Microcomputers 1

Lab 3

Loop Programming and Debugging

Winter 2022

Due Date: Tuesday midnight Feb./15/2022

Concepts:

- Basic assembly program structure: a **loop programming**
- Debugging with CodeWarrior

Objectives:

- Write a looping program to perform the given tasks
- Use trace and breakpoints to monitor a program's execution.

Total Points: 100 points

Part 0 : A looping program to add five numbers.

In Lab3, you will write an assembly program that computes the sum of the given array using the loop structure.

- **Loop:** Execute the code repeatedly until the termination condition is met.
- **Array:** a group of consecutive memory locations. Note that arrays are allocated in the *data* section of the memory. As part of the array specification, we need to know the array's length.
- **Array data processing: Indexed addressing mode** is used to access array elements inside the loop.

In part 0, the program adds a list of five one-byte numbers saved in the memory locations starting from the memory address, $3000. The **memory map for the data** is shown in Figure 1. **The one-byte addition result** will be saved in the address, $3008.

Memory address	Contents
$3000	$3A
$3001	$5E
$3002	$FE
$3003	$8B
$3004	$56
$3005	--
$3006	--
$3007	--
$3008	

Data: an array of five elements

←one byte result

Figure 1. Memory of the Data

Instead of entering data *manually* in the memory locations, you will use assembly *directives* to let CodeWarrior do it for you:

```
          ORG      $3000   ; tells an assembler to place the data in memory starting from $3000:
  Array   DC.B  $3A, $5E, $FE, $8B, $56
```

The following registers are used to calculate the sum of the array elements using the looping structure:

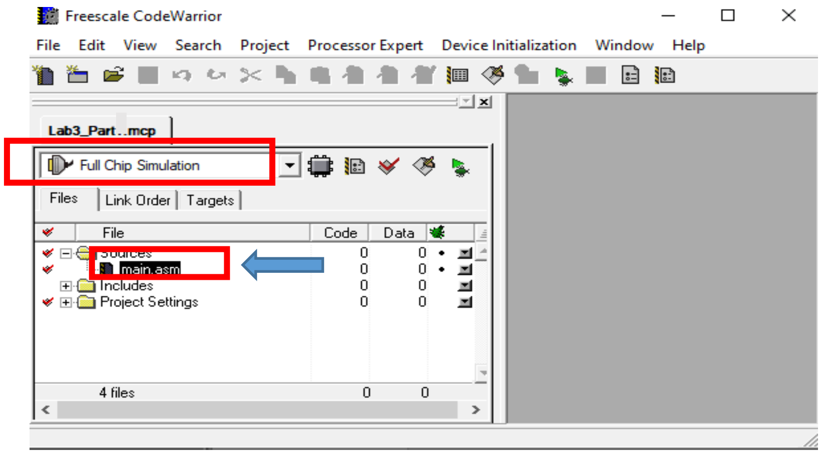
- Index register, X: hold the address of the array element
- Accumulator register, B: count the number of iterations in the looping structure
- Accumulator register, A: accumulator to hold the sum of the array elements.

Figure 2 shows a machine language program and the corresponding assembly instructions.

Code Line	Label	Assembly	Memory Address	Machine Codes
1:		CLRA	4100h	87
2:		LDX #$3000	4101h	CE
			4102h	30
			4103h	00
3:		LDAB #5	4104h	C6
			4105h	05
4:	beginLoop:	BEQ endLoop	4106h	27
			4107h	06
5:		ADDA 0,X	4108h	AB
			4109h	00
6:		INX	410Ah	08
7:		DECB	410Bh	53
8:		BRA beginLoop	410Ch	20
			410Dh	F8
9:	endLoop:	STAA $3008	410Eh	6A
			410Fh	30
			4110h	08
10:	endmain:	BRA endmain	4111h	20
			4112h	FE

Figure 2 - Program to Add Five Numbers.

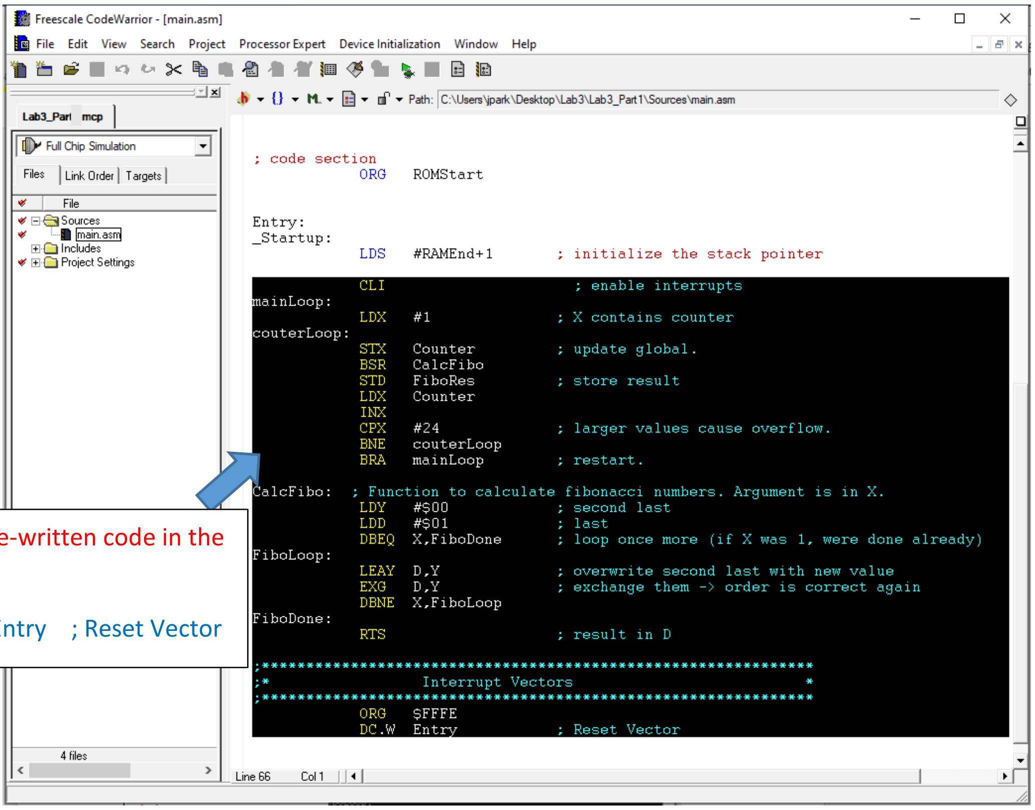
1. Create a new project with the name, '**Lab3_Par0.mcp**'. CodeWarrior main window opens up as shown in Figure3. From the dropdown list on the left, select "**Full Chip Simulation**" if it is not so yet.


Freescale CodeWarrior
File Edit View Search Project Processor Expert Device Initialization Window Help
Lab3_Part..mcp
Full Chip Simulation
Files | Link Order | Targets

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

Figure 3.

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


Freescale CodeWarrior - [main.asm]
File Edit View Search Project Processor Expert Device Initialization Window Help
Lab3_Part mcp
Full Chip Simulation
Files Link Order Targets
Sources
main.asm
Includes
Project Settings
4 files
Path: C:\Users\jpark\Desktop\Lab3\Lab3_Part1\Sources\main.asm
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

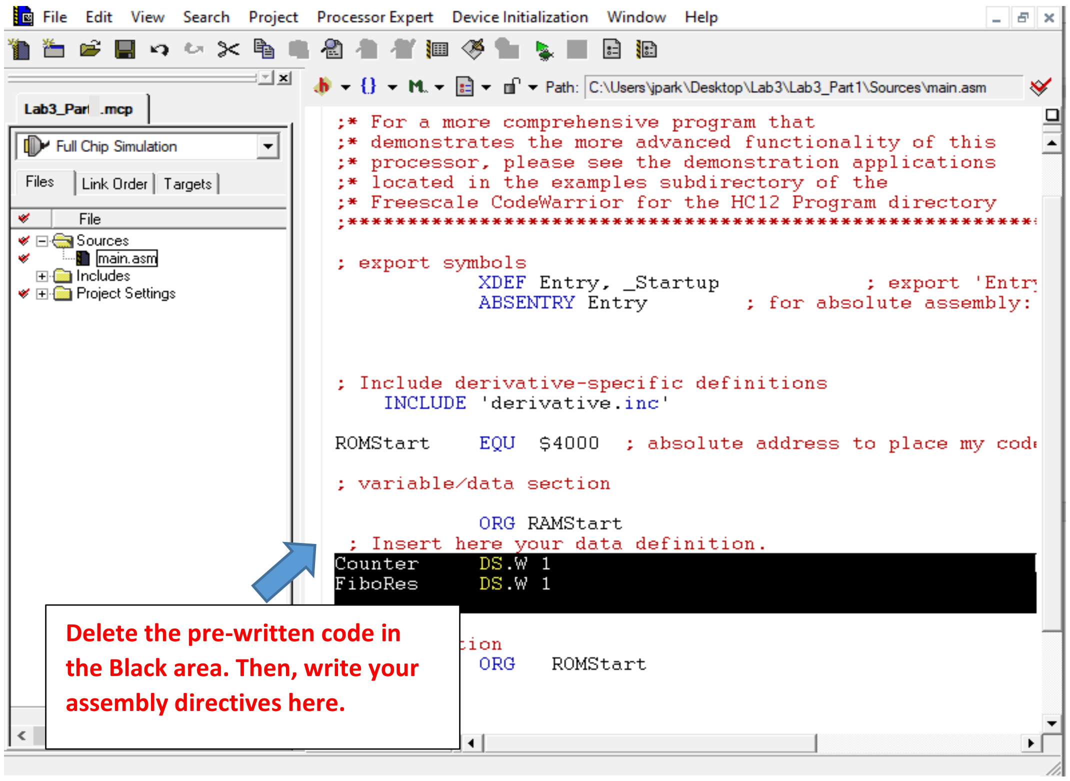
Figure 4.

2. Write the assembly code in the table of Figure 2 in main.asm. To start your assembly instruction at the memory location $4100, you need to **add the assembly directive ORG $4100 just before the code line 1.**

3. Using assembly *directives,* defined the array data in the data section of the main.asm file.

```
;********** "Enter your data here:"
            ORG      $3000      ; tells assembler to place the items in memory starting at $3000:
Array      DC.B  $3A, $5E, $FE, $8B, $56
```

The data definitions are placed after ORG RAMStart. First, **delete** the prewritten sample code provided by the CodeWarrior IDE (marked with black background) in Figure 5. Type the above assembly directives in main.asm.

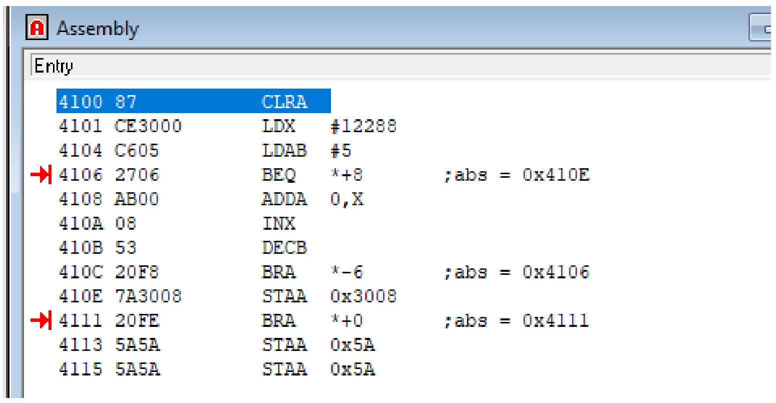

File Edit View Search Project Processor Expert Device Initialization Window Help
Path: C:\Users\jpark\Desktop\Lab3\Lab3_Part1\Sources\main.asm
Lab3_Part .mcp
Full Chip Simulation
Files
Link Order
Targets
File
Sources
main.asm
Includes
Project Settings

```
;* 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 'Entr
            ABSENTRY Entry          ; for absolute assembly:


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

ROMStart    EQU  $4000  ; absolute address to place my code

; variable/data section

            ORG RAMStart
 ; Insert here your data definition.
Counter     DS.W 1
FiboRes     DS.W 1

        tion
            ORG    ROMStart
```

Delete the pre-written code in the Black area. Then, write your assembly directives here.

Figure 5

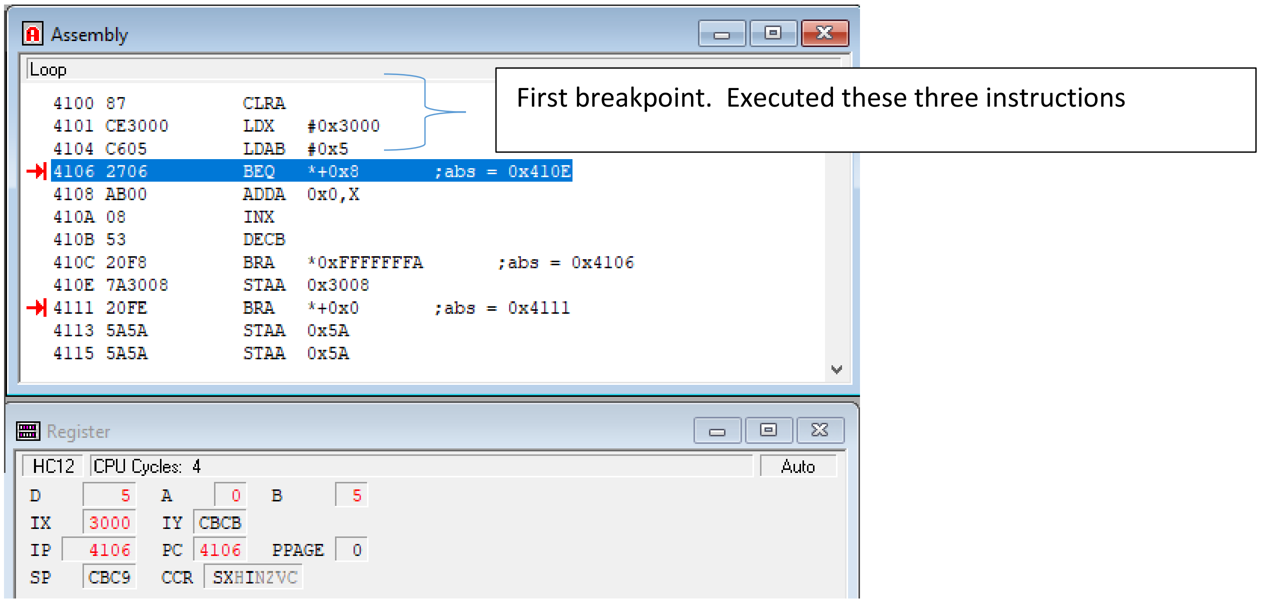
4. Then, click on the green **debug arrow** on the menu bar. The simulator/Debugger window opens up.
5. Before running the program, click on the textbox for **the program counter (PC) in the register window and write the value "4100" for the starting address** of your program so that the processor knows where to start executing your code.
6. Add a breakpoint at the beginning of the loop corresponding to the BEQ instruction "$4106" and **the final instruction's address** "$4111" so that the debugger stops at a logical point for the user's inspection. Breakpoints are set by right-clicking on the instruction and selecting "Set Breakpoint".



```
4100 87          CLRA
4101 CE3000      LDX    #12288
4104 C605        LDAB   #5
4106 2706        BEQ    *+8       ;abs = 0x410E
4108 AB00        ADDA   0,X
410A 08          INX
410B 53          DECB
410C 20F8        BRA    *-6       ;abs = 0x4106
410E 7A3008      STAA   0x3008
4111 20FE        BRA    *+0       ;abs = 0x4111
4113 5A5A        STAA   0x5A
4115 5A5A        STAA   0x5A
```

Figure 6. Break points

7. Run the program from the beginning. Note that when the processor stops at a breakpoint, the instruction that the PC points to has not yet been executed.



First breakpoint. Executed these three instructions

```
4100 87          CLRA
4101 CE3000      LDX    #0x3000
4104 C605        LDAB   #0x5
4106 2706        BEQ    *+0x8       ;abs = 0x410E
4108 AB00        ADDA   0x0,X
410A 08          INX
410B 53          DECB
410C 20F8        BRA    *0xFFFFFFFA       ;abs = 0x4106
410E 7A3008      STAA   0x3008
4111 20FE        BRA    *+0x0       ;abs = 0x4111
4113 5A5A        STAA   0x5A
4115 5A5A        STAA   0x5A
```

```
HC12  CPU Cycles: 4                                   Auto
D      5    A     0    B        5
IX  3000    IY CBCB
IP  4106    PC 4106    PPAGE    0
SP  CBC9    CCR SXHINZVC
```

8. A breakpoint trace is similar to a program trace, except that instead of recording the register values after each line of code is executed, we will record the register values when the processor stops at the breakpoint. Complete the breakpoint trace in Fig. 7, given that you are currently at iteration 0, and the first time that the program returns to the same BEQ instruction is considered iteration 1. Clicking on the "**Start/Continue**" button will resume program execution and halt at the next breakpoint.

Iteration	Code Line		PC	X	A	B	N	Z	V	C
0	4: **BEQ endLoop**		$4106	$3000	0	5	0	0	0	0
1	4: **BEQ endLoop**									
2	4: **BEQ endLoop**									
3	4: **BEQ endLoop**									
4	4: **BEQ endLoop**									
5	4: **BEQ endLoop**									
	10:BRA endmain									

Figure 7. Breakpoint Trace

Part I: Programming Assignment: Working with Array

Create a new project with the name, '**Lab3_Part1.mcp**'. *Write an assembly program that computes the sum of the given array using the loop structure.*

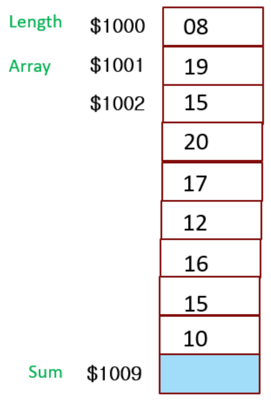
1. In Part2, the array is defined with the length of the array and the element values in the array. The array definition using assembly directives is shown below:

```
;***********Enter your data here:
                ORG    $1000
Length:         DC.B    8                                  ; the length of the array
Array:          DC.B    19, 15, 20, 17, 12, 16, 15, 10     ; elements in the array
Sum:            DS.B    1                                  ; the sum is stored here
```

   Complete the assembly program to calculate the sum of the given array, and place the sum of the array in the memory location also reserved in the above data structure. Let us assume that

   - 0 < Number of elements in the given array < 11
   - Each element < 21, and 0 < (sum of the given array)

2. In the above the data definition, you defined several labels. The label, **'Length'** is a meaningful name of the address $1000, '**Array'** is a another name of the address $1001, and '**Sum**' is the address $1009 as shown in the figure below:



Length =$1000

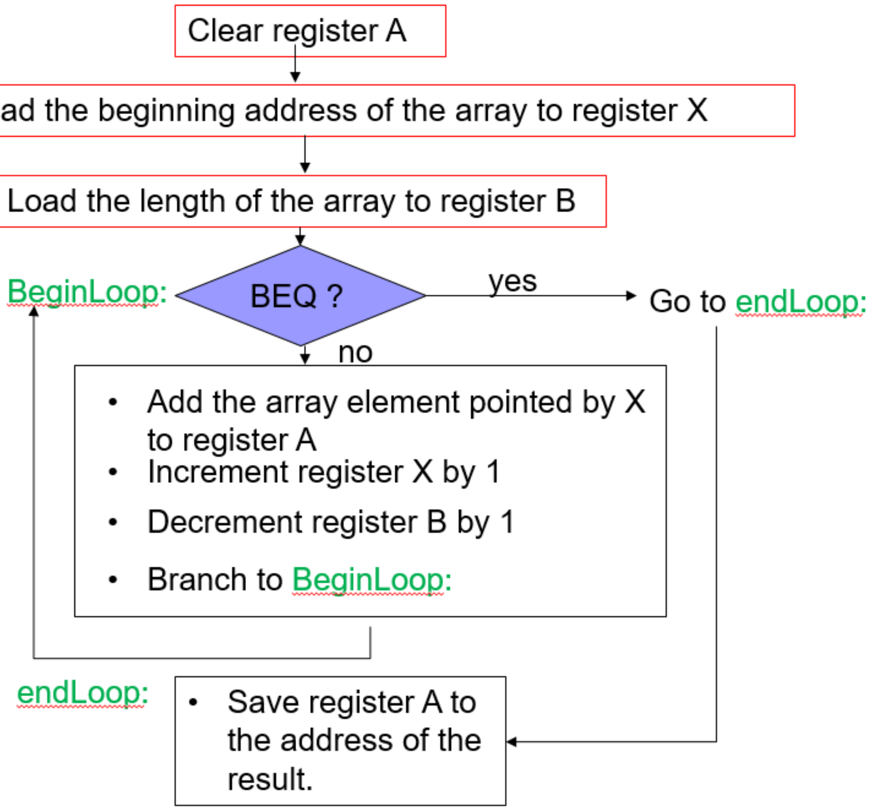
Array =$1001

Sum =$1009

Now, you can use these labels in your program. For example, instructions below are equivalent.

```
LDAB   $1000     →   LDAB   Length
LDX    #$1001    →   LDX    #Array
STAA   $1009     →   STAA   Sum
```

The flowchart of Part I assignment is presented below:


Clear register A
Load the beginning address of the array to register X
Load the length of the array to register B
BeginLoop:
BEQ ?
yes
Go to endLoop:
no
• Add the array element pointed by X to register A
• Increment register X by 1
• Decrement register B by 1
• Branch to BeginLoop:
endLoop:
• Save register A to the address of the result.

3. To start your assembly instruction at the memory location $4100, you need to **add the assembly directive ORG $4100 just before your code.** And also add the below code at the end of your code.

    endMain   BRA   endMain

4. After completion of your code, click on the green **debug arrow** on the menu bar. The simulator/Debugger window opens up.
5. Add breakpoints at the beginning of the loop corresponding to the BEQ instruction and at **the final instruction, endMain   BRA   endMain** so that the debugger stops at a logical point for the user's inspection.

6. Before running the program, click on the textbox for **the program counter (PC) in the register window and write the value "4100" for the starting address** of your program so that the processor knows where to start executing your code.

Q1: Where is the sum value saved in the memory location ? What is the sum of the given array in hex and in decimal?

Part II. Programming Assignment:

***A program to compute the sum of the first N natural numbers (i.e., Sum =N + (N-1) + … + 2 +1)**.*

1. Create a new project with the name '**Lab3_Part2.mcp**'. *Write an assembly program that computes the sum of the first N natural numbers. Assume N is one-byte unsigned integer number whose value is given at memory location **$1000**. You may also assume that the value of N does not exceed 20 ($14 in Hex), so you could safely store (without overflow) **the result of the sum** in a one-byte memory location at address **$1001.***
2. *The below assembly directives assign the N value in the location $ 1000* (= RAMStart) *and reserve the one byre space for the sum at the location, $1001.*

```
            ORG RAMStart
   ; Insert here your data definition.
  Num:    DC.B   10        ; N=10   (N is assigned in the memory location, $1000)
  Sum:    DS.B   1         ; Memory location for the sum (N + N-1 + ....+2+1)
```

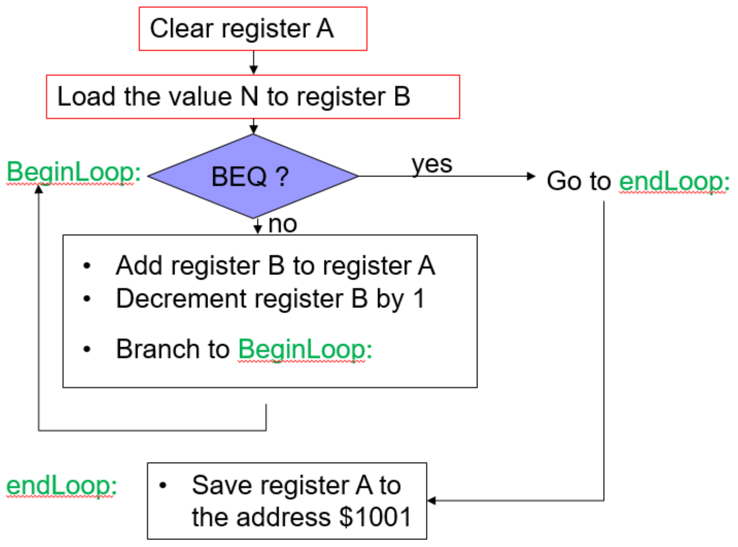
To calculate the sum of the first N natural numbers using the looping structure, the following registers are used:

- Accumulator register, B: count the number of iterations in the looping structure
- Accumulator register, A: accumulator to calculate the sum of the first N natural numbers.

Hint: To calculate sum of the first N natural numbers, use the instruction

```
        ABA            ; add register B to register A every iteration inside the loop
```

See the below flowchart for Part II assignment.


Clear register A
Load the value N to register B
BeginLoop:
BEQ ?
yes
Go to endLoop:
no
• Add register B to register A
• Decrement register B by 1
• Branch to BeginLoop:
endLoop:
• Save register A to the address $1001

3. To start your assembly instruction at the memory location $4100, you need to **add the assembly directive ORG $4100 just before your code.** And also add the below code at the end of your code.

   endMain BRA endMain

4. After completion of your code, click on the green **debug arrow** on the menu bar. The simulator/Debugger window opens up.
5. Add breakpoints at the beginning of the loop corresponding to the BEQ instruction and at **the final instruction, endMain BRA endMain** so that the debugger stops at a logical point for the user's inspection.

6. Before running the program, click on the textbox for **the program counter (PC) in the register window and write the value "4100" for the starting address** of your program so that the processor knows where to start executing your code.

Q2: Where is the sum value saved in the memory location ? What is the ***sum of the first N natural numbers*** in hex and in decimal?

What to Submit in Blackboard: Group submission

1) **Per each member: one pdf file** that includes
   - 30 pts : Complete the work book, **Lab3_workbook.doc**
     - *Fill the table in Figure 7*
     - *Provide the answer for Q1 and Q2.*
   - 10 pts: Part0: Screenshot of *the Assembly pane, the Register pane, and the Memory pane after run the program.*
   - 10 pts: Part1: Screenshot of *the Assembly pane, the Register pane, and the Memory pane after run the program.*

2) **One copy per group : 50 pts**
50 pts: Part2 - source file under "Sources" folder (main.asm file only)