Microprocessor-Based Systems (E155)

J. Spjut and M. Spencer **Fall 2014**

Lab 4: Microcontroller Trainer

Requirement

1) *Get acquainted with the MPLAB IDE software.*

2) *Get acquainted with the PIC32MX675F512H and its assembly language.*

Discussion

You will be using the PIC32MX675F512H for Labs 4-7. In this lab you will get acquainted with the chip, learn how to use its accompanying software and programmer, and write simple assembly language programs.

PIC Assembly Language

To familiarize yourself with the PIC32, do the following steps:

a) Read Sections 4.0-4.1 from the PIC datasheet on the PIC's memory organization, and skim the PDF on the MIPS instruction set.

b) Read the "Getting to Know your PIC32" in Appendix A of this lab manual and try the examples to test that your PIC is working properly.

c) Write a program that, given five 32-bit signed numbers on the stack, will find the largest number and write it to Port D (to display the bottom 8 bits on the LEDs).

d) Write a program that will sort 12 32-bit signed numbers found on the stack. The sorted results should be stored in the same block of memory with the smallest number at the lowest position on the stack (largest address). Download your compiled code to the PIC and run it.

Hints

- You can define multiple labels within the `.ent main / end` block rather than defining separate functions for each place your program can jump to. This is helpful in creating loops within a function. See code in the appendix for an example.

- Be sure you understand how branch and jump instructions are executed. When in doubt, put a `nop` in the branch delay slot.

- The latest version of the MPLAB development environment appears to corrupt projects fairly often, but does not damage the source files. If your project gets corrupted, make a new one and include the .S file and bootloader scripts again.

- Another issue with the latest MPLAB is that it doesn't allow assembly registers to use $ to name them when the file contains the line, "#include <P32xxxx.h>". If you want to use $ for your register names, then you should remove that include line from your file, perhaps by commenting it out.

- MPLAB-X is the new version of MicroChip's development software. While this lab assumes the older MPLAB software, everything should be possible in the newer software, but be found in different places. If you feel like revising the lab for MPLAB-X, contact Prof. Spjut. It will take significantly more effort to use MPLAB-X.

What to Turn In

You may turn in an abbreviated lab report for this lab. Well commented code is extremely important when writing assembly and will play an important role in the lab grading. Your lab write-up should include:

- A listing of your program that finds the greatest number.
- A listing of your program that sorts numbers.
- Explicitly state the test cases you used and the output of the tests on each program. Be sure your tests would convince a skeptic that your algorithm works.
- How many hours did you spend on the lab? Any comments, suggestions, or complaints about the assignment? This will not count toward your grade.

Appendix A: Getting to Know your PIC32

A Few Bits of Background

The PIC32 series of microcontrollers implement an instruction set compliant with the MIPS32 Release 2 standard. The assembly language should look familiar to anyone who has taken E85. Like other MIPS processors, the PIC32 has working registers designated for specific purposes. The functionality of these registers is summarized below.

Register	Purpose
$zero	Constant value 0
$at	Temporary register reserved for multi-cycle instructions
$v0-$v1	Values returned from functions
$a0-$a3	Arguments passed to functions
$t0-$t9	Temporary working registers
$s0-$s7	Saved temporary registers
$k0-$k1	Reserved for OS kernel
$gp	Global pointer
$sp	Stack pointer
$fp	Frame pointer
$ra	Return address

Look through the MIPS32 Instruction Set document on the course website to get a sense for the instructions at your disposal. Note that the Core Extend and Coprocessor 1 / 2 instructions are not implemented by the PIC.

Software Tools:

We will be using the MicroChip MPLAB Integrated Development Environment (IDE) software for writing assembly code and programming the PIC. This software includes an assembler, a simulator, and a hardware debugger. The simulator and hardware debugger have a very similar interface, so be sure you know when you are simulating a program and when you are actually executing it on the physical hardware.

Creating a New Project:

Start the MPLAB software using the "MPLAB IDE" shortcut. Create a new project by selecting *Project* → *New*. Name your project "**leds_xx**" (where **xx** are your initials) and put it in a folder on your Charlie account such as H:\e155\lab4_xx. (Be sure to use backslashes rather than forward slashes in the path name.) This will create a "leds_xx.mcp" file in that folder.

First make sure the software knows which PIC you are working with by selecting "**PIC32MX675F512H**" from the *Configure* → *Select Device* menu. The box has a list of programming tools that are compatible with the PIC chips. Check that MPLAB ICD3 has a green light and MPLAB SIM has a yellow or green light before clicking OK.

Also, set the tools for the PIC32. Under *Project* → *Select Language Toolsuite*…, be sure Microchip PIC32 C-Compiler Toolsuite is selected.

Next you need to enter some code to run. Select *File* → *New* to create a new source file, and enter the following lines of code:

```
/* leds.S
   Written <date> by <your_name>@hmc.edu
   Test PIC by turning on LEDs         */

#include <P32xxxx.h>

#  Define constants
#define  LEDS  0xA5

#  Define functions
.global main

#  Compiler instructions
.text    # store the code in the main program section of RAM
.set noreorder # do not let the compiler reorganize your code

# Main program

.ent main      # Start function block
main:
   la    $t0, TRISD  # Load the address of TRISD into $t0
   addi  $t1, $0, 0xFF00
   sw    $t1, 0($t0) # TRISD = 0xF00 (bottom 8 bits outputs)
   addi  $t1, $zero, LEDS  # $t1 = LEDS (LEDS + 0)

write:   # This is a label you can jump to
   la    $t0, PORTD  # Load the address of PORTD into $t0
   sw    $t1, 0($t0) # PORTD = $t1
   j     write       # Jump back to write
   nop
.end main      # End function block
```

Comments and Constants:
MIPS assembly supports two types of code comments. A # character will comment out any text after it on the same line, and /* block comments */ can span multiple lines. Be careful not to confuse line comments with preprocessor directives. Statements like

#define and #include are not comments, but instructions interpreted by the compiler used to simplify programming. The line

```
#define  LEDS  0xA5
```

defines a constant symbol, `LEDS`, that can be used elsewhere in the program. At build time, the compier's preprocessor will replace every instance of `LEDS` with `0xA5`. Use this feature to avoid magic numbers in your code. The #include statement includes a file at compile time, in this case a file that defines names for the hardware on the PIC (PORTD, TRISE, etc).

The Main Loop:
Every PIC program must contain at least one function, named main. The `.global main` instruction tells the compiler that a function named main exists. The `.ent main / .end main` lines contain the actual code of the function. Additionally, a code label `main:` is placed at the beginning of the function. Labels can be used as the target of jump and branch instructions and can be placed anywhere in your code. A label `write:` is used later on to form a simple loop.

Saving your Source Code:
Save the file as "leds_xx.S" in the project directory. Note the capital 'S' extension, which tells the compiler to interpret preprocessor directives like #define in assembly code (such directives are generally used with C). Insert the source file into your project by using *Project → Add Files* to Project and selecting leds_xx.S. You can save your changes to the project by selecting *Project → Save Project*.

Disabling Default Startup Code:
By default, the PIC32 compiler loads a lengthy startup routine into boot flash memory. This code initializes the stack and global pointers, clears sections of memory, sets up customized interrupt vectors and RAM functions. Some parts of this code take many iterations to complete, making stepping through it a long and tedious process. For assembly programming, all we need the startup routine to do is initialize the stack pointer and jump to the `main` function. Copy the given crt0.S file, containing a shortened startup routine, and elf32pic32mx.ld, a modified linker script, into your lab4 directory and add the files to the project. Next, select *Project → Build Options… → Project* and open the '*PIC32 Suite*' tab. Check the box for '*Don't link startup code*' under '*Target Type*' so the default startup code does not conflict with our modified version. Be sure to read through the original crt0.S initialization code at C:\Program Files (x86)\Microchip\MPLAB C32 Suite\pic32-libs\libc\startup and understand what it does, as it is useful to use when running compiled C code.

Building your Project:
Now we are ready to translate the assembly code into machine code. You can either select *Project → Build All*, or you can use *Ctrl+F10*. A window will pop up and a couple green bars will scroll, but they will probably be gone before you can read what they say.

When the code has finished with assembly, a window labeled Output will appear, and you should see the message **BUILD SUCCEEDED** at the bottom of the compiler output.

If you get any warning or error messages, go back and fix your code. When you are working with more complicated assembly language code, error or warning messages will appear in this box. Double-clicking an error message will take you to the offending line of code.

If for some reason MPLAB can't find its assembler (the paths seem to occasionally get messed up), select *Project* → *Set Language Tool Locations*… Under the Microchip PIC32 C-Compiler Toolsuite, select *Executables*. Then click on *MPLAB ASM32 Assembler* (pic32-as.exe). Click the browse button and browse to C:\Program Files (x86)\Microchip\MPLAB C32 Suite\bin\pic32-as.exe. Similarly, set the paths for pic32-gcc.exe, pic32-ar.exe and pic32-ld.exe (all in the same directory).

Simulating:

Now we are ready to start debugging. Chose MPLAB SIM from the *Debugger* → *Select Tool* menu. This will simulate your code and allow you to debug it before actually running the code on the PIC.

While you are simulating, it is helpful to see what is going on going on in the chip's memory. The View menu lists a number of windows to watch the activity. The Memory window displays the contents of all memory addresses on the PIC. This is useful to see the instructions the machine is actually running, and to see the contents of RAM. You can change the contents of memory simply by editing the values in the 'Opcode' field. You may also find the Special Function Registers window helpful, and the Watch window is convenient for monitoring only the registers you want to look at.

Bring up the *Watch* and *Memory* windows. Add TRISD, PORTD, t0, and t1 as registers in the Watch window. At the top of the window will be two buttons and two pulldown menus. Click on the first pulldown menu and type in "TRISD". Use the ADD SFR button to the left of the menu to display its value. Do the same for PORTD.

Now we are ready to debug some code. Chose *Debugger* → *Reset* → *Processor Reset* or press F6. A green arrow will appear in your Memory window, pointing to an instruction in Boot Flash Memory. This green arrow tells you where you are in your program. Chose *Debugger* → *Step Into* or press F7. The green arrow will move to the next line. After a value is changed in memory, whether in the Memory window or Watch window, the value will appear in red. This is helpful for pinpointing the effect of commands on particular registers while you are stepping through assembly code. Watch the register values change as you step through the program. Finally, the value 0xA5 will be written to PORTD.

Reset the processor and repeat these steps until you are comfortable with what you are observing and can relate it back to the crt0 startup code and to your program. Remember

that the la pseudoinstruction in your program assembles into two real instructions: lui and addiu.

Once you have verified that your code works, you are ready to run it on the PIC.

Running your Code on the PIC:

Apply power to your board from the bench power supply. The PIC receives the same clock as the FPGA. This is a great time to erase your PROM for the FPGA (using the Quartus programmer and the directions from Lab 1) to ensure that FPGA outputs don't contend with PIC outputs in this or future labs.

Setting Configuration Bits:

Next, go to *Configure* → *Configuration Bits*. This sets some of the hardware options on your PIC microprocessor. You may have to drag some of the columns to make them large enough to read. Uncheck the box labeled Configuration Bits set in code. The clock source is the external clock coming from the oscillator to PIC pin OSC1. Therefore, set the **Oscillator Selection Bits** configuration option to "**Primary Osc (XT,HS,EC)**." The PIC supports several different types of external clock sources. Set the **Primary Oscillator Configuration** to "**External Clock Mode.**" This tells the PIC to expect a clock signal input on OSC1, rather than attempting to excite a crystal or other type of passive resonator. Also check that the secondary oscillator is disabled. Most peripherals on the PIC32, such as the UART, run on the peripheral clock. For simplicity, set **Peripheral Clock Divisor** to "**Sys_Clk/2**" to run the peripherals at 40 MHz / 2 = 20 MHz. The PIC32 can be programmed through multiple sets of pins depending on application. For the debugger to talk to the PIC on this board, ensure the **ICE/ICD Comm Channel Select** is set to use pins **PGC1/PCG1.** Configuration bits are saved in each project file, so you will have to set them for each new project you create.

The full changes are listed below:

Address	Value	Field	Category	Setting
1FC0_2FF0	FFFFFFFF	USERID		
		FSRSSEL	SRS Select	SRS Priority 7
		FMIIEN	Ethernet RMII/MII Enable	MII Enabled
		FETHIO	Ethernet I/O Pin Select	Default Ethernet I/O
		FUSBIDIO	USB USID Selection	Controlled by the USB Module
		FVBUSONIO	USB VBUS ON Selection	Controlled by USB Module
1FC0_2FF4	FFFFFFFF	FPLLIDIV	PLL Input Divider	12x Divider
		FPLLMUL	PLL Multiplier	24x Multiplier
		UPLLIDIV	USB PLL Input Divider	12x Divider
		UPLLEN	USB PLL Enable	Disabled and Bypassed
		FPLLODIV	System PLL Output Clock Divider	PLL Divide by 256
1FC0_2FF8	FFFFDCDA	FNOSC	Oscillator Selection Bits	Primary Osc (XT,HS,EC)
		FSOSCEN	Secondary Oscillator Enable	Disabled
		IESO	Internal/External Switch Over	Enabled
		POSCMOD	Primary Oscillator Configuration	External clock mode
		OSCIOFNC	CLKO Output Signal Active on the OSCO Pin	Disabled
		FPBDIV	Peripheral Clock Divisor	Pb_Clk is Sys_Clk/2
		FCKSM	Clock Switching and Monitor Selection	Clock Switch Disable, FSCM Disabled
		WDTPS	Watchdog Timer Postscaler	1:1048576
		FWDTEN	Watchdog Timer Enable	WDT Enabled
1FC0_2FFC	7FFFFFF6	DEBUG	Background Debugger Enable	Debugger is enabled
		ICESEL	ICE/ICD Comm Channel Select	ICE EMUC1/EMUD1 pins shared with PGC1/PGD1
		PWP	Program Flash Write Protect	Disable
		BWP	Boot Flash Write Protect bit	Protection Disabled
		CP	Code Protect	Protection Disabled

Uploading:
Once these options have been set, switch to running your hardware on the chip itself by selecting "*MPLAB ICD3*" from the *Debugger* → *Select Tool* menu. If the Setup Wizard comes up, select *USB* for the Com Port and "*Target has its own power supply*." Also check the boxes to automatically connect with MPLAB ICD3 and download the firmware. Otherwise check these options under the *Debugger* → *Settings* dialog.

In the Output window, a new tab labeled MPLAB ICD3 will appear. You should see the following messages:

```
MPLAB ICD 3 detected
Connecting to MPLAB ICD 3...
Firmware Suite Version..... 01.26.05
Firmware type.....................PIC32MX
MPLAB ICD 3 Connected.
Target Detected
Device ID Revision = 04300053
```

If you see a "ICD3Err0040: The target device is not ready for debugging" error, double-check your configuration bits settings, in particular the ICD Comm Channel selection. MPLAB sometimes forgets your configuration bits settings, which can lead to this error. If you get a "ICD3Err0045: You must connect to a target device to use MPLAB ICD3" message, you probably forgot to turn on the power supply or connect the board to the debugger. Turn on power and choose *Debugger* → *Connect* to try again. If that doesn't work, check the solder joints on the ICD connector. If you see "ICD3Err0086: Target Device ID does not match expected Device ID," check under *Configuration* → *Select Device*… that you have selectd **PIC32MX675F512H**. If it still doesn't work, you may have damaged your PIC. If there are still issues, the ICD hocky puck is probably damaged. Several get blown out each year for no obvious reason; please treat them with care. Try a different computer. If that works, you've confirmed the bad ICD. Label it "Bad" with a piece of tape and put it in the supply cabinet so nobody else has the problem. Let the instructor know.

NOTE: On Windows XP, if you unplug the USB cable for the ICD and then plug the same or a different one back in, you may confuse the USB driver. You will then have to log out and back in before MPLab will recognize the ICD again. We do not yet know how robust the drivers are under Windows 7.

You are finally ready to run your code on the PIC. Choose "*Program*" from the Debugger menu. After you have downloaded your code, you will be able to step through it just as you did with the simulator. View the Program Memory to watch the program counter step through the machine language as well. While you are in debug mode, you can change values in the PIC's data memory by changing the values in Memory or Watch. Modify the instruction that specifies the value to write to PORTD and change the value. Verify that the new value is written to the LEDs after stepping through the next loop iteration. If something goes wrong, re-program the PIC.

As you step through your program, MPLAB will highlight values in red that it has changed in the last step. Stepping is fairly slow because the ICD sends a large amount of information back and forth after each step. Having many windows open from the View menu slows stepping even more. If you step to the end of the code, you may get a bunch of spurious "stepping target" messages and have to quit MPLAB.

If you want to run your entire program without stepping through it, choose Run (F9) and your code will run at full speed. You need to Halt the program (F5) before you can reset or re-program the PIC.