Temple University

College of Engineering

Department of Electrical and Computer Engineering (ECE)

Student Lab Report Cover Page

Course Number : 3613

Course Section : 001 / 002

Experiment # : Lab #3

Student Name (print) : Von Kaukeano

TUid# : 915596703

Date : 9/19/19

Grade : _____ **/100**

TA Name : Sung Choi

ACTIVITIES:

Activity 1

Write assembly codes to perform the given instructions (40 pts).

1.1 Load the number 57 in decimal directly to the three GPRs. You select three GPRs and load the number 57 with three different number format (20 pts).

Requirements:

1) Load the decimal number 57 in the first GPR

2) Load the hex format of the number 57 in the second GPR

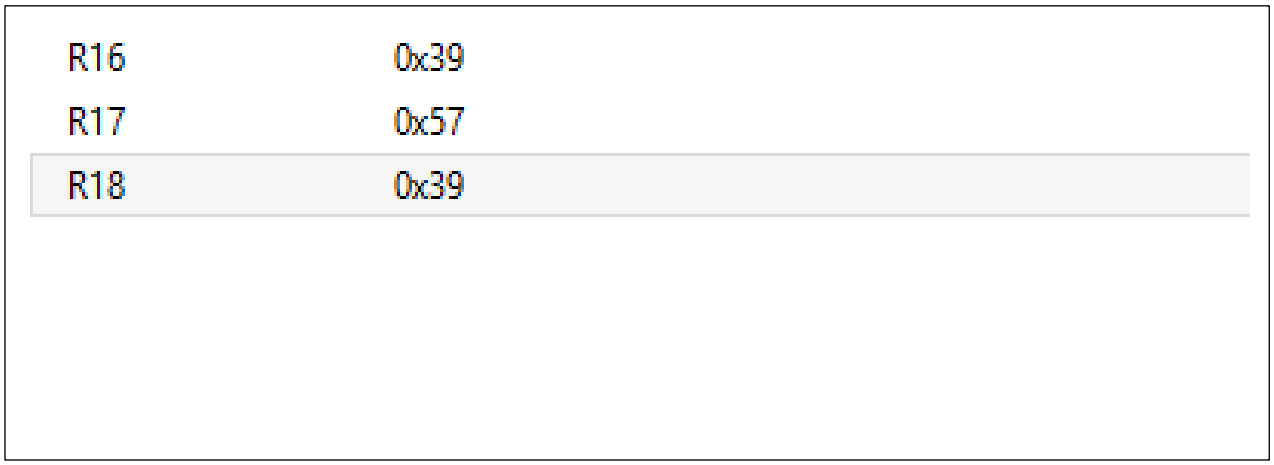
3) Load the binary format of the number 57 in the third GPR.

- Your code must show three different formats of the number 57 in decimal in three different registers.
- You must put comments for each line of code.
- Your code must stop at the end of the code with the infinite loop using a label (ex. here: rjmp here)

CODE:

```
1-1
; Lab3.asm
;
; Created: 9/19/2019 9:03:55 AM
; Author : Von Kaukeano
;
; Replace with your application code
start:
   LDI R16, 57 //DECIMAL
   LDI R17, $57 //HEX
         LDI R18,0B0111001 //BINARY
here: jmp here
```

RESULT: Screenshot of the resulted GPR or memory values



1.2 Load the number $9A to the data memory location 0x0100 and load the number $B2 to the data memory location 0x0150. Then, swap the values of the memory locations (20 pts).

Requirements:

1) Load the number $9A in 0x0100

2) Load the number $B2 in 0x0150

3) Swap the numbers, so the final values of the locations must be $B2 in 0x0100 and $9A in 0x0150.

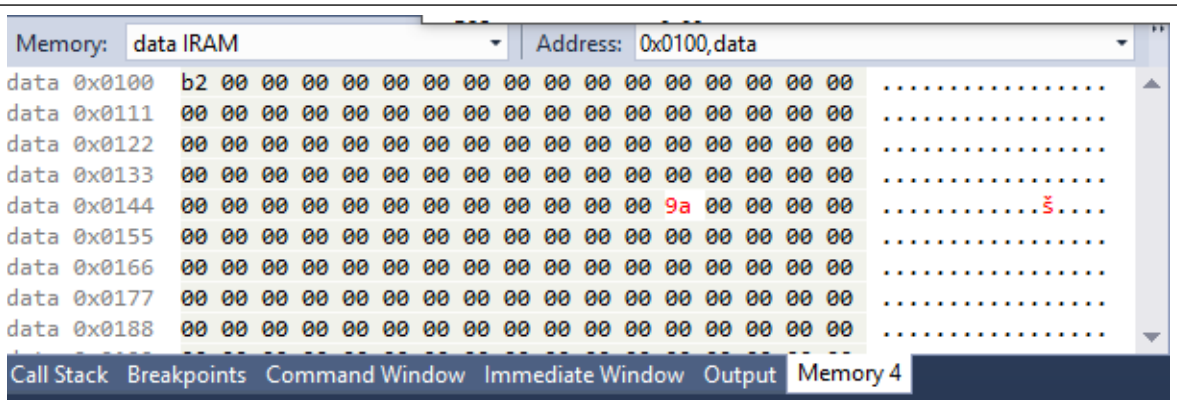
- Your code must show the loading the values in the memory locations and the process of swapping the contents of the location 0x0100 and 0x0150.
- You must put comments for each line of code.
- Your code must stop at the end of the code with the infinite loop using a label (ex. here: rjmp here)

CODE:

```
start:
   LDI R16, $9A // LOAD R16
        LDI R17,$B2 // LOAD R17
        STS 0X0100, R16 // LOAD SRAM WITH R16
        STS 0X0150, R17 // LOAD SRAM WITH R17

        STS 0X0100,R17 // LOAD SRAM WITH R17
        STS 0X150,R16 // LOAD SRAM WITH R16
        here: jmp here
```

RESULT: Screenshot of the resulted GPR or memory values



Activity 2

Write assembly codes to perform the basic arithmetic operations and observe the flags related the arithmetic operations (30 pts).

2.1 Add the numbers, $41 and $E8 and store the sum in the memory location 0x0110. Find the flags that change and list them (10 pts).

Requirements:

1) Load $41 and $E8 in GPRs

2) Add the numbers

3) Store the sum in the memory location 0x0110

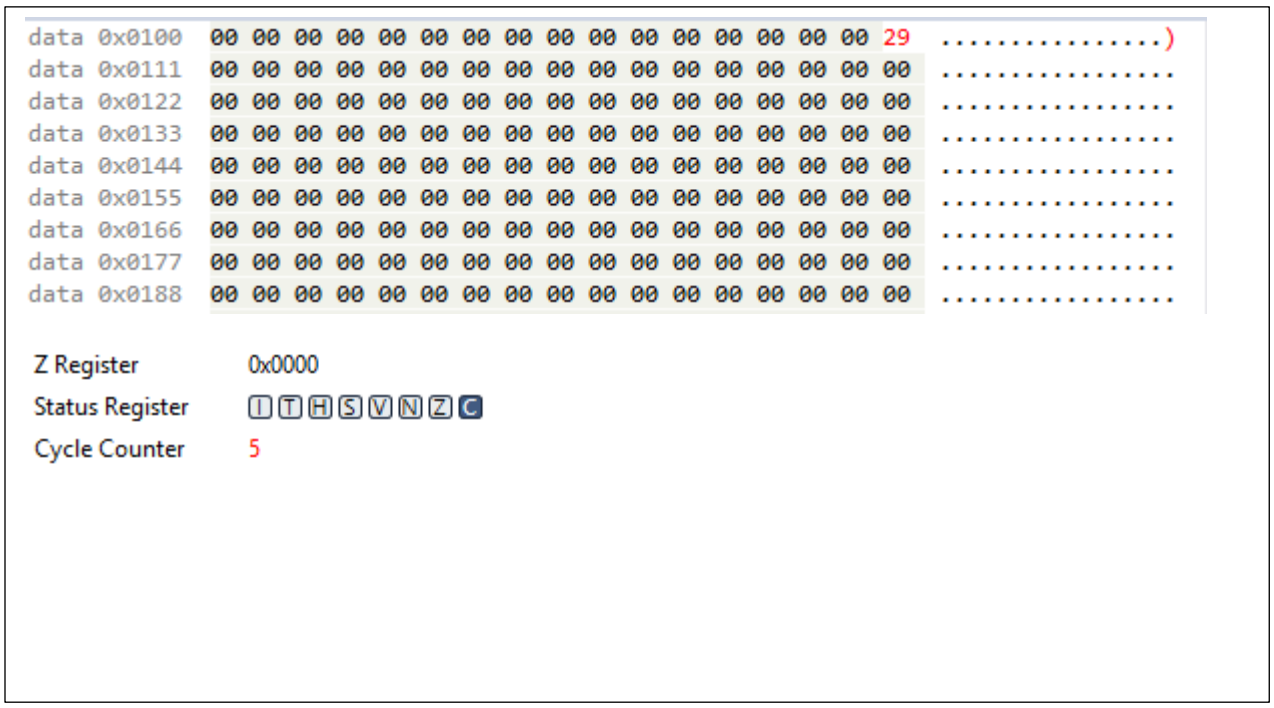
4) Find the flags that set and list them

- Your code must show the complete operation.
- Your result must show the flags that change when the arithmetic operation is completed.
- You must put comments for each line of code.
- Your code must stop at the end of the code with the infinite loop using a label (ex. here: rjmp here)

CODE:

```
start:
   LDI R16, $41 // LOAD R16
         LDI R17,$E8 // LOAD R17
         ADD R16,R17 // ADD REGISTERS
         STS 0X0110, R16 // LOAD SRAM WITH R16
         here: jmp here
```

RESULT: Screenshot of the resulted GPRs, status register (flags), and memory values



2.2 Store the number $59 to the location 0x0200 and store the 2's complement of $59 to the location 0x0201. Fine the flags that change when you perform the arithmetic operation and list them (10 pts).

Requirements:

1) Load $59 in a GPR and store the number in the memory location 0x0200

2) Find the 2's complement of $59 and store in the memory location 0x0201

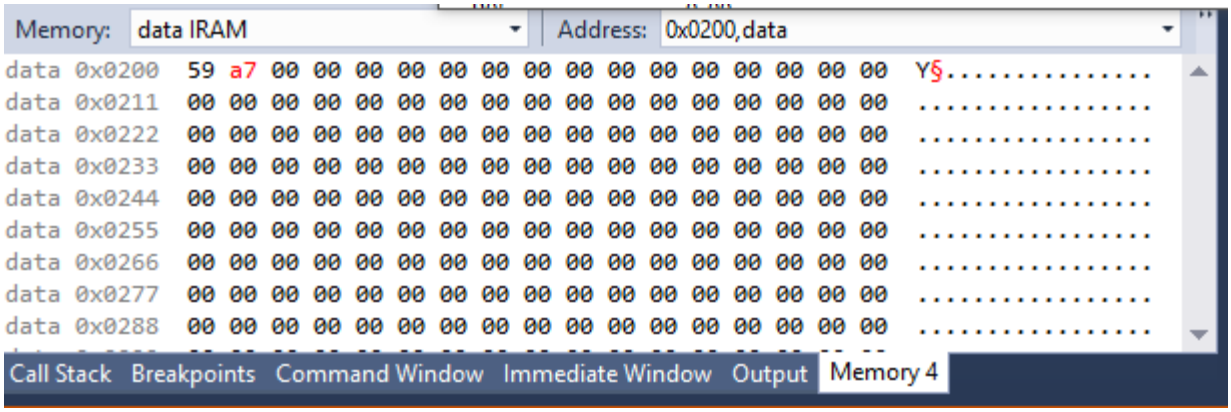
3) Find the flags that set, and list them

- Your code must show the complete operation.
- Your result must show the flags that change when the arithmetic operation is completed.
- You must put comments for each line of code.
- Your code must stop at the end of the code with the infinite loop using a label (ex. here: rjmp here)

CODE:

```
start:
   LDI R16, $59 // LOAD R16
        LDI R17, $59 // LOAD R17
        STS 0X0200, R16 // LOAD SRAM WITH R16
        NEG R17 // TWOS COMPLEMENT
        STS 0X201, R17 // LOAD SRAM WITH R17
        here: jmp here
```

RESULT: Screenshot of the resulted GPRs, status register (flags), and memory values



2.3 Subtract $8 from $10 twice and find the flags that change when you perform the arithmetic operation and list them (10 pts).

Requirements:

1) Load $8 and $10 in GPRs

2) Subtract $8 from $10, then find the flags that set

3) Subtract $8 from the result of 2), then find the flags that set

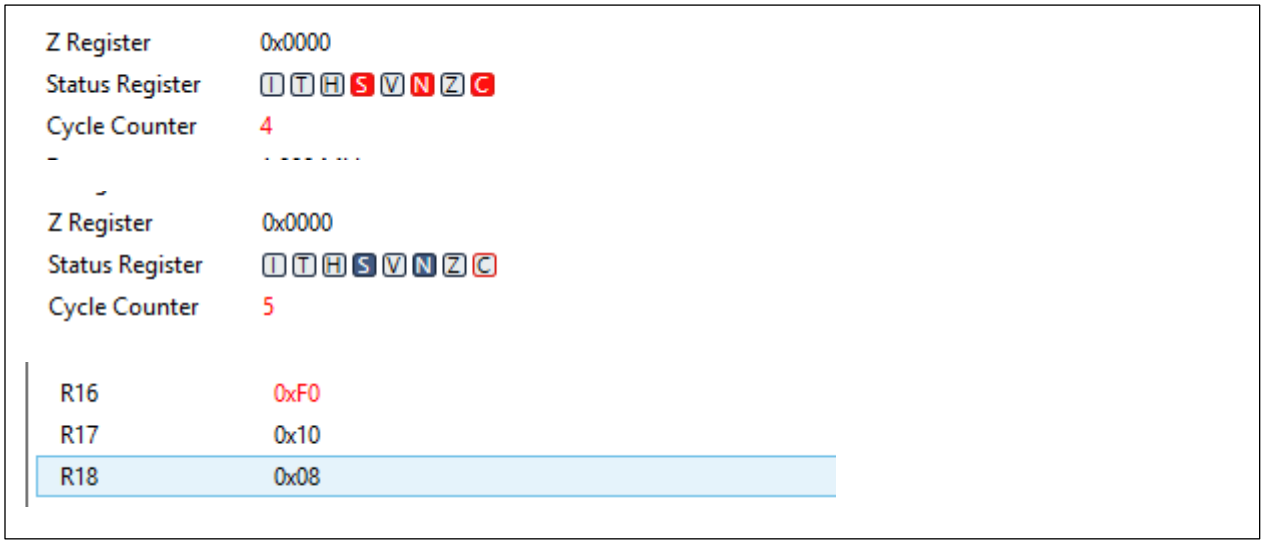
- Your code must show the complete operation.
- Your result must show the flags that change when the arithmetic operation is completed.
- You must put comments for each line of code.
- Your code must stop at the end of the code with the infinite loop using a label (ex. here: rjmp here)

CODE:

```
start:
  LDI R16, $8 // LOAD R16
        LDI R17, $10 // LOAD R17
        LDI R18, $8 // LOAD R18
        SUB R16, R17 //  R16 -> R16-R17
        SUB R16, R18 // R16 -> R16-R18

        here: jmp here
```

RESULT: Screenshot of the resulted GPRs, status register (flags), and memory values



Activity 3

Write assembly codes to perform the iterations required to complete the task using the AVR branching instructions (30 pts).

Store $6 in the memory location 0x0100. Decrease the value by 1 and store each value in the next location. Complete the task with two branch instructions and labels. **The branching instructions must use different flags**. Explain the branching instructions you select.

You write two separate codes with each branch instruction.

Requirements:

1) Load $5 in a GPR and store the value in the memory location 0x0100

2) Decrement the value by 1 each loop and store the decreased values in the memory locations (see the following order)

0x0100 = $5
0x0101 = $4
0x0102 = $3
0x0103 = $2
0x0104 = $1
0x0105 = $0

3) Select one AVR branching instruction and write an assembly program to accomplish the task.

4) Select another AVR branching instruction that uses a different flag from the previous branching instruction in 3) and write another assembly program to accomplish the task.

- You must explain the selected branching instructions and their flags.
- You must also show the values in the specified memory locations at the end of the operation.
- You must put comments for each line of code.
- Your code must stop at the end of the code with the infinite loop using a label (ex. here: rjmp here)

The code block to increment the address value in the loop is given below.

**Code for increment the address location

```
        ldi r16, $5          ;load $5 to r16
        sts 0x100, r16       ;store r16 in 0x100
        ldi xl, 0x00         ;assign lower byte of the address (16 bits) to the x
        pointer
                              lower byte location
        ldi xh, 0x01         ;assign higher byte of the address (16 bits) to the x
        pointer
                              higher byte location

op:     st x+, r16           ;increase the pointer value (address where the pointer
        is
                               pointing)

        ;Your code to decrement the value of r16 and to control the loop
        here

here:   jmp here             ;stay here forever
```

CODE 1.

Explaining of the branching instruction for the code 1:

What is the branching instruction used for your code to control the loop?

I used BRNE.

What is the flags used for the instruction?

Z flag.

What is the condition to exit the loop?

The Z flag becomes one when the register has a zero value inside.

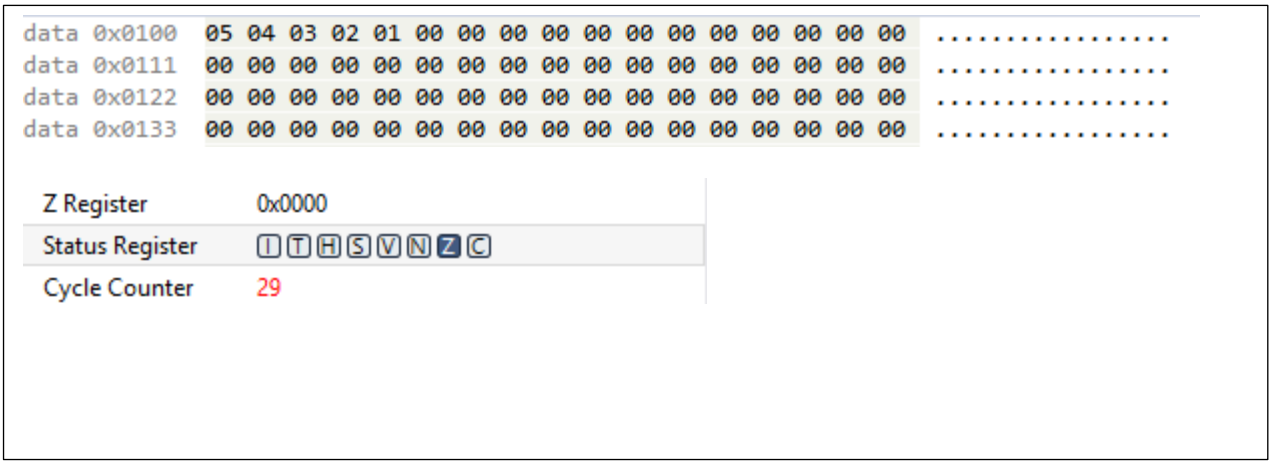
CODE 1:

```
start:
   LDI R16, $5 // LOAD R16
        STS 0X0100, R16 // STORE R16 IN 0X0100
        LDI xl,0X00 // ASSIGN LOWER BYTE OF THE ADDRESS TO THE X POINTER
        LDI Xh,0X01 // ASSIGN HIGHER BYTE OF THE ADDRESS TO THE X POINTER
OP: ST X+,R16 // INCREASE POINTER VALUE
        DEC R16 // DECREASE R16

BRNE OP // LOOP UNTIL Z = 1, R16 = 0

        here: jmp here
```

RESULT of CODE 1: Screenshot of the resulted memory values



CODE 2.

Explaining of the branching instruction for the code 2:

What is the branching instruction used for your code to control the loop?

I used BRPL.

What is the flags used for the instruction?

N flag.

What is the condition to exit the loop?

The N flag becomes one when the number in the register becomes negative.

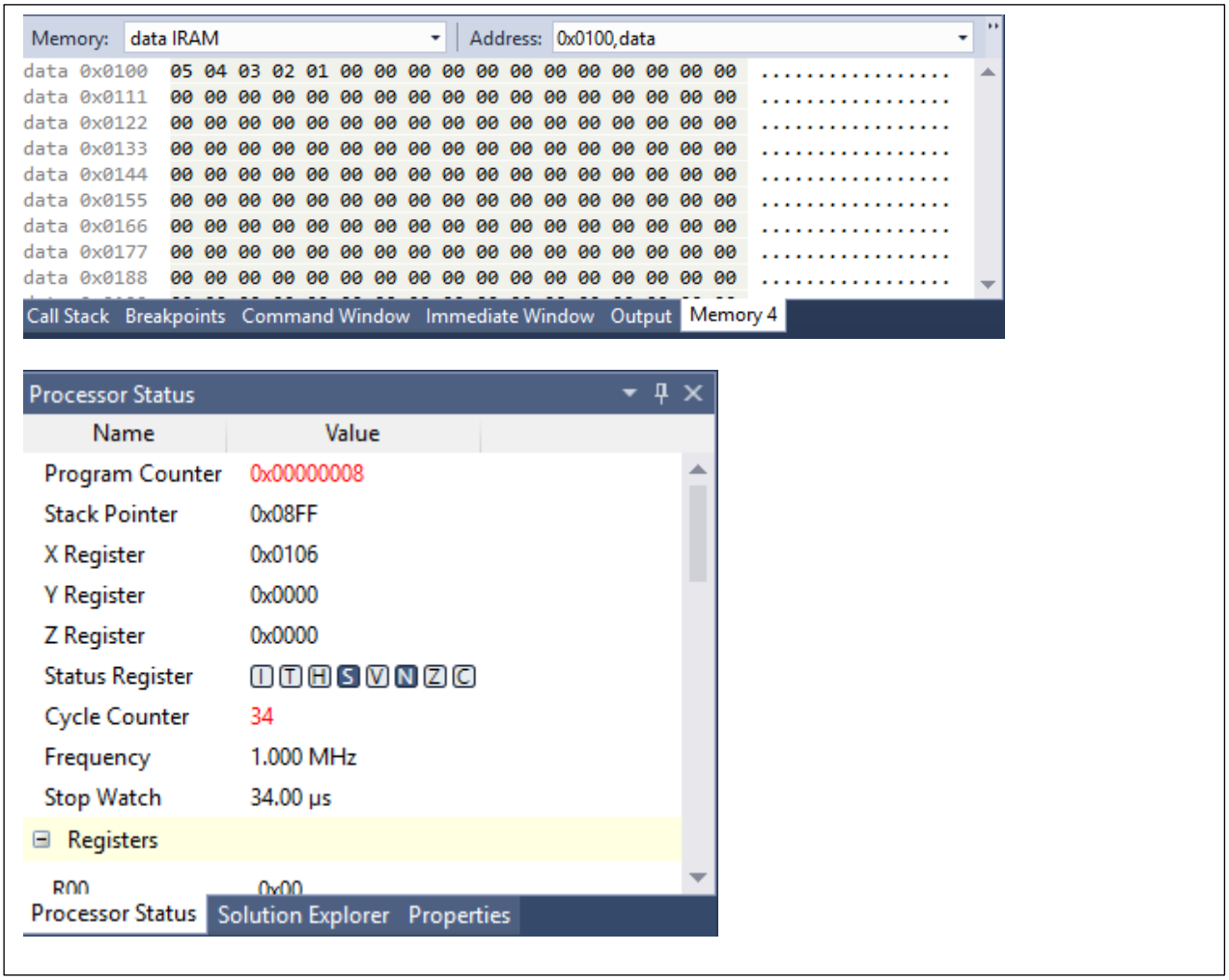
CODE 2:

```
    LDI R16, $5 // LOAD R16
        STS 0X0100, R16 // STORE R16 IN 0X0100
        LDI xl,0X00 // ASSIGN LOWER BYTE OF THE ADDRESS TO THE X POINTER
        LDI Xh,0X01 // ASSIGN HIGHER BYTE OF THE ADDRESS TO THE X POINTER
OP: ST X+,R16 // INCREASE POINTER VALUE
        DEC R16 // DECREASE R16

BRPL OP // LOOP UNTIL N = 1, S = 1

        here: jmp here
```

RESULT of CODE 2: Screenshot of the resulted memory values



ECE3613 Processor System Laboratory Rubric
Lab #: 3
Section: 001 / 002
Name: _____

Activity	Section	Task	Full Points	Earned Points	Comment
1	1.1	Code	10		
		Result	10		
	1.2	Code	10		
		Result	10		
Subtotal			40		
2	2.1	Code	5		
		Result	5		
	2.2	Code	5		
		Result	5		
	2.3	Code	5		
		Result	5		
Subtotal			30		
3	Code 1	Explanation	5		
		Code	5		
		Result	5		
	Code 2	Explanation	5		
		Code	5		
		Result	5		
Subtotal			30		
Total			100		