Microcomputers 1
Lab 7

Display the BCD output on the 7-segment LEDs

Total Points: 150 pts

Concepts:

- Using the HCS12 general-purpose I/O ports
- Get hands-on experience with output Ports: configuring ports as output and writing into them.
- Get a better understanding of how to create specific delays

Objectives:

- Use the HCS12 I/O ports connected to the Dragon12+ 7-segment LED displays.
- Download and execute programs on the Dragon12+ board using CodeWarrior.

Assignment:

In Lab7, you will develop a complete project that **finds *the max element in the given array and displays the result (the max element in the given array) using the four* seven-segment display LEDs**. Each element in the array is a one-byte **unsigned** number.

Lab7 consists of four major procedures as below:

Step0: Data declaration and initialization.
Step1: Find the max value in the given array.
Step2: Convert the max value found in Step1 into the Binary Corded Decimal (BCD) format.
Step3: Display the BCD number (the max value in the array) on the four 7-segment LEDs.

Create a new project with the name '**Lab7_Part1.mcp**' as you did in the previous labs. Open the main.asm file in the project to write the code. From the dropdown list on the left, select "**Full Chip**

Simulation” if it is not so yet. Delete the prewritten sample code marked with black background in main.asm provided by the CodeWarrior IDE.

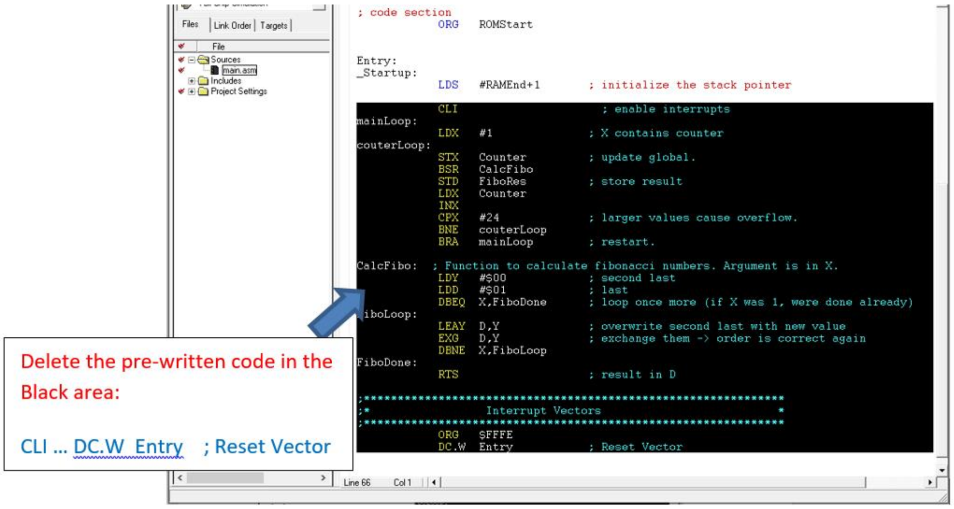


Fig. 1: Main program part.

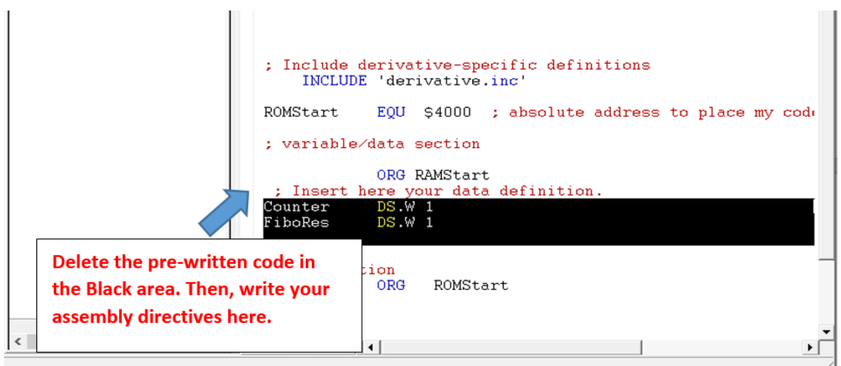


Fig. 2: Data declaration section.

Step0: Data declaration and Initialization

1) Array Data: In the data section, the input array and the size of the given array can be defined as below:

```
Array1:  DC.B   $64, $45, $22, $25, $52, $66, $48, $53, $50,$AF
N        EQU    10
```

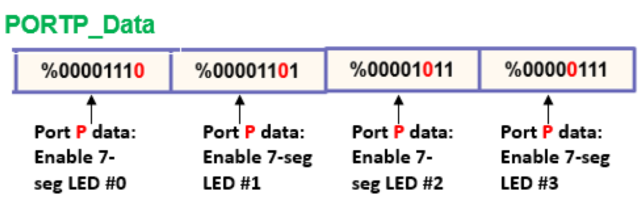
There are 10 elements in the given array, and the element of the array is saved in the memory locations starting from **RAMStart** (=$1000). The memory location $1000 has two label names, RAMStart and **Array1**. The size of the array is saved in the label name **N**.

2) BCD Data: After the max value in the given array is found in Step1, the max value is converted into Binary Coded Decimal (BCD) format to display onto the four 7-segment LEDs. The converted BCD digits are saved in the memory locations from $1010 (= BCDNum) to $1013.

```
; The memory allocation for the BCD number for the array maximum
            ORG $1010
BCDNum:   DS.B   4
```

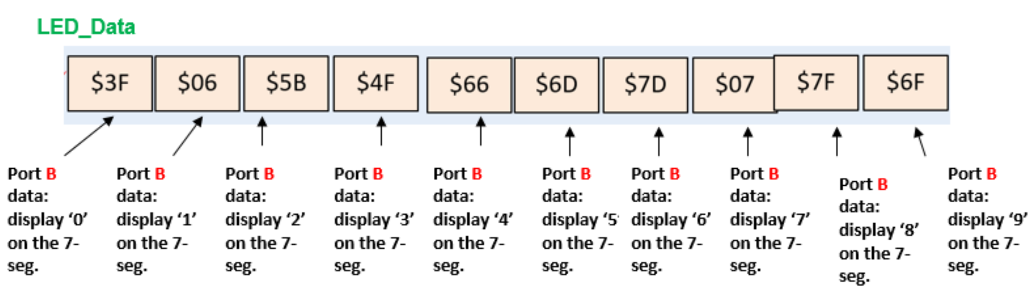
3) Port P data: When displaying the BCD digits onto the four 7-segment LEDs, only one 7-segment LED is activated at a time using the time-multiplexing technique. To enable only one 7-segment LED display, only one bit in Bit 0 to Bit3 in PORT P **should be set to zero**, and the other bits are set to ones. The PORT P data to enable/disable the four 7-segment LEDs are defined below.

```
;PORT P data to enable the 7-segment LED#0 - LED #3
PORTP_Data:  DC.B %00001110, %00001101, %00001011, %00000111
```



4) Port B Data: The LED_Data contains the hex-values for PORT B to represent the digit 0-9.

```
LED_Data:   DC.B $3F, $06, $5B, $4F, $66, $6D, $7D, $07, $7F, $6F
```



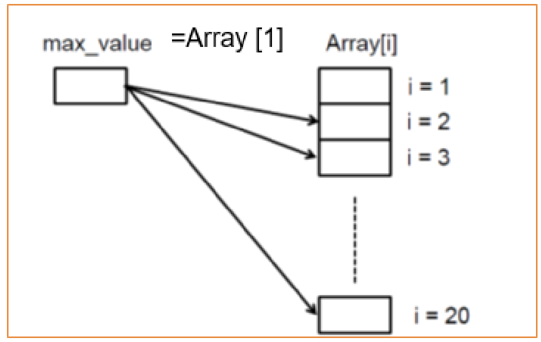
You need to place the data definition in the data section shown in Fig 2. Using the assembly directives, define the data. Now, you have completed the data declaration part. To display the *max value in the given array using the four 7*-segment LEDs, three major programming steps are explained next.

Step1: Find the maximum value in a given array.

To find the maximum value, you need to search through the given array elements using a loop structure. The data type of the given array is one unsigned byte number. In assembly, programmers have the responsibilities to choose the right version of instruction when using comparison instructions. Therefore, you have to use **unsigned Conditional Branch instructions.** The pseudo-code for this task is described in the box below:

Find the max value in the given array

1) Max_value = Array [1]
2) Scan the array from Array[2] to Array [N]
   In each iteration:
      if Max_value < Array[i] then
         Max_value = Array[i]
3) After scanning all the array elements,
   The variable, Max_value, holds the max value in the given array.



1. Register A is used to hold the max value in the given array. First, assign the first element into register A by loading the first element in the given array to register A.
2. Register X is used to hold the address of the element in the array. Load the address of the second element into register X.
   LDX #Array1+1 ; X is holding the address of the array second element
3. Register B is used as a loop counter. Load the array length N-1 into register B because you are going to scan N-1 elements.
   LDAB #N -1

Loop: Compare register A with the content at the memory location given by register X

4. If register A >= Mem[X], branch to **SKIP** ; use unsigned branch instruction **BHS**

   If not, the value in register A is less than the value in the memory location indexed by register X. Replace the value in register A with Mem[x] using the loading instruction. ; A=Mem[X]

SKIP: 5. Increment register X ; X will hold the address of the next element in the array

6. Decrement the counter register B
7. If register B is not equal to 0, branch to **Loop**

Step2: Implementing the BCD conversion using the loop structure

The max value in the given array is represented in a binary format. This is not a convenient format for reporting the number to a human user. A binary number is converted into the BCD format using repeated division by 10 to display the result on the 7-segment LEDs. In Lab4, you implemented the BCD code conversion code. Use the code with minor modifications. The pseudo-code for the BCD conversion is described below:

1. The maximum value in the given array found in Step1 is saved in register A. Transfer register A into register B.

2. Clear register A. So register D= A and B registers

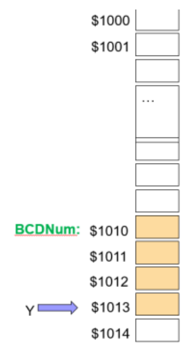
3. Index register Y id holding the address of BCD digits, starting from the least significant byte first.
   LDY #BCDNum+3

4. **BCDLoop:**
   4.1 Assign register X with the value of the Divider
       LDX #10
   4.2 Do the integer division:

       IDIV ; D/X X=Quotient, D=Remainder

   4.2 Store the remainder in register B at the location given by register Y
   4.3 Decrement register Y
   4.4 Exchange register X with register D ; XGDX
   4.5 Compare D with $00
   4.6 If D register is NOT equal to 0, go to **BCDLoop**

5. Move $00 to **BCDNum.**



Step3: Display the BCD number (the max value in the given array) on the four 7-segment LEDs

A. Configuration of output ports:

First, you need to configure the data direction registers for PORT B, PORT P, and PORT J. Then disable the flashing LEDs by setting a value in Bit1 in PORT J.

```
BSET   DDRB,  %11111111       ;configure 8bits in  Port B as  output pins
BSET   DDRJ,  %00000010       ;configure Bit 1 in PORT J  as an output pin
BSET   DDRP,  %00001111       ; configure the lower bits in PORT P as output pins
```

```
;Disabling eight flashing LEDs by setting Bit1 in POTR J  to 1
 BSET   PTJ,  %00000010                 ; Disable eight flashing LEDs
```

B. BCD digit display on the 7-segment LED.

The BCD digits for the given array's max value are saved at the memory locations starting from the memory location $1010 to the location $1013. The starting address $1010 is labeled as **BCDNum**, as shown in Fig. 3. The PORT P data to enable /disable the 7-segment LEDs is saved at the memory locations starting from $1014 (= **PORTP_Data**) to $1017. The LED data table is saved after that. The data needed for Step 3 are organized as below:

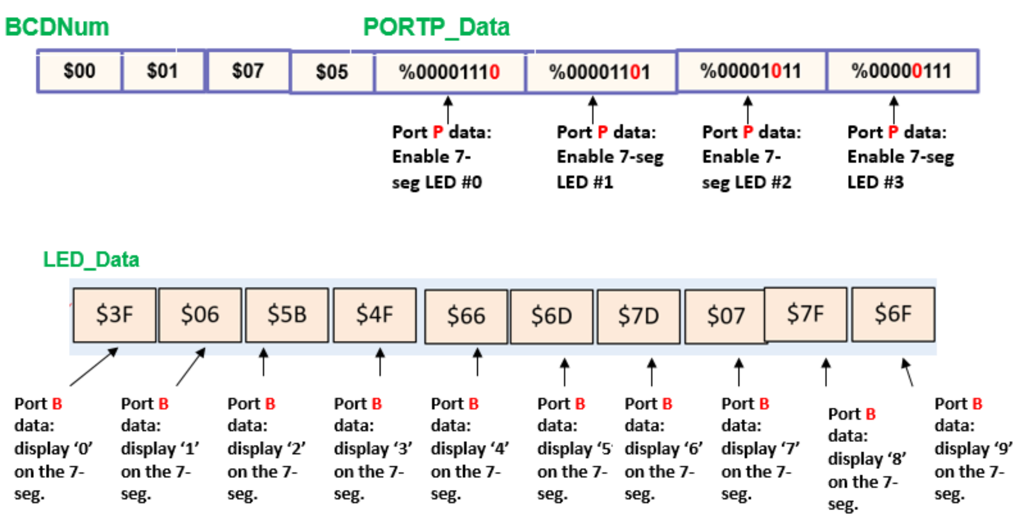
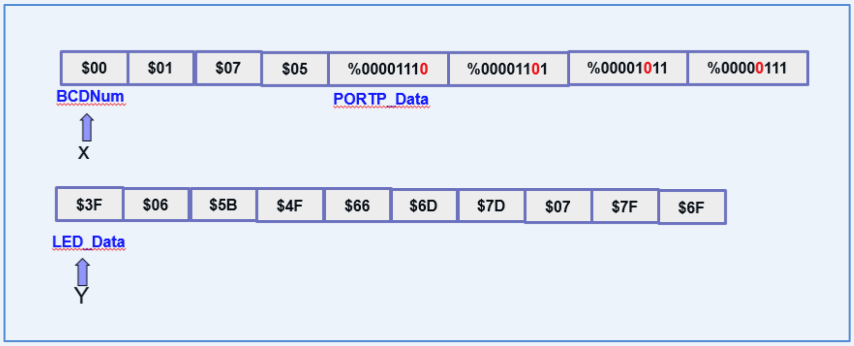


Fig.3. Data definition in the data section

For each BCD digit, the following steps are required. The pseudo-code is below to display the BCD on the enabled 7-segment LED.

Forever:

1. Register X has the address of the BCD digit. Initially, it has the address of the first BCD digit.
   LDX #BCDNum
2. Register Y holds the beginning address of **LED_Data.**
   LDY #LED_Data



LEDLoop:

3. Load the BCD digit into register A from the memory location in register X.
   A= $05
4. Register B has the data for PORT B. Load register B from **the address Y + A.**
   Ex) If register A has the value $05, the corresponding 7-segment data to display the digit '5' is located in the memory address, Y + A. In this case, the address is Y + 5.
   B= 6D
5. Then, Store register B into PORTB
6. For each BCD digit, the Port P data is located in the memory address at X+4. Move the data at the memory address X+4 into PORT P to enable the corresponding 7-segment LED.

7. 1 ms delay :
   - Before assigning 1ms to register Y, save register Y into the Stack using PSHY **because register Y is used as an index register** to hold the beginning address of the LED_data.
   - Assign 1ms to Y register
   - Call the subroutine Delay_yms
   - After the return from the subroutine, Pull Y (restore Y register) : **PULY**
8. Increment X register
9. Compare register X with #BCDNum+4 ; use the instruction CPX
10. If register X is Less than #BCDNum+4, branch **LEDLoop**
    else Branch to the beginning at step 1 (BRA **Forever**)

After completing your code, click on the green **debug arrow** on the menu bar to start the CODE Warrior Simulator/Debugger.

a) Add two breakpoints at the beginning of the inner loop, '**LEDLoop**' and the outer loop, **'FOREVER'**. Breakpoints are set by right-clicking on the instruction and selecting "Set Breakpoint".

b) In Lab7, **the program counter (PC) in the register window starts at $4000.** The instruction starting at the memory location $4000 is to initialize the stack pointer as below:

    LDS #RAMEnd+1

c) Run the program from the beginning by clicking on the "Start/Continue" button. Note that when the processor stops at a breakpoint, the instruction that the PC points to has not been executed.

Task1: Check the memory locations $0001 (PORT B) and $0258(PORT P). For each inner loop iteration in Step3, write down the values in the memory locations (at $0001 and $0258) and generate the memory screenshots. The sample screenshots of the memory map are provided below.

Table 1. Memory map

Iteration	Memory $0001 (PORT B)	Memory $0258 (PORT P)
1		
2		
3		
4		

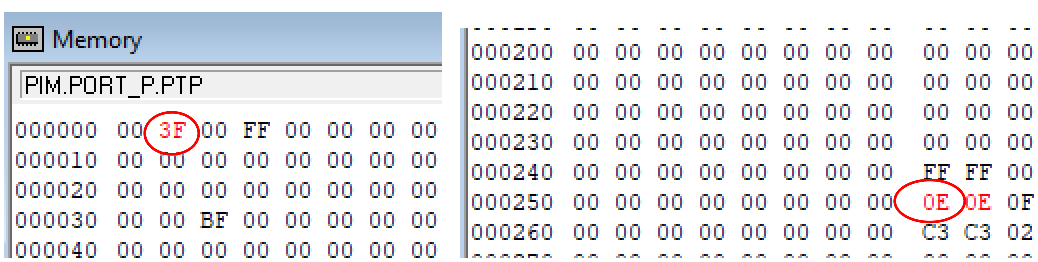


Fig. 4. Screenshots of the Memory map: display '0' on LED #0.

Task2: Test the program with the new data set in Array1 and N as below.

```
Array1: DC.B  $FA, $02, $34, $FD, $52, $11
N       EQU   6
```

Add two breakpoints at the beginning of the inner loop, '**LEDLoop**', and the outer loop, **'FOREVER'**. Then, rerun the program and check the memory locations $0001 (PORT B) and $0258(PORT P). For each inner loop iteration, write down the values in the memory locations (at $0001 and $0258) and generate the screenshots of the memory.

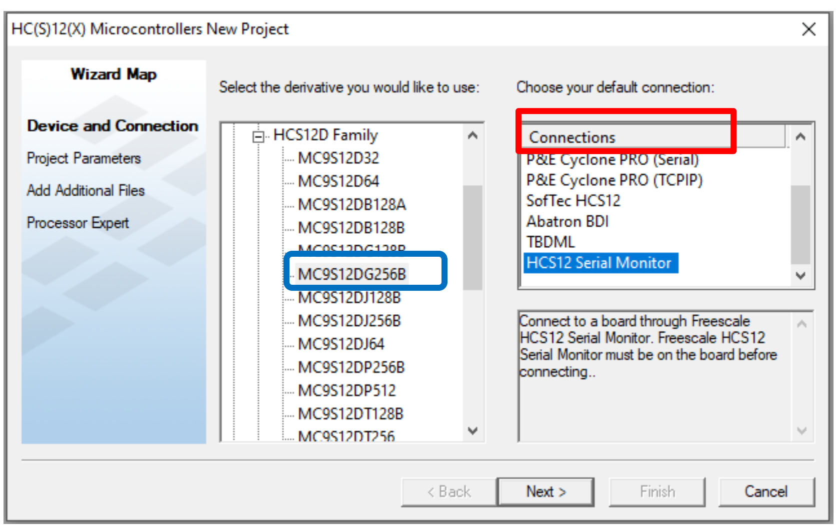
Table 2. Memory map

Iteration	Memory $0001 (PORT B)	Memory $0258 (PORT P)
1		
2		
3		
4		

Part II. Download and execute programs on the Dragon12+ board

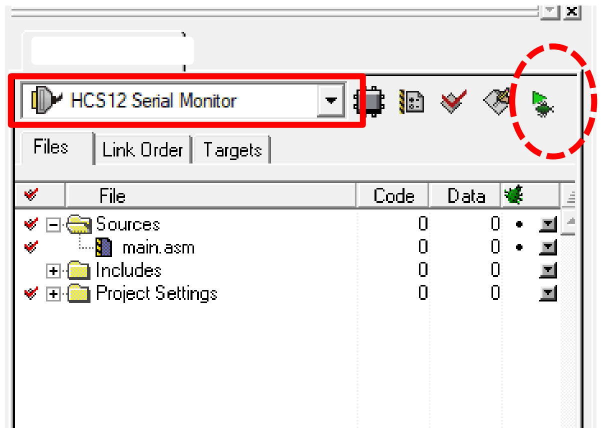
This Part II is intended to familiarize you with Wytec's **Dragon12+ evaluation board**. **The Dragon12+ board runs the program that CodeWarrior communicates with over the RS-232 monitor.** This allows the PC to control the Freescale HCS12 in a debugging environment.

1. Create a new project by clicking the 'Create New Project' button. In the HC(S)12(X) Microcontrollers New Project wizard window, select the same processor ("**HCS12**" →"**HCS12D Family**" → "**MC9S12DG256B**"). In the Connections pane, select "**HCS12 Serial Monitor**", as shown in the below figure. Click Next.

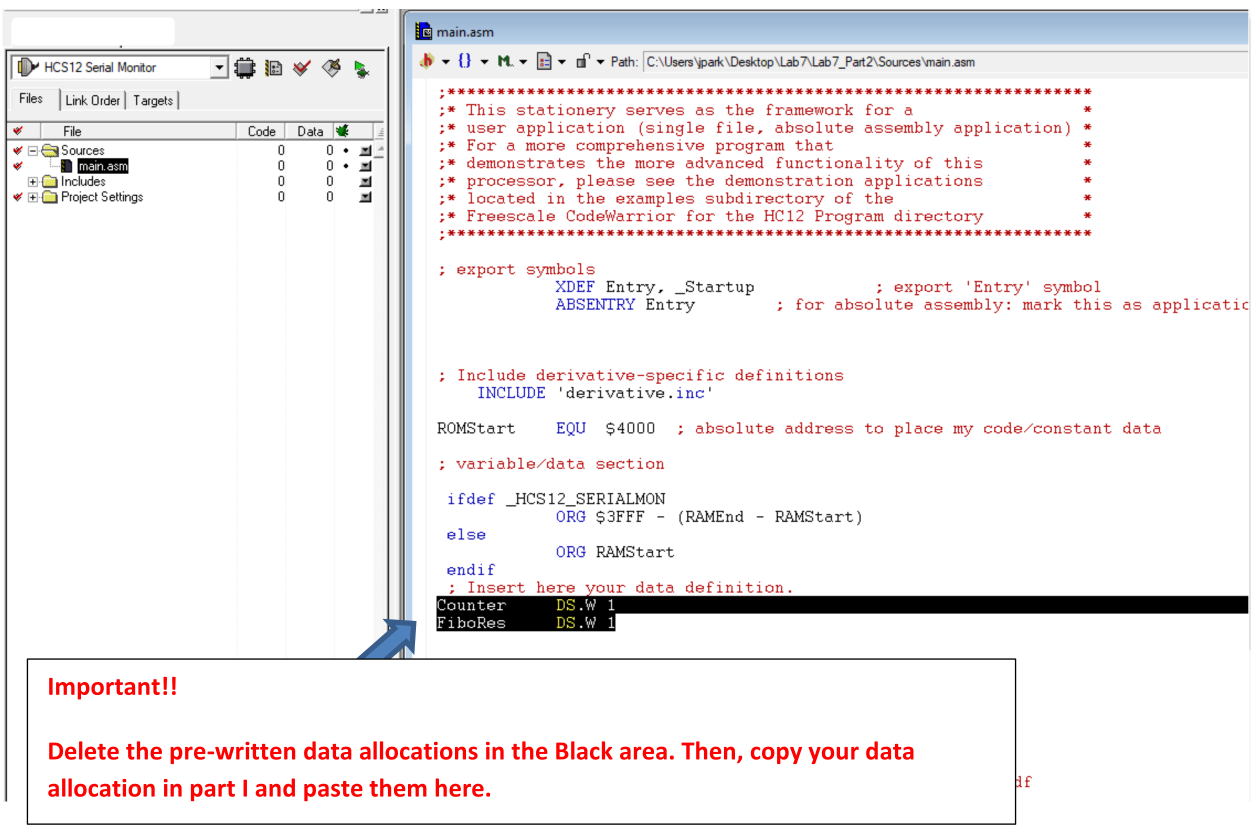


2. In the **Project name** field, type **Lab7_Part2.mcp**. Uncheck all the checkboxes and then check the **Absolute assembly** checkbox. Click on **Finish** (NOT Next!).

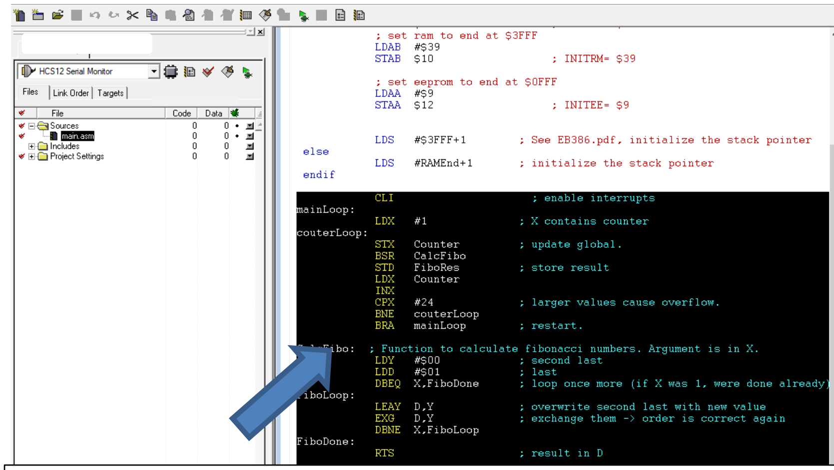
3. CodeWarrior main window opens up, as shown below. From the dropdown list on the left, select "**HCS12 Serial Monitor**" if it is not so yet.



4. Delete the prewritten data allocation marked with a black background. Then, copy your data allocations in Part I and paste them here.



5. Delete the prewritten sample code marked with the black background in main.asm provided by the CodeWarrior IDE.

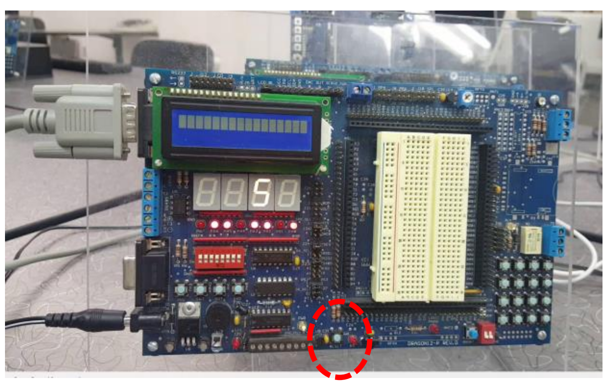


Delete the pre-written code in the Black area: CLI … …….. DC.W Entry ; Reset Vector

Copy your lines of code in Lab7_partI starting Step 1 to the end of your code.

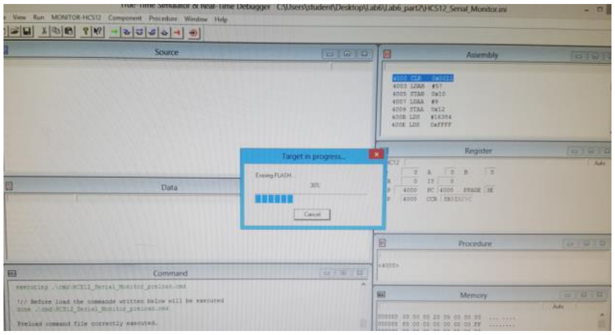
Copy your lines of code in Lab7_partI starting Step1 to the end of your code and paste them here.

6. **Press the light blue RESET button on the Dragon12+ board** located along the bottom edge in the middle of the board. The 8 red LEDs should light up from right to left, indicating that the CodeWarrior firmware is running on the evaluation board.

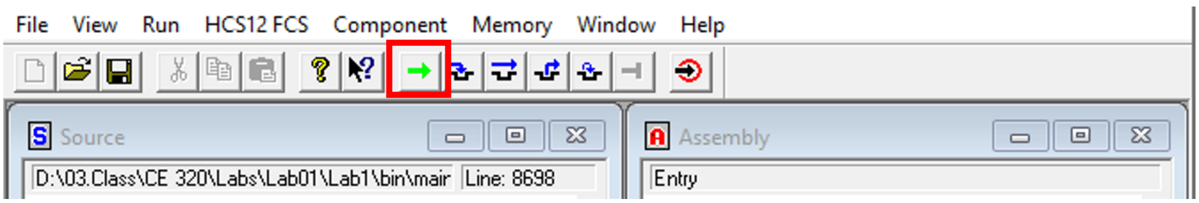


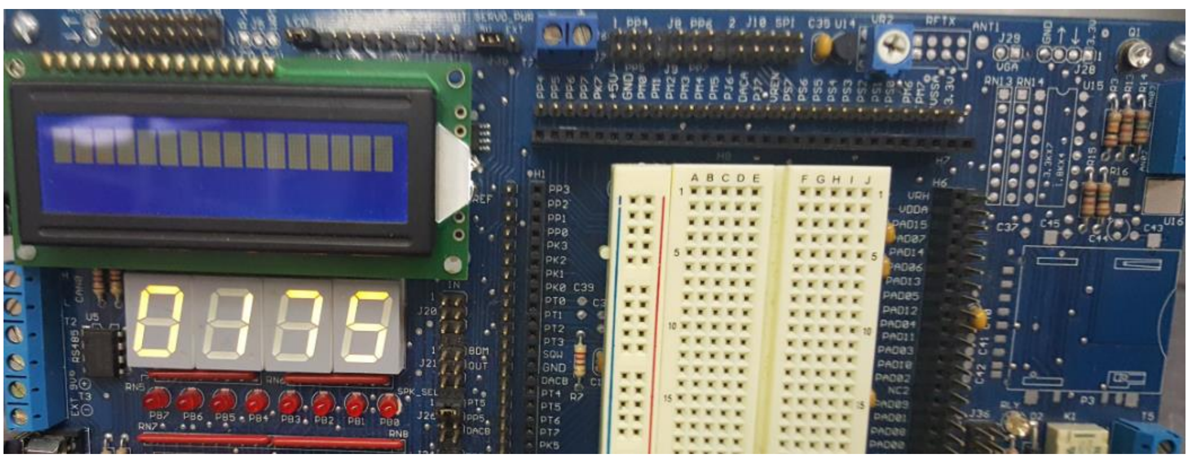
7. Then, click on the "debug" button, shown with a green arrow. This should assemble the program and open a second window. As this second window opens, **CodeWarrior will automatically download the machine code into the Dragon12+ board**.

8. Then, click on the "debug" button, shown with a green arrow. This should assemble the program and open a second window. As this second window opens, **CodeWarrior will automatically download the machine code into the Dragon12+ board**.



9. Run the program by clicking on the “Start/Continue” button.





What to Submit in Blackboard:

1) Per each member: 50pts

Complete Lab7_WorkBook.doc :

- **(25 pts) Task1: Memory-mapped I/O**
- **(25 pts) Task2 :Memory mapped I/O**

2) One copy per group : 100 pts

- 60 pts: The assembly source code for Part II (main.asm files only)

- 40 pts: a short video clip that shows the four 7-segment LED displays

Pay attention to the file name convention:

- **Individual file: Lab7_Student1_Firstnname_Lastname.pdf**
  Ex) Lab7_Smith_Green.pdf

- **Group file:**
  Source file: main.asm
  Video files: Lab7 _Student1 Lastname_Student2 Lastname.mp4