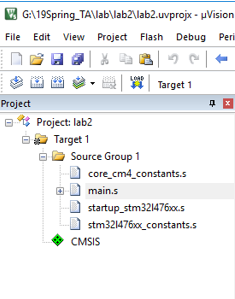
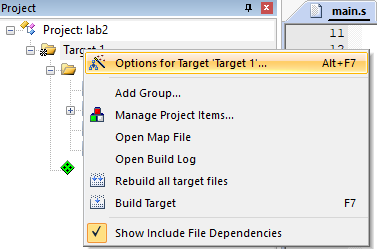
# Memory, Register and Debugging

Jingtong Hu

January 10, 2019

## Step 1: Project Creation

Create a new program as you did in the lab1 with the provided files in the Courseweb. **Remember to set the options of target (as you did in the lab1)** . And then build it and load it to your board.

## Step 2: Debugging

If you want to monitor the memory and register, you need to get into the debugging mode. The following is some basic knowledge of debugging.

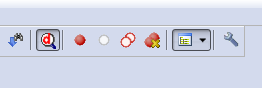
## Software *vs* Hardware Debug

There are two methods to debug your program: software debug and hardware debug. By using the software debug, you do not have to have the hardware board to debug a software program. However, the hardware debug requires you to connect the board to the computer. **In lab2, we will use hardware debug**.

|  |  |
| --- | --- |
|  |  |
| Selecting **software** debug | Selecting **hardware** debug |

## Debug Control

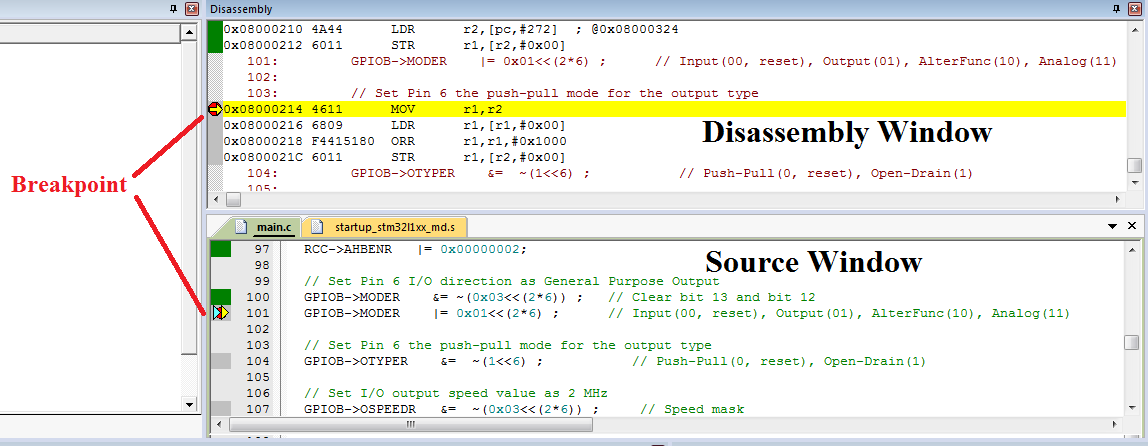
* You can program the STM32 flash by clicking the LOAD button  (You need to do it on step 1).
* Click the debug button  to start the debug and click it again to exit the debug. You can use the breakpoint button  to set a break point in either disassembly or source windows.
* STM32 allows up to six breakpoints during hardware debugging. When a program stops at a breakpoint, the corresponding instruction has not been executed yet.
* **If the disassembly window is in focus, the debugger executes assembly instructions step by step. If the source window is focused, the debugger then steps through the source lines instead.**



The following table summarizes commonly used debug control buttons.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| Start Debug | Set a Breakpoint | Run | Stop Debug | Step In | Step Over | Step Out |

* **Run**: Continues the execution from the current position until you click ***Stop*** or the program is paused by a breakpoint.
* **Step In**: Execute one step and enter the function if the current step calls a function.
* **Step Over**: Execute one step and run the function all at once if the current step calls a function.
* **Step Out**: Execute until the current function returns.

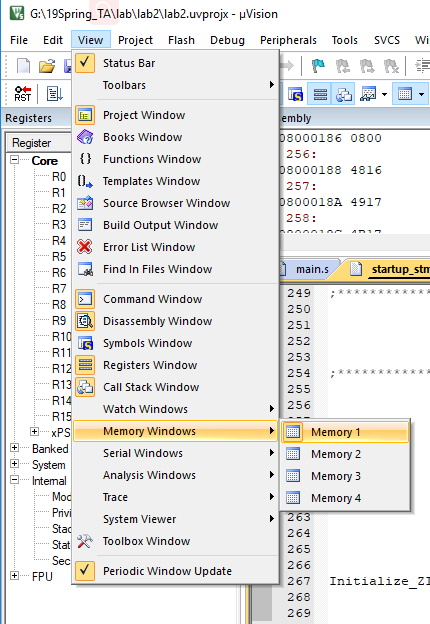


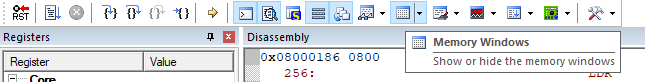
In step 2, Click the debug button  to get into debugging mode.

**Step3: Monitoring Memory**

**Memory Window**

The memory window is used to view the memory content in real time. The memory window can be found from menu View->Memory Windows->Memory 1 or you can find the memory window button in the toolbar.





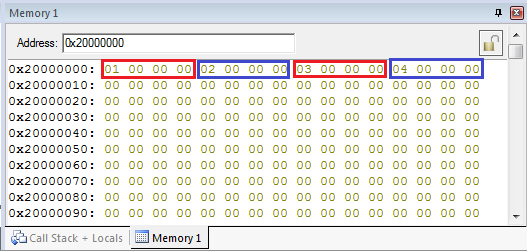
**By default, the address of data memory (RAM) starts at 0x2000\_0000. This is specified in the scatter-loading file (\*.sct).**

The following assembly program (main.s) defines and allocates an array of four words, and write the result of addition to memory (**the address of variable value**).

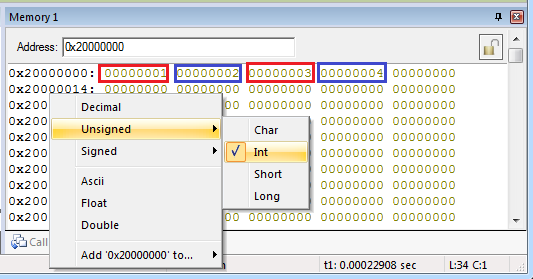
|  |
| --- |
| INCLUDE core\_cm4\_constants.s ; Load Constant Definitions  INCLUDE stm32l476xx\_constants.s  AREA main, CODE, READONLY  EXPORT \_\_main ; make \_\_main visible to linker  ENTRY  \_\_main PROC  MOVS r1, #0x01  MOVS r2, #0x02  ADDS r3, r1, r2  LDR r0, =value  STR r3, [r0]  stop B stop ; dead loop & program hangs here  ENDP    ALIGN  AREA myData, DATA, READWRITE  ALIGN  array DCD 1, 2, 3, 4  value DCD 0  END |

**(1) Watch memory content before execution**

Set a breakpoint in the line “MOVS r1, #0x01”, and click “Run” to execute until this line. When we type the memory address 0x20000000, we can see the content. **The memory content is displayed in bytes (Char) by default**.



By right click, we can select different display format. For example, we can show the content as unsigned integers.



**The value of variable “value” is shown at address 0x20000010**. Before execution, the value of this variable is 0.

## (2) Watch memory content after execution

Set another breakpoint in the line “MOVS r0, #0x01”, and click “Run” to execute until this line. After execution, the result of “1+2” is written to variable “value” (0x20000000), and the value of corresponding address is changed from 0 to 3.

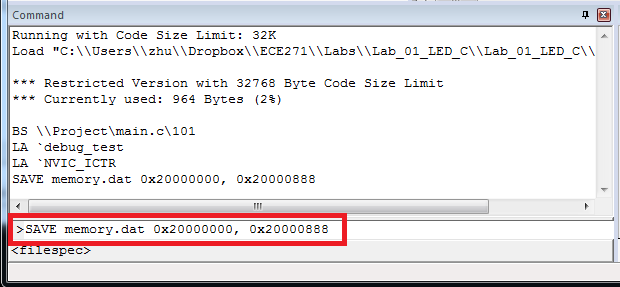
## (3)Save Memory Content to a File

In the debug environment, run the following command in the Command Window:

**SAVE <filename> <start address>, <end address>**

This allows you to perform data analysis in other software tools, such as Microsoft Excel and Matlab. The output is saved in Intel HEX format (the location of this file is under your project directory by default).

For example, SAVE memory.dat 0x20000000, 0x20000888



**Exercise**

In previous step, the value of variable “value” in memory 0x20000010 has been changed to 3. **Save memory content to the file memory.dat after execution, and upload it to Courseweb.**

**Step 4: Monitoring Processor Registers**

Before monitoring the processor register, you need to learn some basic knowledge about processor register (you can refer to the content in the next page). In lab2, you need to monitor the NZCV flags in xPSR register in the debugging mode with the codes provided (main.s).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Core Registers:   * Program counter (PC) r15 holds the memory address (location in memory) of the next instruction to be fetched from the instruction memory. * Stack point (SP) r13 holds a memory address that points to the top of the stack. SP is a shadow of either MSP or PSP. * xPSR (Special-purpose program status registers) is a combination of the following three processor status registers: * Application PSR * Interrupt PSR * Execution PSR  |  |  | | --- | --- | | N | Negative or less than flag (1 = result negative) | | Z | Zero flag (1 = result 0) | | C | Carry or borrow flag (1 = Carry true or borrow false) | | V | Overflow flag (1 = overflow) | | Q | Q Sticky saturation flag | | T | Thumb state bit | | IT | If-Then bits | | ISR | ISR Number ( 6 bits ) |   System:   * Base priority mask register (BASEPRI) defines the minimum priority for exception processing. * Priority mask register (PRIMASK) is used to disable all interrupts excluding hard faults and non-maskable interrupts (NMI). If an interrupt is masked, this interrupt is ignored (i.e. disabled) by the processor. * Control register (CONTROL) sets the choice of main stack or process stack, and the choice of privileged or unprivileged mode. * Fault mask register (FAULTMASK) is used to disable all interrupts excluding non-maskable interrupts (NMI). |

**(1) Watch C (Carry) flag**

In the part of monitoring memory, you have got into debugging mode. Now click debug button  again to exit. Delete the breakpoints you set before and Set some new breakpoints. The locations of these breakpoints are as follows:

1. line21: MOV r1, #0xFFFFFFFF
2. line 25: MOV r1, #0xFFFFFFFF
3. line 29: MOV r1, #0x7FFFFFFF
4. line 32: stop B stop

After finishing setting the new breakpoints, click debug button  to get into debugging mode. Click run to execute until line 21. Note that at this moment, the NZCV flags are all cleared (value is 0). Click run to execute until line 25 and you can see the C flag is set.

**(2) Watch Z (Zero) flag**

Click run to execute until line 29 and you can see the Z flag is set.

**(3) Watch V (Overflow) and N (Negative) flags**

Click run to execute until line 32 and you can see the V and N flags are set.