Microcomputers 1

Lab 4

Number Format Conversion

Concepts:

- Different forms of number representations: hexadecimal (binary), binary coded decimal, and ASCII.

Objectives:

- Write an assembly program that converts the number into binary coded decimal and ASCII formats.

Introduction:

We have mostly been using hexadecimal (actually binary number system) to store values in memory. This method requires the fewest number of bits to store a given value. For example, **65535_{10}** is "$**FFFF**" and requires two bytes. However, this is not a convenient format for reporting the number to a human user.

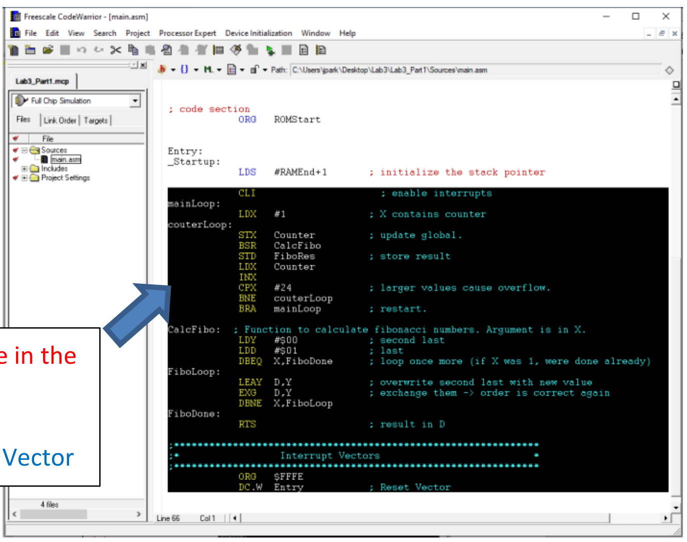
One common user-friendly format is **Binary Coded Decimal (BCD)**. In BCD code, **each decimal digit is represented by its binary equivalent**. For example, **5678_{10}** is represented by 0101 0110 0111 1000. This method makes the value being stored appear to a human as the decimal representation but at the cost of more space. For example, storing the value 65535_{10} in memory appears as "$06 $05 $05 $03 $05". This format is often used in conjunction with 7-segment LED displays.

Another common format is **ASCII** (American Standard Code for Information Exchange). This method is used to communicate with devices like text-based terminals and printers. In ASCII, each item is given a 7-bit code that is often stored as a two-digit (1-byte) hexadecimal number. ASCII has codes assigned to letters, numbers, and non-printable items like backspaces and carriage return. In the hexadecimal representation for the ten numbers in ASCII, the first digit is a "3" and the second digit is the number. For example, storing 65535_{10} in memory appears as "$3**6** $3**5** $3**5** $3**3** $3**5**".

Lab4 will work with three different number representations: **Binary (hexadecimal), Binary Coded Decimal (BCD), and ASCII Code.** We have mostly been using binary (hexadecimal) to store values in memory. The input and output equipment generally uses decimal numbers. For example, if we were going to display a number on 7-segment LED displays in decimal, we would prefer to have the number in the BCD format. That means decimal numbers must be coded in terms of binary signals.

Part I: Binary to Binary Coded Decimal (BCD) conversion

Create a new project with the name '**Lab4_part1.mcp**' as you did in the previous labs. CodeWarrior's main window opens up. From the dropdown list on the left, select "**Full Chip Simulation**" if it is not so yet. Next, delete the prewritten sample code marked with black background in the main.asm provided by the CodeWarrior IDE.



```
; code section
            ORG    ROMStart


Entry:
_Startup:
            LDS    #RAMEnd+1       ; initialize the stack pointer
            CLI                     ; enable interrupts
mainLoop:
            LDX    #1              ; X contains counter
couterLoop:
            STX    Counter         ; update global.
            BSR    CalcFibo
            STD    FiboRes         ; store result
            LDX    Counter
            INX
            CPX    #24             ; larger values cause overflow.
            BNE    couterLoop
            BRA    mainLoop        ; restart.

CalcFibo:  ; Function to calculate fibonacci numbers. Argument is in X.
            LDY    #$00            ; second last
            LDD    #$01            ; last
            DBEQ   X,FiboDone      ; loop once more (if X was 1, were done already)
FiboLoop:
            LEAY   D,Y             ; overwrite second last with new value
            EXG    D,Y             ; exchange them -> order is correct again
            DBNE   X,FiboLoop
FiboDone:
            RTS                    ; result in D

;**************************************************************
;*                 Interrupt Vectors                          *
;**************************************************************
            ORG    $FFFE
            DC.W   Entry           ; Reset Vector
```

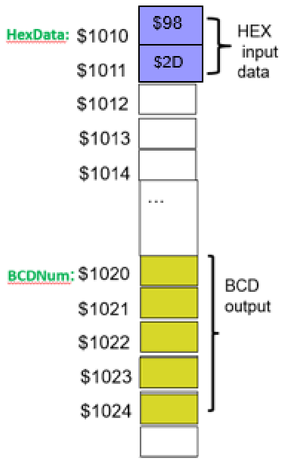
Delete the pre-written code in the Black area:

CLI … DC.W Entry ; Reset Vector

- Write an assembly program that meets the following requirements. First, the program converts the 2-byte hexadecimal number to a 5-byte BCD number using the **loop structure with the comparison branch instruction**. The hexadecimal number is supplied in addresses \$1010 and \$1011, and the BCD number (result) will be saved in addresses \$1020-\$1024.

- **Data allocation**:

  First, assign the 2-byte hexadecimal input number at the memory location \$1010 and \$1011. Then reserve the memory spaces for the 5-byte BCD result starting from \$1020 using assembly directives.



The data definitions are placed after ORG **RAMStart**. First, delete the prewritten sample code provided by CodeWarrior in main.asm. Type the below assembly directives.

```
                ORG $1010
HexData:    DC.W      $982D     ; Binary number to be converted

                ORG $1020
BCDNum:     DS.B      5         ; reserve 5-byte for the BCD output
```

- To convert the binary number into the BCD form, procedures are summarized in the pseudocode below:

  Step 1. Use register Y to hold the address of the BCD digits. Initialize register Y with the address of the least significant digit (\$1024).

Use '**#**' sign to load the address itself rather than the contents in the address:

Ex) LDY #BCDNum+4 **;** **BCDNum**=$1020, Y=$1020+4=1024

or **LDY #$1024**

or LDY #$1020+4

Step 2. Load the 16 bit binary number at the memory location $1010:$1011 into register D ; register D is the dividend

Step 3. Compare Y with #BCDNum **; BCDNum**=$1020

Ex) **CPY #BCDNum** ; compare register Y with $1020

Step 4. Exit loop if register Y is Lower than #**BCDNum**

Use the unsigned branch instruction (ex: BLO) and the branch will be taken to the end of the program ('ExitLoop').

Step 5. Load #10 into register X ; register X is a divider

Step 6. Use **IDIV instruction** **; IDIV means D / X**

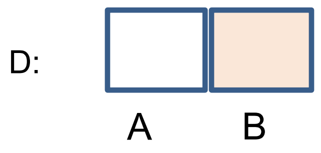
- Quotient will be saved in register X

- Remainder will be saved in register D

Step 7. Since the divider is 10, the remainder is less than 10 ([0 to 9]).

The remainder is in register B.

Save the remainder in register B into the memory pointed by register Y.



Use the index-addressing mode when you save register B into the memory pointed by register Y.

Step 8. **XGDX** ; Exchange D with X

→ D register can hold the quotient for the next division.

Note: you can also use a TFR instruction instead of exchange instruction. (TFR X,D)

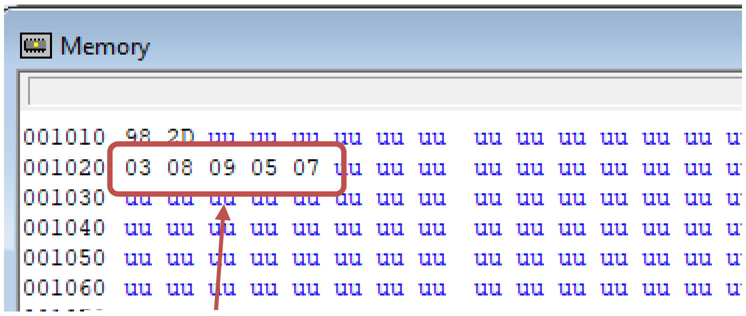
Step 9. Decrement register Y by 1

Step 10. Take unconditional branch to Step3.

(The loop will be terminated if register Y < #BCDNum at Step 4).

ExitLoop: SWI ; end of the program

1. *To start your assembly instruction at the memory location $4500,* ***you need to add the assembly directive ORG $4500 just before your code line 1 (Step1)***.
2. After you complete the code, then, click on the green **debug arrow** on the menu bar. The simulator/Debugger window opens up.
3. Before running the program, click on the textbox for **the program counter (PC) in the register window and write the value "4500" for the starting address** of your program so that the processor knows where to start executing your code.
4. F11 is the single-step button. Use it to execute one instruction at a time. For the debugging, check the values in the registers, D, Y, and X and the output digits in the memory locations $1020:1024.
5. The BCD output will be displayed at the memory location, $1020:1024. Take the screen shots of the BCD output in the memory pane, Assembly pane and Register pane.

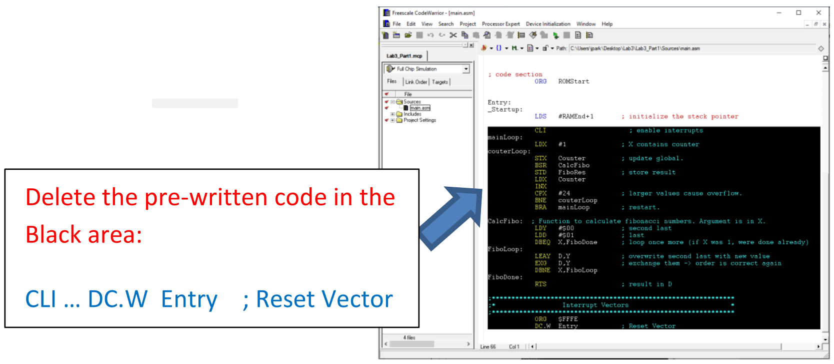


```
001010  98 2D uu uu uu uu uu uu   uu uu uu uu uu uu u
001020  03 08 09 05 07 uu uu uu   uu uu uu uu uu uu u
001030  uu uu uu uu uu uu uu uu   uu uu uu uu uu uu u
001040  uu uu uu uu uu uu uu uu   uu uu uu uu uu uu u
001050  uu uu uu uu uu uu uu uu   uu uu uu uu uu uu u
001060  uu uu uu uu uu uu uu uu   uu uu uu uu uu uu u
```

BCD result at $1020:1024

Part II: Binary to ASCII conversion

Create a new project with the name '**Lab4_part2.mcp**' as you did in the previous labs. CodeWarrior's main window opens up. 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 the main.asm provided by the CodeWarrior IDE.


Freescale CodeWarrior - [main.asm]
File Edit View Search Project Processor Expert Device Initialization Window Help
Path: C:\Users\jpark\Desktop\Lab3\Lab3_Part1\Sources\main.asm
Lab3_Part1.mcp
Full Chip Simulation
Files Link Order Targets
Sources
main.asm
Includes
Project Settings
4 files
Line 66 Col 1

```
; code section
            ORG     ROMStart


Entry:
_Startup:
            LDS     #RAMEnd+1       ; initialize the stack pointer

            CLI                      ; enable interrupts
mainLoop:
            LDX     #1              ; X contains counter
couterLoop:
            STX     Counter         ; update global.
            BSR     CalcFibo
            STD     FiboRes         ; store result
            LDX     Counter
            INX
            CPX     #24             ; larger values cause overflow.
            BNE     couterLoop
            BRA     mainLoop        ; restart.

CalcFibo:  ; Function to calculate fibonacci numbers. Argument is in X.
            LDY     #$00            ; second last
            LDD     #$01            ; last
            DBEQ    X,FiboDone      ; loop once more (if X was 1, were done already)
FiboLoop:
            LEAY    D,Y             ; overwrite second last with new value
            EXG     D,Y             ; exchange them -> order is correct again
            DBNE    X,FiboLoop
FiboDone:
            RTS                     ; result in D

;**************************************************************
;*                 Interrupt Vectors                          *
;**************************************************************
            ORG     $FFFE
            DC.W    Entry           ; Reset Vector
```

Delete the pre-written code in the Black area:

CLI … DC.W Entry   ; Reset Vector

Write an assembly program that meets the following requirements. The program converts the 2-byte hexadecimal number to a 6-byte ASCII code using the **loop structure with the comparison branch instruction**. The hexadecimal number is supplied in addresses $1010 and $1011, and the ASCII code(result) will be saved in addresses $1030-$1035.

The ten decimal digits (0, 1, …, 9) have consecutive ASCII codes, starting with $30 for '0' and ending with $39 for '9'. For example, storing 65278_{10} in memory appears as "$3**6** $3**5** $3**2** $3**7** $3**8**". The ASCII string is the simplest way to represent a series of characters (a string). There is no length information in an ASCII string. The only way to say the end of a string is ending with a special character that is **$00**. This means that an ASCII string should end with $00, which implies you need to have an extra one byte to store an ASCII string. For example, if you have five ASCII characters in a string, you should put $00 at the end, and then the string becomes six bytes.

1. Copy your code in Part1 and paste it in Part2. Next, modify your program to convert the 2-byte hexadecimal number → a 5-byte BCD number → then a 6-byte ASCII string. Each ASCII digit is calculated by adding $30 to the BCD digit.

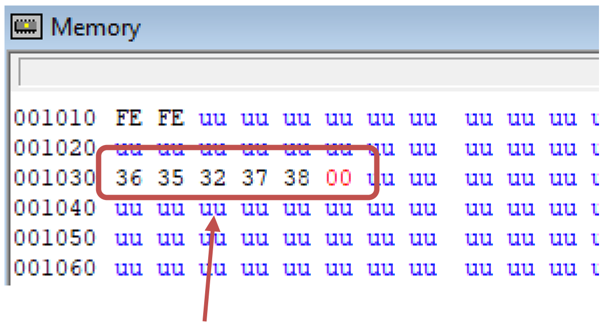
2. The 2-byte hexadecimal input number is provided in the memory locations, $1010: $1011. The ASCII result is saved in the memory locations, $1030-$1035. Modify the data section as below:

```
            ORG $1010
HexData:    DC.W     $FEFE     ; the corresponding decimal number is 65278₁₀

            ORG $1030
ASCiiNum:    DS.B    6         ; reserve 6 bytes for the ASCII output
```

3. After the last ASCII byte is calculated, the program must store $00 to indicate the end of the string. For example, if $FEFE ($65278_{10}$) is supplied as the hexadecimal number, your program should store the bytes "$36 $35 $32 $37 $38 $**00**".

6. After you complete the code, then, click on the green **debug arrow** on the menu bar. The simulator/Debugger window opens up.

7. Before running the program, click on the textbox for **the program counter (PC) in the register window and write the value "4500" for the starting address** of your program so that the processor knows where to start executing your code.
8. F11 is the single-step button. Use it to execute one instruction at a time. For the debugging, check the values in the registers, D, Y, and X. Check the ASCII output digits in the memory locations $1030:1035.
9. Take the screen shots of the ASCII outputs in Memory pane, Assembly pane and Register pane.


Memory

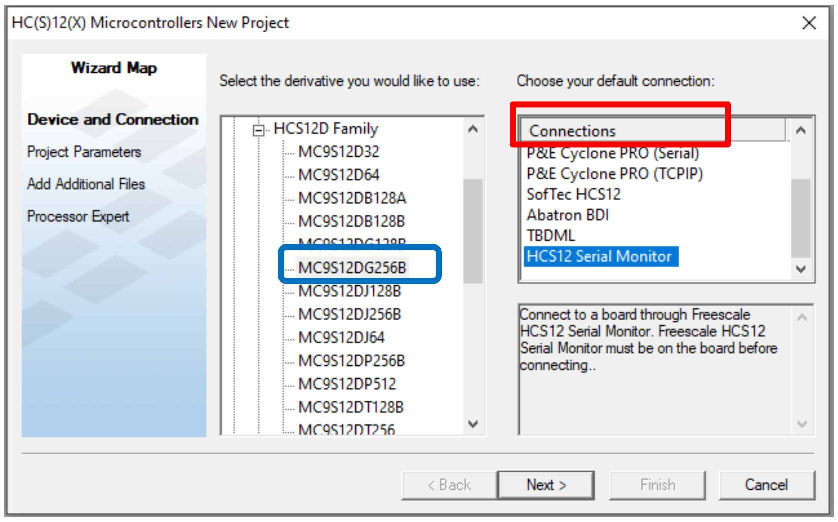
```
001010  FE FE uu uu uu uu uu uu   uu uu uu u
001020  uu uu uu uu uu uu uu uu   uu uu uu u
001030  36 35 32 37 38 00 uu uu   uu uu uu u
001040  uu uu uu uu uu uu uu uu   uu uu uu u
001050  uu uu uu uu uu uu uu uu   uu uu uu u
001060  uu uu uu uu uu uu uu uu   uu uu uu u
```

ASCII code result at $1030:1034 and ending with a special character, **$00** at $1035

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

This Part III is intended to familiarize you with Wytec's **Dragon12+ evaluation board**. **The Dragon12+ board runs the program that CodeWarrior communicates with over RS-232.** 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.


HC(S)12(X) Microcontrollers New Project

Select the derivative you would like to use:

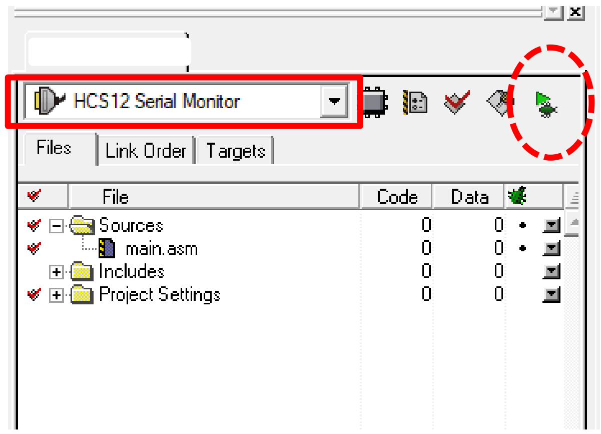
- HCS12D Family
  - MC9S12D32
  - MC9S12D64
  - MC9S12DB128A
  - MC9S12DB128B
  - MC9S12DG128B
  - MC9S12DG256B
  - MC9S12DJ128B
  - MC9S12DJ256B
  - MC9S12DJ64
  - MC9S12DP256B
  - MC9S12DP512
  - MC9S12DT128B
  - MC9S12DT256

Choose your default connection:

Connections
- P&E Cyclone PRO (Serial)
- P&E Cyclone PRO (TCPIP)
- SofTec HCS12
- Abatron BDI
- TBDML
- HCS12 Serial Monitor

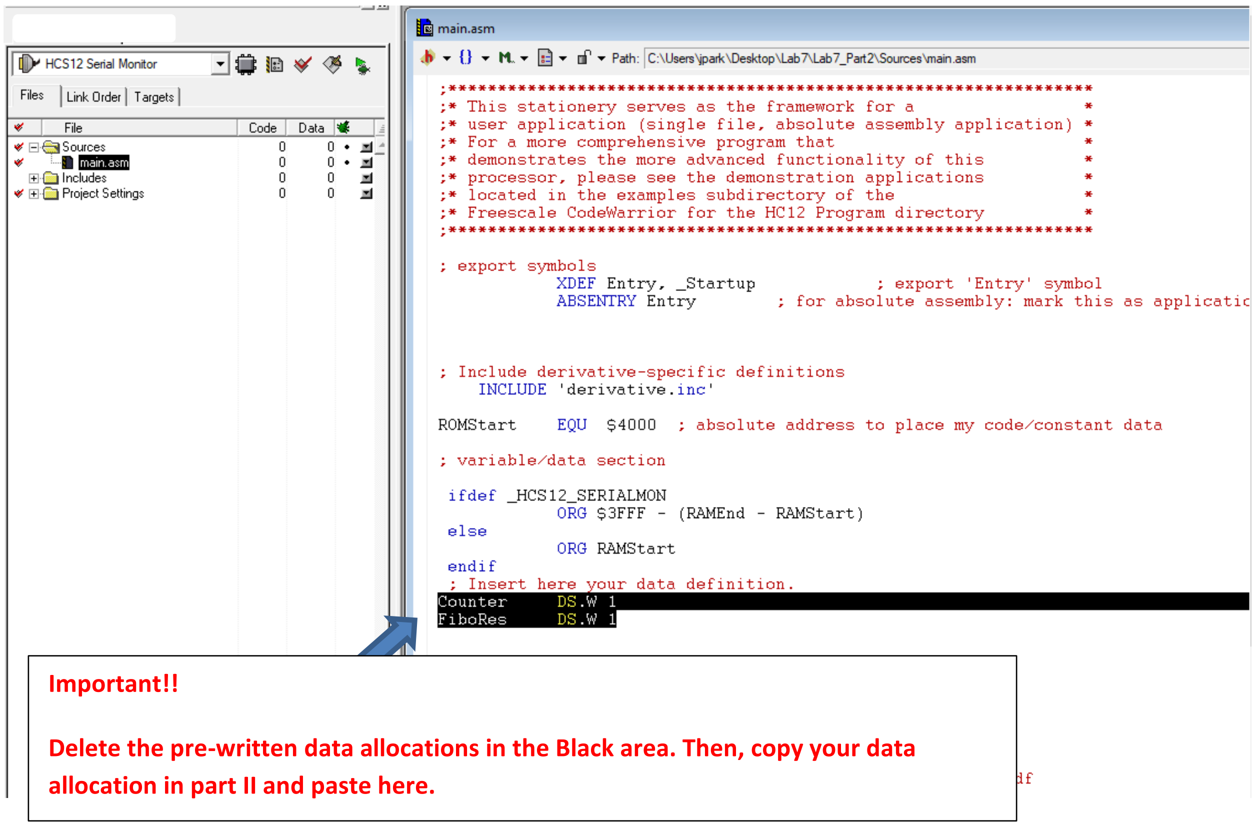
Connect to a board through Freescale HCS12 Serial Monitor. Freescale HCS12 Serial Monitor must be on the board before connecting..

2. In the **Project name** field, type **Lab4_Part3.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 in Figure5. From the dropdown list on the left, select "**HCS12 Serial Monitor**" if it is not so yet.



File	Code	Data
Sources	0	0
main.asm	0	0
Includes	0	0
Project Settings	0	0

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

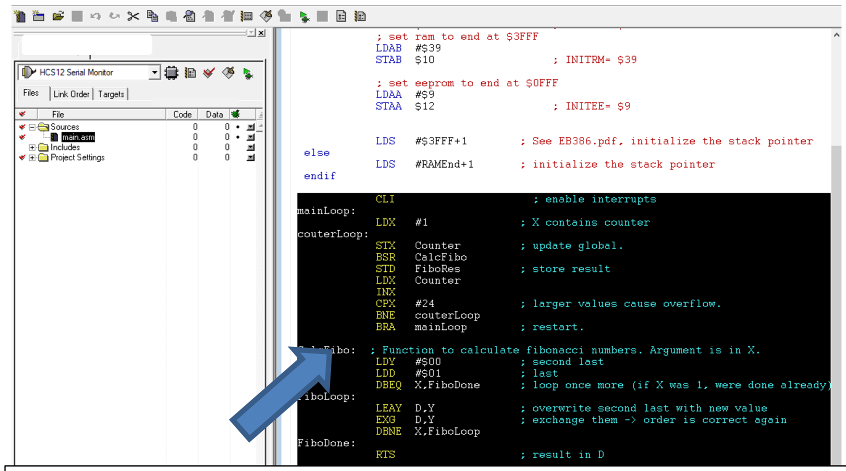

HCS12 Serial Monitor
Files
Link Order
Targets
main.asm
Path: C:\Users\jpark\Desktop\Lab7\Lab7_Part2\Sources\main.asm

```
;*******************************************************************
;* This stationery serves as the framework for a                   *
;* user application (single file, absolute assembly application) *
;* For a more comprehensive program that                           *
;* demonstrates the more advanced functionality of this            *
;* processor, please see the demonstration applications            *
;* located in the examples subdirectory of the                     *
;* Freescale CodeWarrior for the HC12 Program directory            *
;*******************************************************************

; export symbols
            XDEF Entry, _Startup            ; export 'Entry' symbol
            ABSENTRY Entry        ; for absolute assembly: mark this as applicatic


; Include derivative-specific definitions
    INCLUDE 'derivative.inc'

ROMStart    EQU  $4000  ; absolute address to place my code/constant data

; variable/data section

 ifdef _HCS12_SERIALMON
            ORG $3FFF - (RAMEnd - RAMStart)
 else
            ORG RAMStart
 endif
 ; Insert here your data definition.
Counter     DS.W 1
FiboRes     DS.W 1
```

Important!!

Delete the pre-written data allocations in the Black area. Then, copy your data allocation in part II and paste here.

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



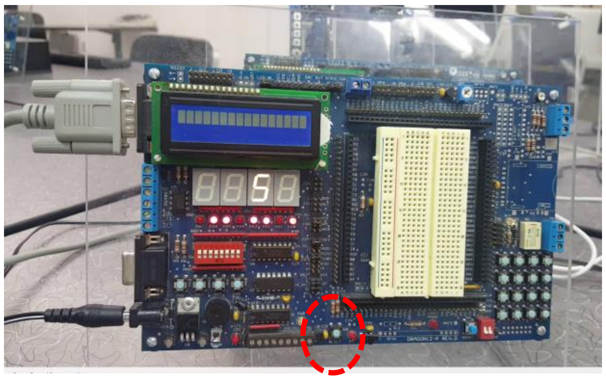
```
            ; set ram to end at $3FFF
            LDAB   #$39
            STAB   $10                   ; INITRM= $39

            ; set eeprom to end at $0FFF
            LDAA   #$9
            STAA   $12                   ; INITEE= $9


            LDS    #$3FFF+1         ; See EB386.pdf, initialize the stack pointer
 else
            LDS    #RAMEnd+1        ; initialize the stack pointer
 endif
            CLI                          ; enable interrupts
mainLoop:
            LDX    #1               ; X contains counter
couterLoop:
            STX    Counter          ; update global.
            BSR    CalcFibo
            STD    FiboRes          ; store result
            LDX    Counter
            INX
            CPX    #24              ; larger values cause overflow.
            BNE    couterLoop
            BRA    mainLoop         ; restart.

    Fibo:   ; Function to calculate fibonacci numbers. Argument is in X.
            LDY    #$00             ; second last
            LDD    #$01             ; last
            DBEQ   X,FiboDone       ; loop once more (if X was 1, were done already)
 iboLoop:
            LEAY   D,Y              ; overwrite second last with new value
            EXG    D,Y              ; exchange them -> order is correct again
            DBNE   X,FiboLoop
FiboDone:
            RTS                     ; result in D
```

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

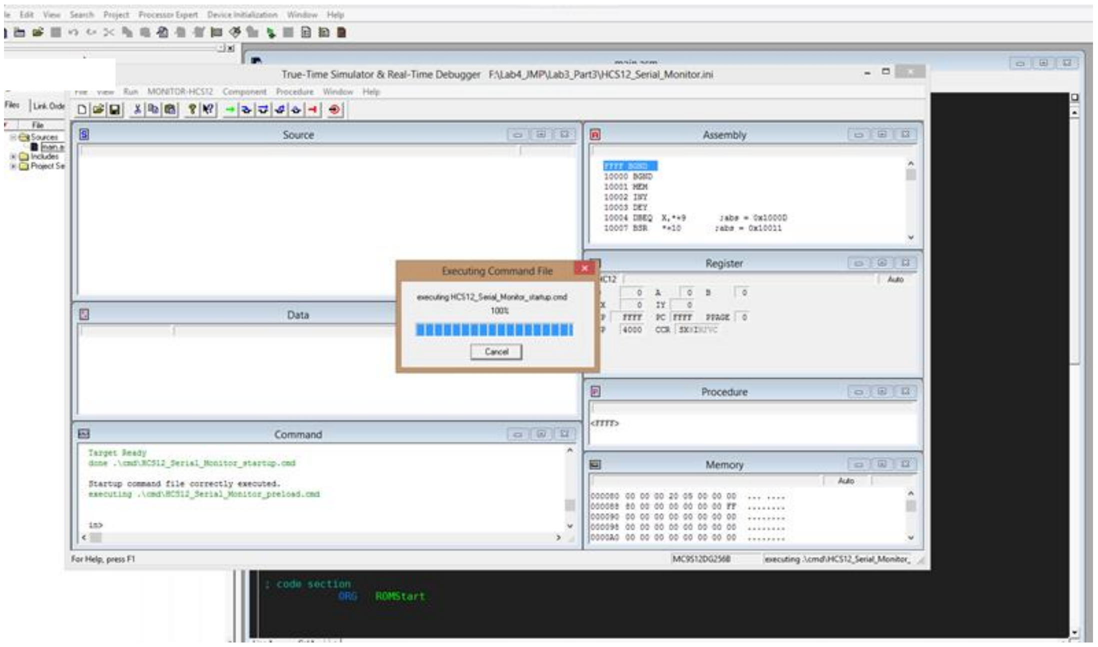
Copy your program in Lab4_partII (ORG $4500…… ExitLoop SWI) and paste here!!

Copy your code for the ASCII Code generation part in Lab4_PartII and paste here. You only need to copy your code starting from ORG $4500 ….. ExitLoop SWI.

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**.


True-Time Simulator & Real-Time Debugger F:\Lab4_JMP\Lab3_Part3\HCS12_Serial_Monitor.ini
Source
Data
Command
Assembly
Register
Procedure
Memory
Executing Command File
executing HCS12_Serial_Monitor_startup.cmd
100%
Cancel
Auto
For Help, press F1
MC9S12DG256B

```
FFFF BGND
10000 BGND
10001 MEM
10002 INY
10003 DEY
10004 DBEQ  X,*+9       ;abs = 0x1000D
10007 BSR   *+10       ;abs = 0x10011
```

```
D     0   A     0   B     0
IX    0   IY    0
SP FFFF   PC FFFF   PPAGE 0
IP 4000   CCR SXHINZVC
```

```
Target Ready
done .\cmd\HCS12_Serial_Monitor_startup.cmd

Startup command file correctly executed.
executing .\cmd\HCS12_Serial_Monitor_preload.cmd


in>
```

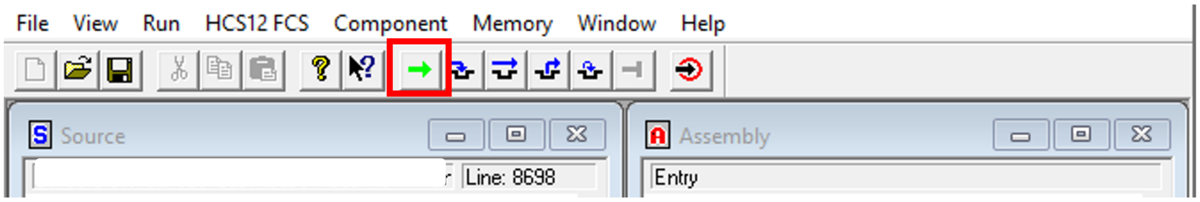
```
<FFFF>
```

```
000080  00 00 00 20 05 00 00 00   ... ....
000088  80 00 00 00 00 00 00 FF   ........
000090  00 00 00 00 00 00 00 00   ........
000098  00 00 00 00 00 00 00 00   ........
0000A0  00 00 00 00 00 00 00 00   ........
```

executing .\cmd\HCS12_Serial_Monitor_

```
; code section
            ORG     ROMStart
```

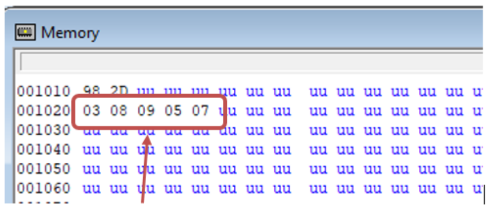
8. Before running the program, click on the textbox for **the program counter (PC) in the register window and write the value "$4500" for the starting address** of your program. Run the program from the beginning by clicking on the "Start/Continue" button.



What to Submit in Blackboard: Group submission

1) **Per each member: one pdf file** that includes
    - 25 pts: Part I: Screenshot of *the Assembly pane, the Register pane, and the Memory pane after run the program*

Example for Memory pane in PartI:

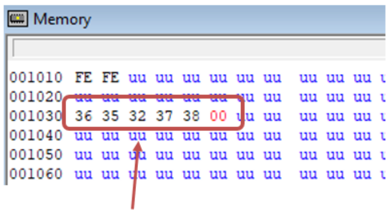


```
001010  98 2D uu uu uu uu uu uu  uu uu uu uu uu uu u
001020  03 08 09 05 07 uu uu uu  uu uu uu uu uu uu u
001030  uu uu uu uu uu uu uu uu  uu uu uu uu uu uu u
001040  uu uu uu uu uu uu uu uu  uu uu uu uu uu uu u
001050  uu uu uu uu uu uu uu uu  uu uu uu uu uu uu u
001060  uu uu uu uu uu uu uu uu  uu uu uu uu uu uu u
```

BCD result at $1020:1024

- 25pts: Part II: Screenshot of *the Assembly pane, the Register pane, and the Memory pane after run the program.*

  Example for Memory pane in Part II:



```
001010  FE FE uu uu uu uu uu uu  uu uu uu
001020  uu uu uu uu uu uu uu uu  uu uu uu
001030  36 35 32 37 38 00 uu uu  uu uu uu
001040  uu uu uu uu uu uu uu uu  uu uu uu
001050  uu uu uu uu uu uu uu uu  uu uu uu
001060  uu uu uu uu uu uu uu uu  uu uu uu
```

2) **One copy per group :**

- **25 pts: Part III - source file under "Sources" folder (main.asm file only)**
- **25 pts:** Part III: Screenshot of *the Assembly pane, the Register pane, and the Memory pane after run the program.*