Computer Architecture Course: IT089IU

International University – VNU HCM Date: March 2021

Dr. Le Hai Duong & Dr. Ly Tu Nga Time: 6 hours

**Laboratory Session 1**

**Introduction to Qtspim and Code composer Studio**

ObjectivesAfter completing this experiment, you will be able to:

-How to use QtSpim for simulation ([Link](https://www.youtube.com/watch?v=r8WcV7AiLXs)) and Code Composer Studio for implementation ([Link](https://www.youtube.com/watch?v=11lsNYW7zkw)).

-How to understand, explain, and analysis the sample codes in simulation andreality.  
 Activities -Sample codes.

-Demo Kit MSP430 (devices are supported by Instructor)

Materials Needed

-Read carefully: Assemblers\_Linkers\_and\_the\_SPIM\_simulator.pdf, QtSpim-Tutorial.pdf, MIPS\_ short.pdf, msp430g2553\_doc2.pdf, MSP\_doc1.pdf

# **I. Introduction to SPIM (70pts)**

1. **SPIM**

Spim is a self-contained simulator that runs MIPS32 programs. It reads and executes assembly language programs written for this processor. Spim also provides a simple debugger and minimal set of operating system services. Spim does not execute binary (compiled) programs.

Download and install the newst version of Spim called QtSpim from: <http://spimsimulator.sourceforge.net/>

1. **QtSpim**

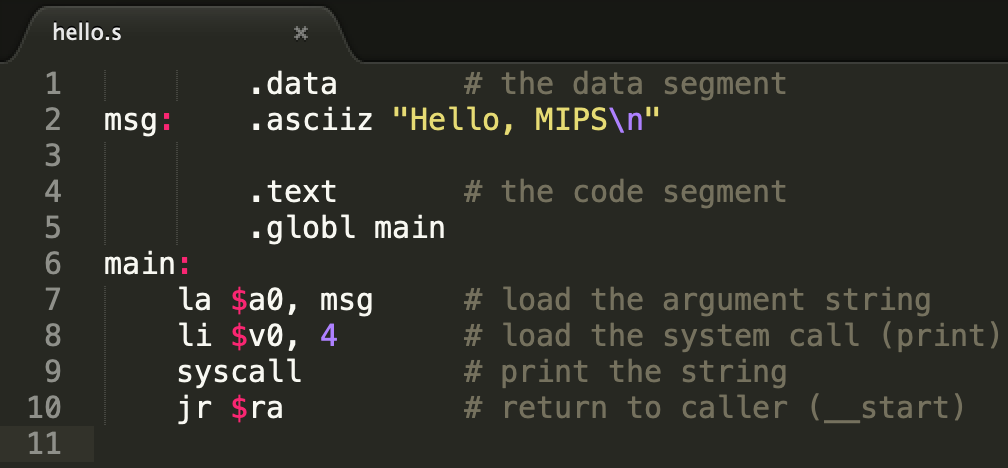
QtSpim has three windows:

* Left Window: Registers
* Text Window: Code
* Data Window: Data, Stack, and Kernel data

Memory organization:

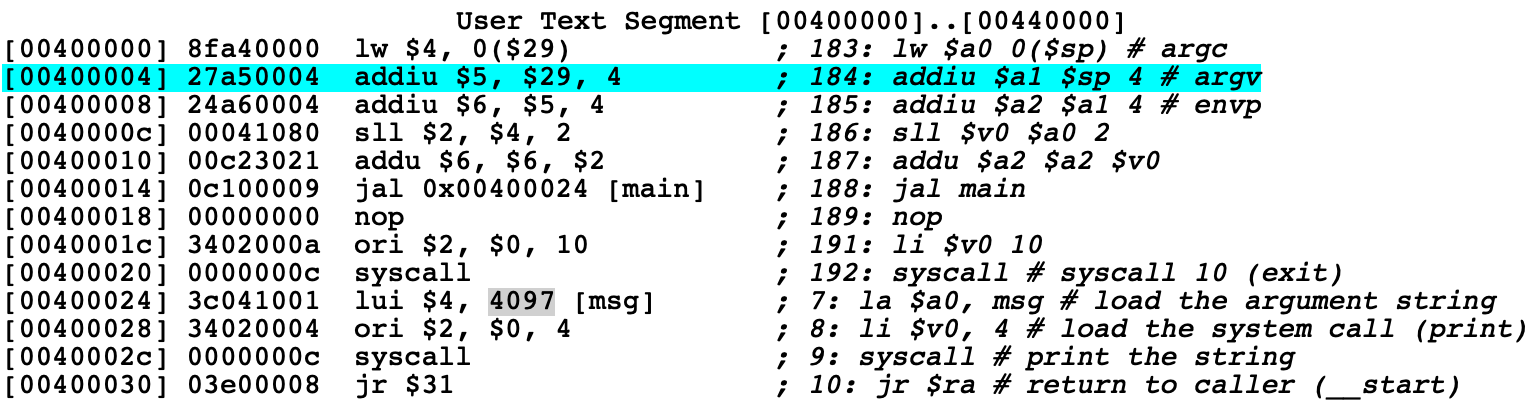
|  |  |
| --- | --- |
| 0x00400000 | Code |
| 0x10000000 – 0x10040000 | Data |
| 0x7fffeffc | Stack |
| 0x80000180 | Kernel code |
| 0x90000000 | Kernel data |

1. **First MIPS code (25pts)**

****

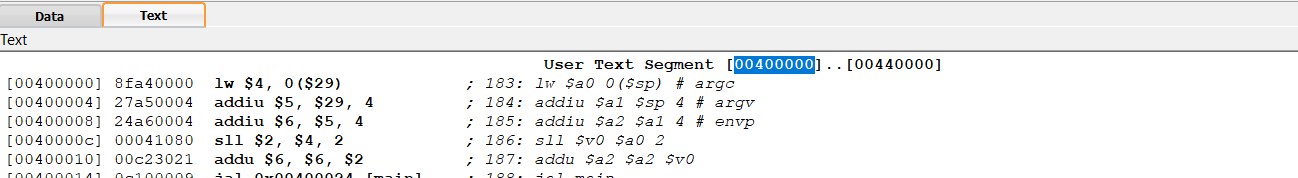
* Download ***hello.s***
* In QtSpim, **File > Reinitialize and Load File** to open hello.s
* **Simulator > Run** or press **F5** to run the program
* The code will print “Hello, MIPS” on Console Window

**Questions and Tasks:**



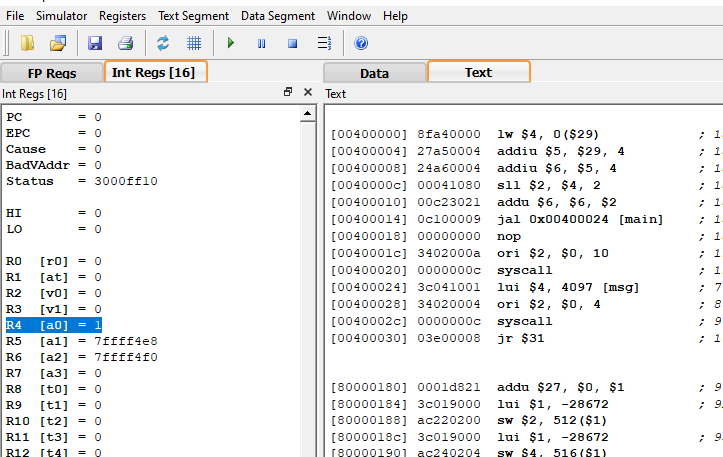
* 1. The code was loaded into memory. Determine the memory address where the code resides.

In User Text Segment

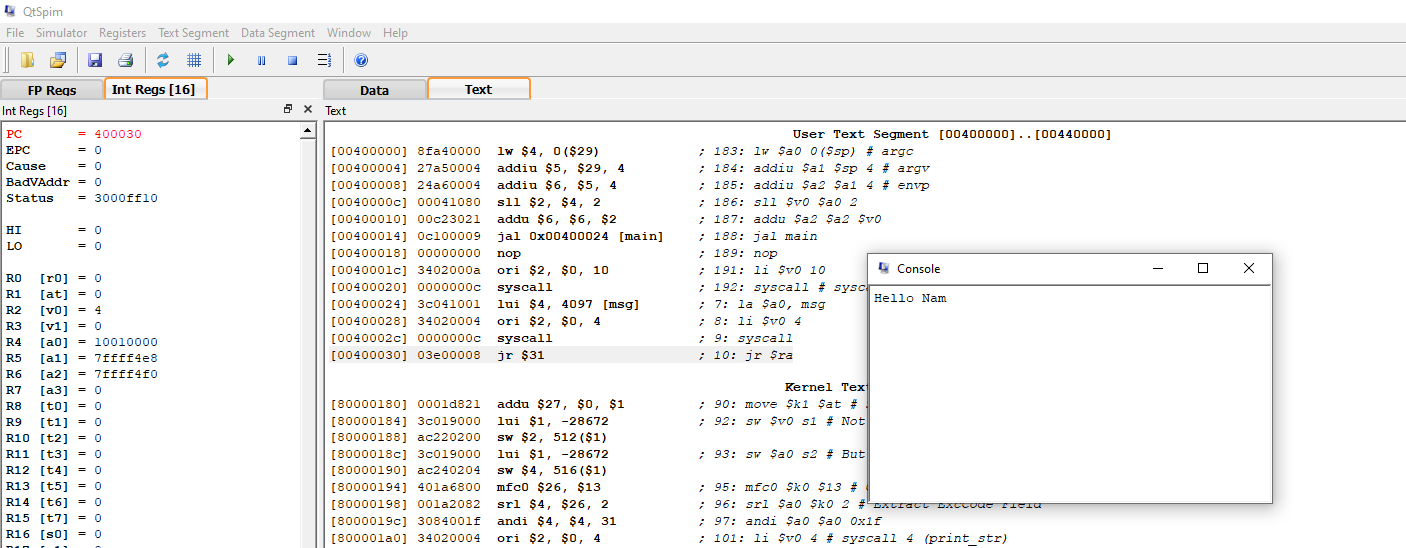


The adr is 0040000

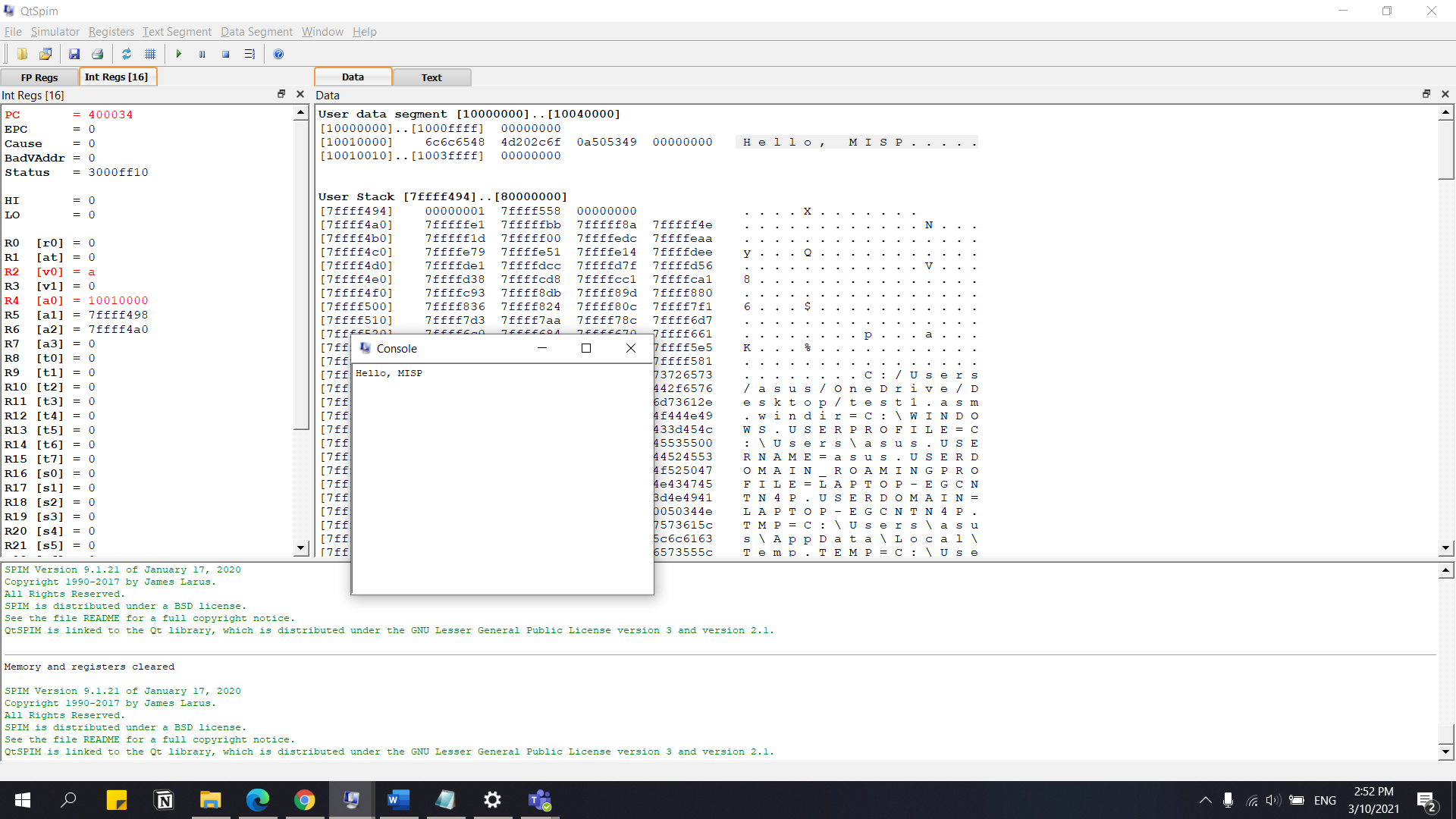
* 1. Using **Single Step** button or **F10** to step through the code (execute the code one instruction at a time). In the Text Window, one instruction is highlighted after every step. Is this instruction the current executing instruction or the next instruction? How can you know that? Hint: there is a register indicates which instruction will be executed next.
  + The answer is the next instruction
  + Before



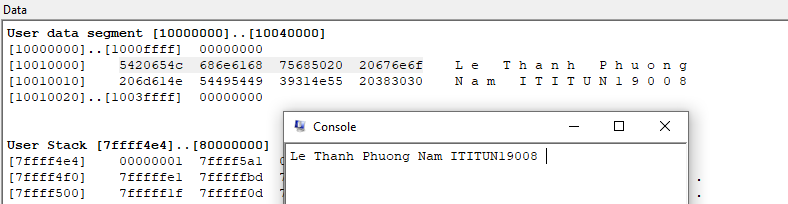
* + After



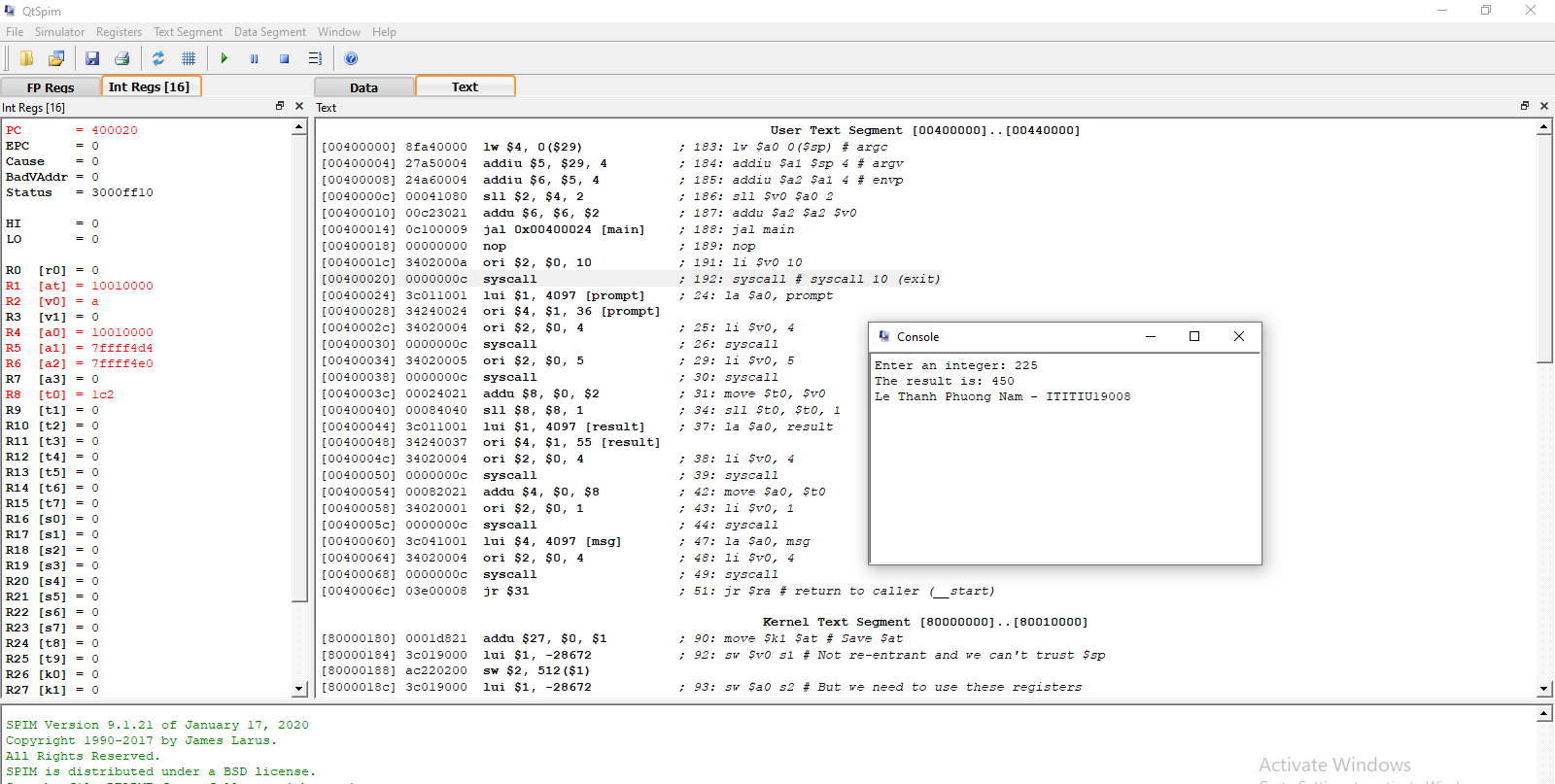
* 1. The string “Hello, MIPS\n” was loaded into memory. Determine its location in memory. Show the content of the memory segment which stores that string (in hex). Hint: look into Data Window.



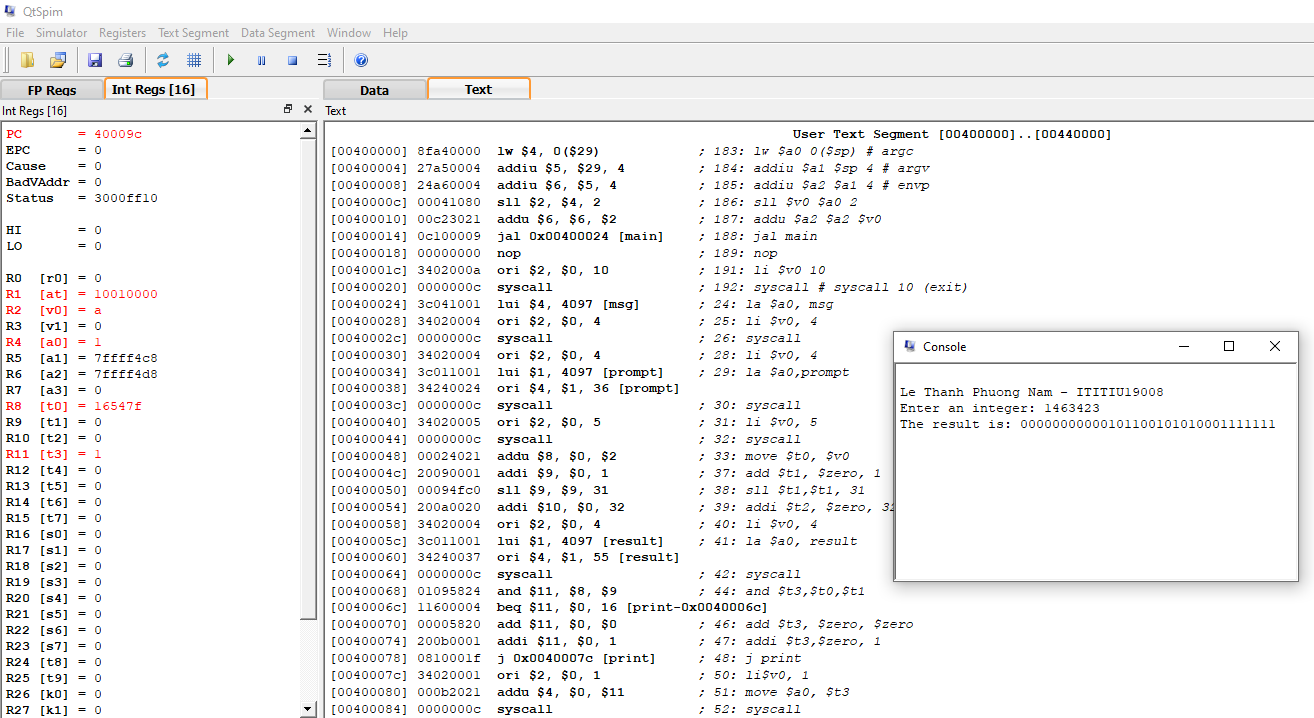
* 1. Create an assembly code to print out your full name and student ID on separate lines.



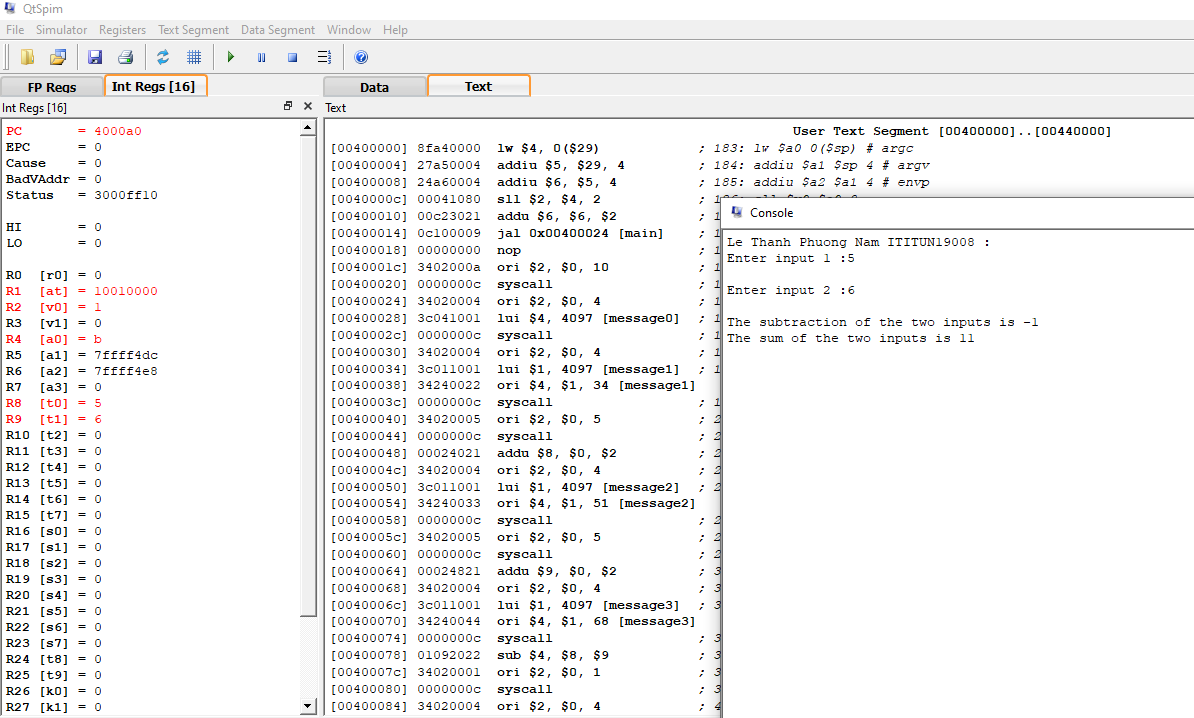
1. Load the program ***lab1\_integer\_double.s*** and run it in QtSpim. Remember to refresh register after each run. (**25pts**)
   1. What is the output when you enter the integer ***343523343532***? Is the output correct? If not, explain what happened.
   * The answer is not correct because it is out of range integer values when it is doubled from that input.
   1. What is the maximum and minimum input values that program can calculate correctly?
   * To calculate correctly, the output is integer value and the range of the input is from -1073741824 to +1073741824
   1. Modify the program so that it prints newline after outputting the result. Save your assembly as ***lab1\_integer\_double\_01.s***



* 1. Modify your program so that it print out the value of the ***first byte*** of the input. Test your program with the input ***1463423***. Save your program as ***lab1\_integer\_double\_02.s***



1. Write an assembly that reads in 2 integers and prints out the result of addition and subtraction of those 2 integers. Save the assembly as ***lab1\_twointegers.s*** (**20pts**)



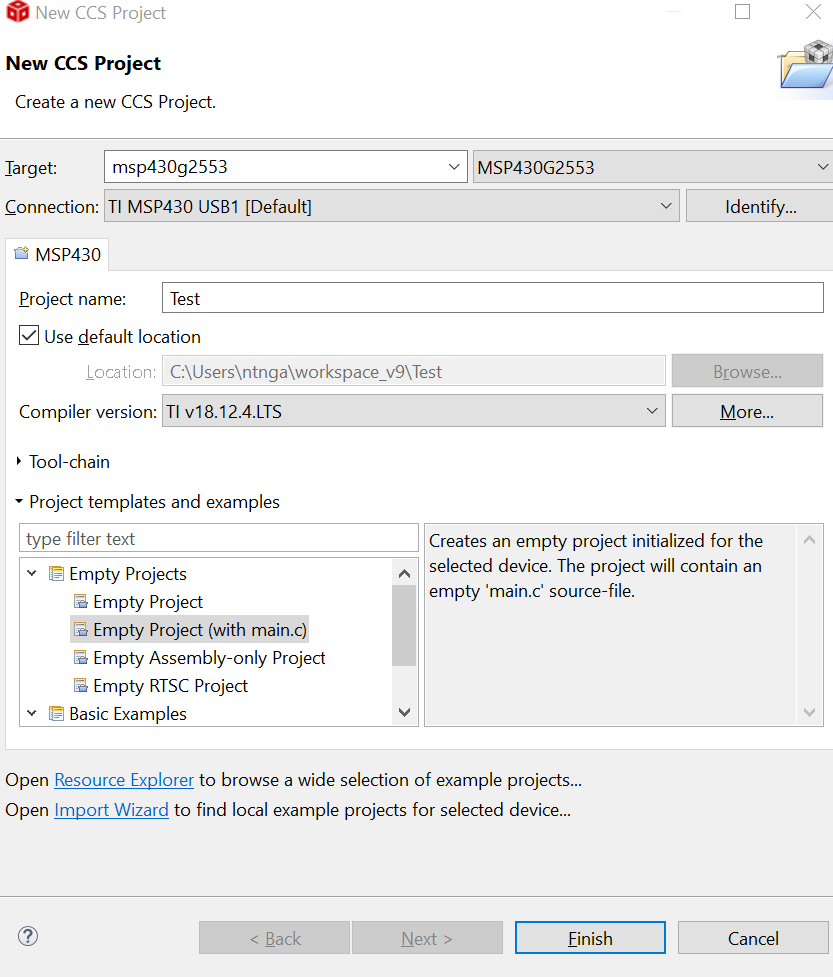
# **II. Introduction to CCS (30pts)**

+Download and install the newst version of Code Composer Studio from:

https://software-dl.ti.com/ccs/esd/documents/ccs\_downloads.html

+ Create the new project by **Alt+Shift+N**

+ In **Target**, type msp430g2553 and in project name please type **yourfullname\_lab1**, select **Empty Project (with main.c)**

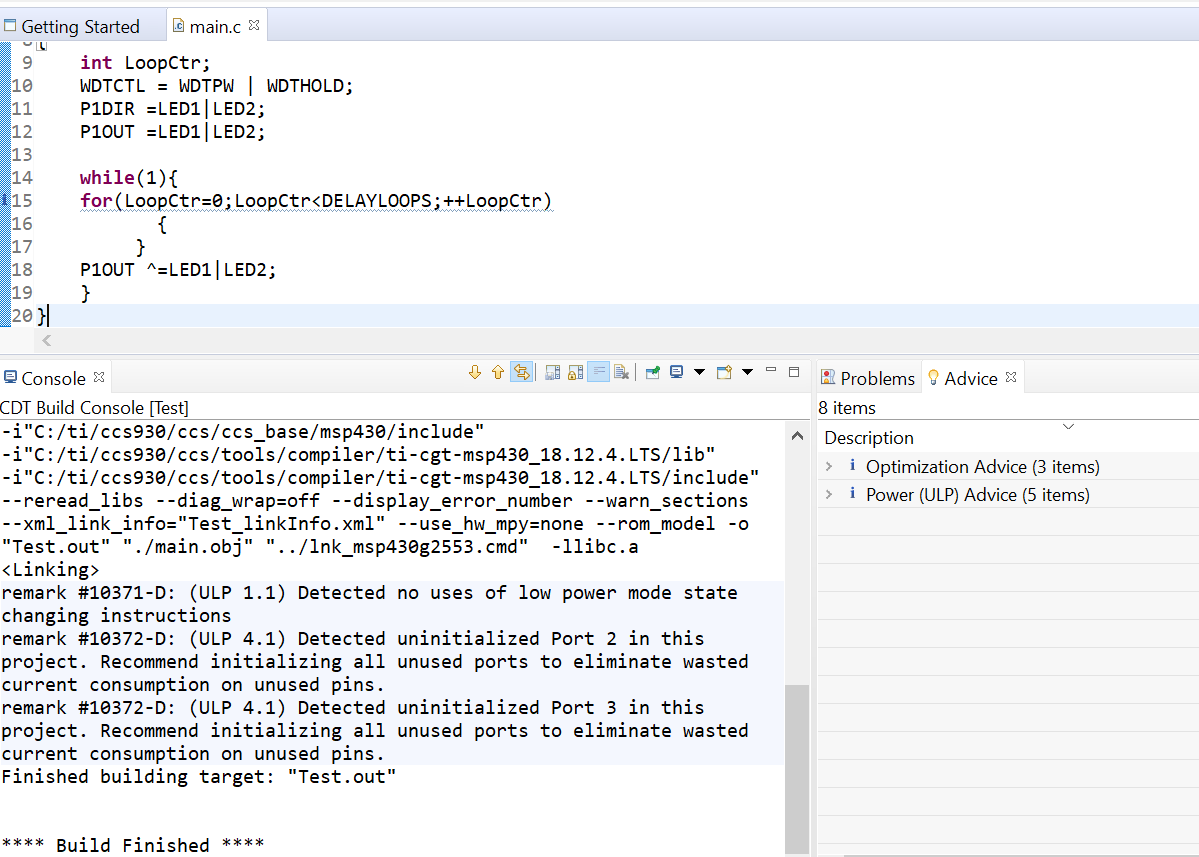


+ In *main.c*, type the sample code of Table 1

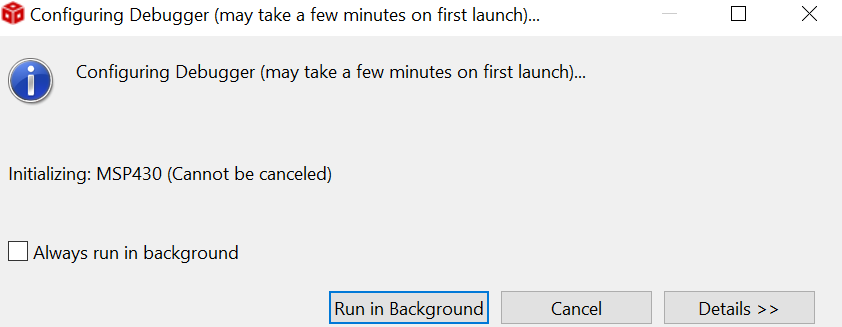
Table 1. control LED under controlled Loop in MSP430

|  |  |  |
| --- | --- | --- |
| **No.** | **Test (Sample Code)** | **Comments/Results/Functions** |
| **1.**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  20 | **#include** <msp430.h>  **#define** LED1 BIT0  **#define** LED2 BIT6  **#define** DELAYLOOPS 32000  /\* main.c \*/  **void** **main**(**void**)  {  **int** LoopCtr;  WDTCTL = WDTPW | WDTHOLD;  P1DIR =LED1|LED2;  P1OUT =LED1|LED2;  **while**(1){  **for**(LoopCtr=0;LoopCtr<DELAYLOOPS;++LoopCtr)  {  }  P1OUT ^=LED1|LED2;  }  } | Macro LED1 (assigned P1.0 or BIT0)  Macro LED2 (assigned P1.6 or BIT6)  Stop watchdog timer.  Setup LED1 and LED2 to direction output and turn on  empty delay loop  toggle LEDs |

+ Click ***build “debug”*** and observe the console window: ***Build finished***



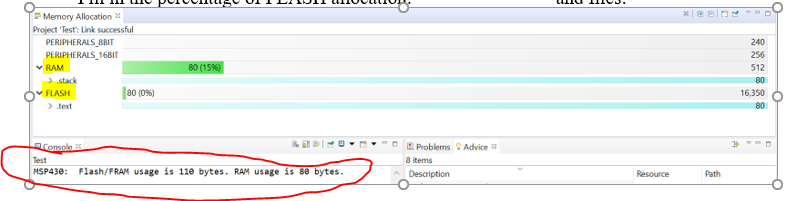
+ Click ***Run Debug*** and select ***Run in background***



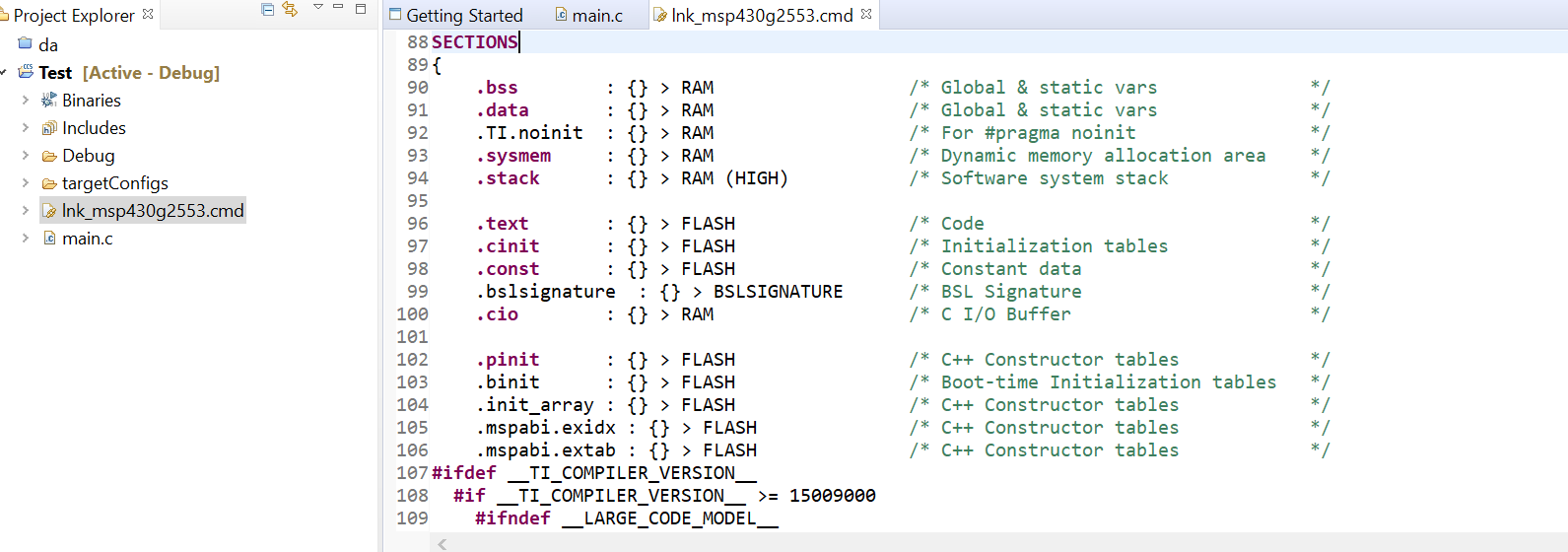
**Step 1:** Evaluation memory allocation: click **View> Memory Allocation**

Fill in the percentage of RAM allocation: \_\_\_\_15%\_\_\_\_\_\_\_\_\_\_ and files:

Fill in the percentage of FLASH allocation: \_\_\_\_\_\_0%\_\_\_\_\_\_\_\_ and files:

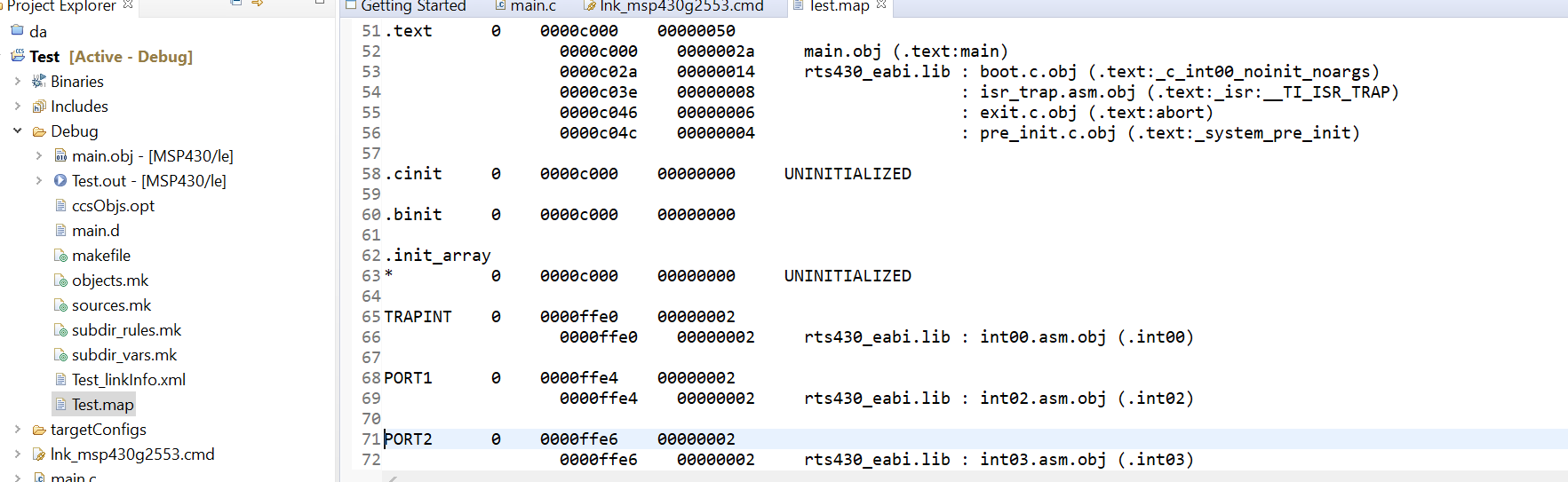


**Step 2:** **Not run**, click ***lnk\_msp430g2253.cmd*** to fill in these answers.



Summary some sentences of SECTIONS (Specify the sections allocation into memory: lines 88-109

**Step 3:** open *test.map* and fill in these addresses following:



address: 0000c02a

In Section Allocation MAP:

-text: 0 0000c000 00000050

-stack: 0 000003b0 00000050

-WDT: 0 0000fff4 00000002

-Port 1: 0 0000ffe4 00000002

-Reset: 0 0000fffe 00000002

Memory Configuration: INT02: 0000ffe4

Global symbols: sorted alphabetically by Name:

-P1IN: 00000020

-P1OUT: 00000021

- main: 0000c000

-WDTCL:\_ 00000120

-LoopCtr: \_\_\_\_\_29285\_\_\_\_\_

How many symbols? 125

**Step 4:** The Value of registers:

P1IN: before 00000020 and after 00000020

P1OUT: before 00000021 and after 00000021

LoopCtr: before \_\_\_\_\_29285\_\_\_\_\_ and after \_\_913\_\_\_\_\_\_\_\_\_

*Hint: you can use Expression or open the Core Registers tab* **(Port\_1\_2)** *to find the solution*

**Explain the values of P1IN and P1OUT registers in Port\_1\_2: \_\_\_\_\_\_\_no change\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Step 5:** When running and pausing, **observe** the Core Registers in Registers tab and **please write down** comments about the transition of these registers in there.

Core Registers PC value change from 0xC000 to 0xC01A

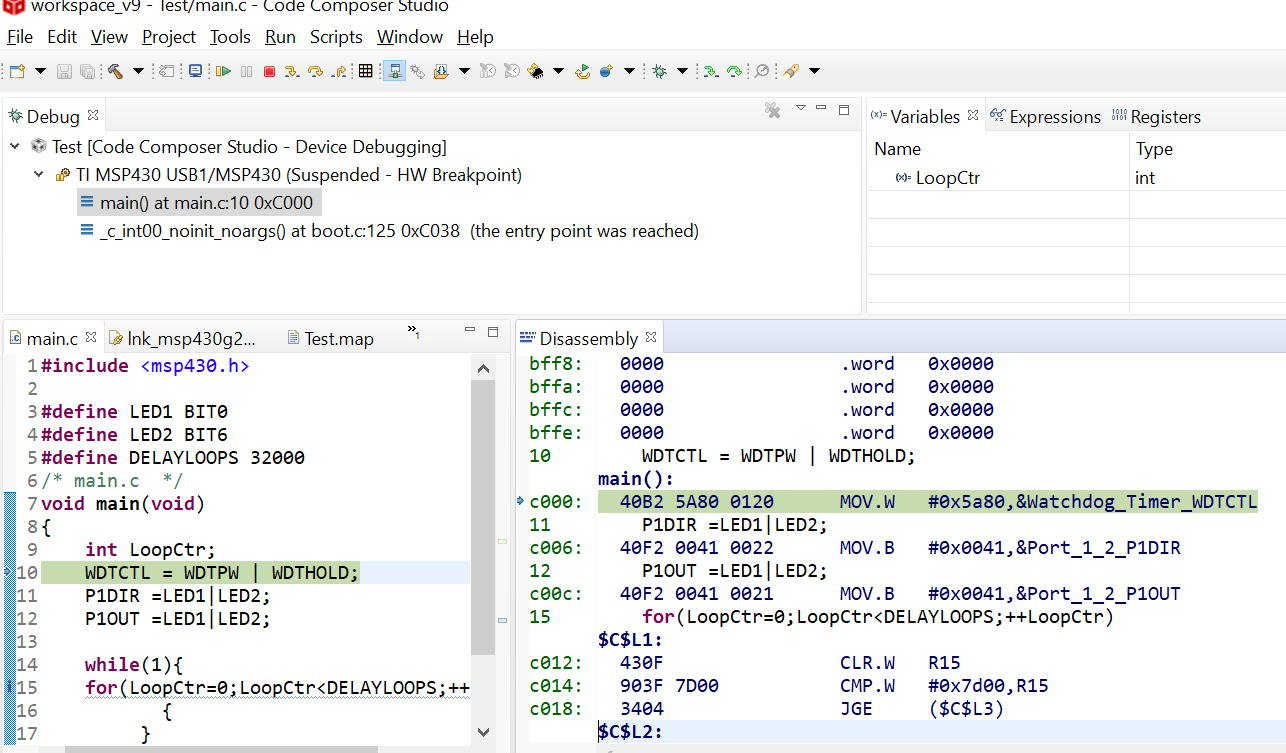
Core Registers SR value change from 0x0000 to 0x0004

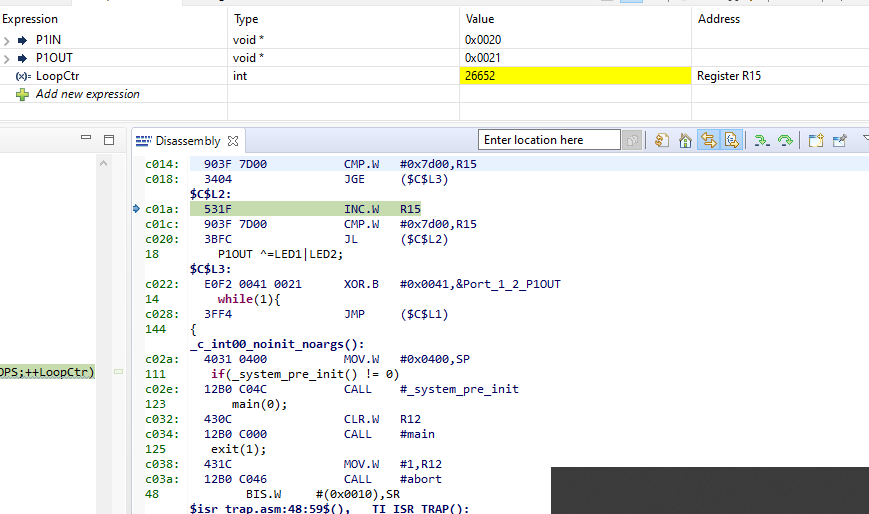
Core Registers R15 value change from 0x7265 to 0x0391

**Step 6:** how to use breakpoint in CCS : **Alt+Shift+Q,B**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Step 7:** When running and pausing, click View and open ***Disassenbly*** window, write down **these instructions of sample code:**





**Step 8:** Based on the step 7, please explain the process of sample code under Computer Architecture structure

**Step 9: C**hange the time **DELAYLOOPS** to the period of LED1 is twice that of LED2.

**#include** <msp430.h>

**#define** LED1 BIT0

**#define** LED2 BIT6

**#define** DELAYLOOPS 32000

/\* main.c \*/

**void** **main**(**void**)

{

**int** LoopCtr;

WDTCTL = WDTPW | WDTHOLD;

P1DIR =LED1|LED2;

P1OUT =LED1|LED2;

**while**(1){

**for**(LoopCtr=0;LoopCtr\*2<DELAYLOOPS;++LoopCtr)

{

}

P1OUT ^=LED1|LED2;

}}

Your code is changed (after successfully checked)

Explain the registers, address, memory, etc which will be changed to fulfill the new condition. **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# **Reference:**

1. <https://en.wikibooks.org/wiki/MIPS_Assembly/Pseudoinstructions>
2. <https://courses.missouristate.edu/KenVollmar/MARS/Help/SyscallHelp.html>
3. <https://www.assemblylanguagetuts.com/mips-assembly-programming-tutorials/#MIPS_Data_Types>
4. <https://en.wikibooks.org/wiki/MIPS_Assembly/Arithmetic_Instructions>
5. <https://gab.wallawalla.edu/~curt.nelson/cptr280/lecture/mips%20arithmetic%20instructions.pdf>