compiler thus sticks with the recommendation of the MIPS manufacturer.

### Stack and stack pointer

When compiling a programming language, most of the times a so-called stack is used for implementing function calls. A stack is a piece of volatile memory that is reserved for keeping track of e.g. function arguments and return values. It is a last-in first-out (LIFO) queue. Read the explanation on the following website for a good understanding of stack use:

<https://www.youtube.com/watch?v=8-ht2AKyH4>

The stack pointer thus holds the address of the top of the stack, and is stored in register 29 of the MIPS general purpose registers, *sp*

Table 1: Overview of the suggested use for the 32 general purpose registers of the MIPS M4K core, taken from [2]

Name	Number	Function	Preserved accross a function call?
\$zero	\$0	permanently 0	n/a
\$at	\$1	assembler temporary (reserved)	no
\$v0-\$v1	\$2-\$3	values for function returns and expression evaluation	no
\$a0-\$a3	\$4-\$7	function arguments	no
\$t0-\$t7	\$8-\$15	temporaries	no
\$s0-\$s7	\$16-\$23	saved temporaries	yes
\$t8-\$t9	\$24-\$25	temporary	no
\$k0-\$k1	\$26-\$27	reserved for OS kernel	no
\$gp	\$28	global pointer	yes
\$sp	\$29	stack pointer	yes
\$fp/\$s8	\$30	frame pointer	yes
\$ra	\$31	return address	n/a

### Example

Have a look at the example shown on the website:

<http://www.johnloomis.org/microchip/pic32/calc/calc.html>

The example shows how the disassembly listing can be used to understand how the translation to machine code is being made. Using the information provided in the earlier text, and the instruction set document for the MIPS M4K (available in the Datasheets folder of our

---



**Fontys**

Hogeschool Engineering



St. name:

St. number:

---

N@Tschool entry, or via [3]), we can understand the contents of the example. In the assignment, we will do something similar for a program with a function call. You will analyze the disassembly listing.

## Assignment

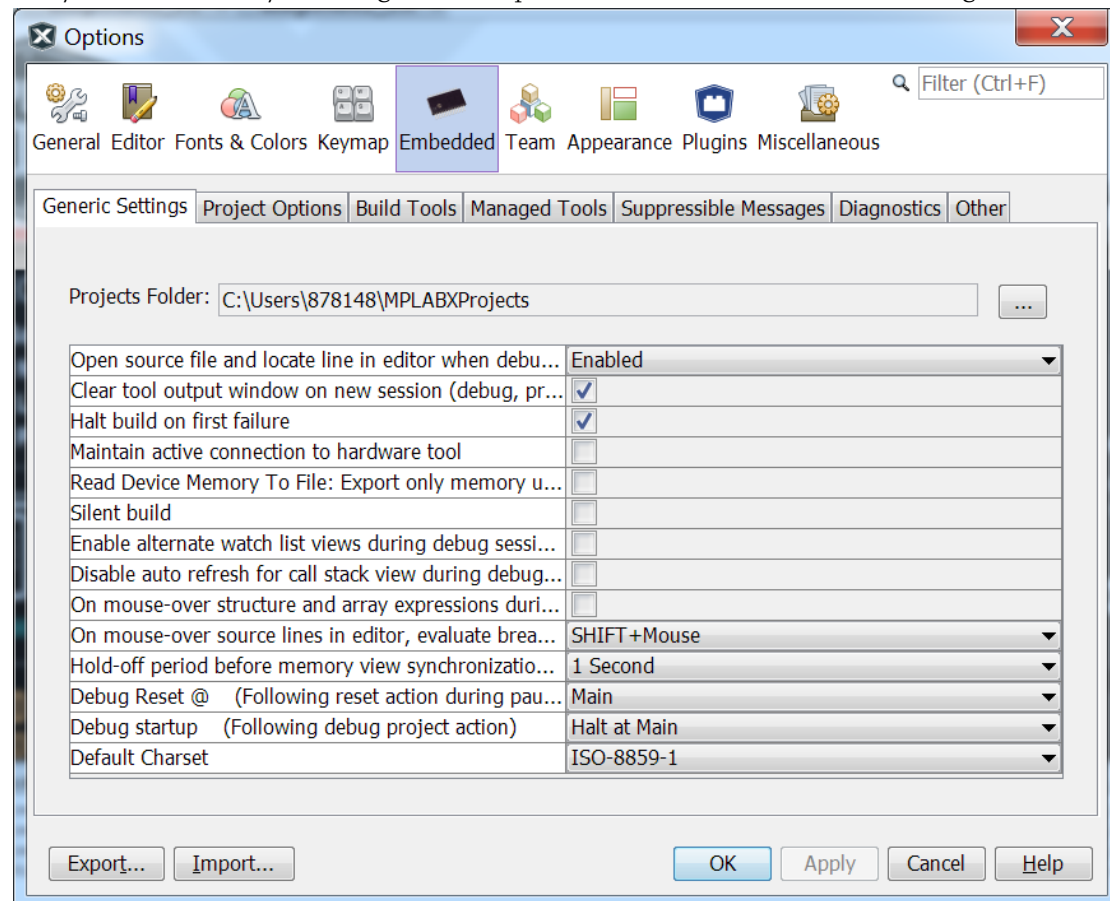
Take the 6 steps given below, put the results into a report and hand the report in before the next theory class.

St. name:



St. number:

### Step 1

- Download the file *assignment1\_2.X.zip* from our N@Tschool entry.
- Unzip the file to your embedded systems folder (e.g. F:\es\). The directory *assignment1\_2.X* will be automatically created from the zip-file.
- Open the file *assignment1\_2.X*. The program environment has been prepared for you. In a future lab you will learn how to set up a program yourself.
- Make sure that you configured the debugger to halt the program execution at the main: you can do this by selecting Tools→Options → Embedded-> Generic Settings



### Step 2

- Run the compiler by pressing the Clean and Build Project button .
- Debug the program by pressing the Debug button .
- The program will stop at the beginning of the main loop in case that you instructed the debugger to halt at the main.
- Have a look at the disassembly listing, by selecting Window→Debugging → Disassembly Listing.
- Analyze the disassembly listing.

### Step 3

St. name:

St. number:

- 
- Copy the disassembly listing into a new text file, and provide a comment for each instruction, in the same way as given in the example. Attach the listing to your report.

**Step 4**

- Have a look at the Watch window (Window→ Debugging → Watch if it is not available for you). What value does integer c have?

**c** =

**Step 5**


- Change the addition operator (+) in the `addnumbers` function into a multiplication operator (\*).
- Recompile by pressing the Make button (F10) .
- Which MIPS instruction do you now find for the multiplication operator?
- Do the same for division (/), the or operator (|) and the logical shift left operator (<<), and fill out Table 2.

Table 2: MIPS instructions used for performing operators

operator	C-symbol	MIPS instruction
addition	+	
multiplication	*	
division	/	
or		
logical shift left	<<	

**Step 6**

- Have a look at the file `assignment1_2.hex` in the bin directory. This file contains the hexadecimal values for the machine code that will be programmed into flash memory of the microcontroller.

**References**

- [1] <https://www.mikroe.com/>
  - [2] [http://wiki.osdev.org/MIPS\\_Overview](http://wiki.osdev.org/MIPS_Overview)
  - [3] <http://www.cs.tau.ac.il/~afek/MipsInstructionSetReference.pdf>
-