Experimental Procedure:

Lab 3.1.1: Straight Line

Create a new project for this lab. Write and test an assembly language program that solves the following straight line equation for y:

$$y = mx + b$$

Generate an integer result as close as possible that solves the following straight-line equation for y. Generate an integer result as close as possible to the actual answer. Assume variable x is always an unsigned 8-bit number.

The slope (m) is equal to 0.68 and the offset (b) is equal to 12. Leave the result in accumulator A. Assume value x is in the memory location labeled `Val`. Use assembler directives to define these values. Note that the result will always fit in 8 bits.

Test and confirm the program is working properly by testing the following values in `Val`. Modify `Val` using the data window in the debugger.

`Val = 0` Result: 12

`Val = 10` Result: 18

`Val = 75` Result: 63

`Val = 200` Result: 148

`Val = 255` Result: 185

Extra Credit: The number after the decimal point will always be truncated off of the result. The result will not be rounded up. For extra credit on the lab checkout, display the number to the left of the decimal point in register A and the result to the right of the decimal point in register B.

Lab 3.1.2: Parabolic Line

Write and test an assembly language program that solves the following equation for y. Assume variable x is an unsigned 8-bit number.

$$y = mx^2 + b$$

The slope (m) is equal to 0.68 and the offset (b) is equal to 12. Leave the result in register D. Assume value x is in the memory location labeled `Val`. Use assembler directives to define these values. Note that the result will always fit in 16 bits.

Test and confirm the program is working properly by testing the following values in `Val`. Modify `Val` using the data window in the debugger.

`Val = 0` Result: 12

`Val = 10` Result: 80

`Val = 75` Result: 3837

`Val = 200` Result: 27212

`Val = 255` Result: 44229

Hint: Using the scaling approach outlined in this experiment will result in a number that is larger than the available registers of the HC(S)12. The above program can be written without having to write a complex algorithm to handle a large 32-bit number. Use the reference manual to select the correct arithmetic instruction. If done correctly, the program will return an accurate result.

Laboratory 3.2.1: Condition Code Register

The program below defines num_1 as a byte containing 0x40 and num_2 as a byte containing 0x50. The program loads accumulator A with num_1 and then adds num_2 to the accumulator.

```
My_Constant:  section
num_1:  dc.b  $40
num_2:  dc.b  $50
My_Code:      section
Entry:  ldaa  num_1
        adda  num_2
        nop
```

Start a new project and enter the code above. Assemble and run your program. Step through the program (until the nop instruction is reached) and show the contents of Accumulator A after program execution.

A = $90

Show the states of the following bits of the Condition Code Register (CCR) after execution.

Carry Flag (C)	Set [ ]	Clear [✓]
Overflow Flag (V)	Set [ ]	Clear [✓]
Negative Flag (N)	Set [ ]	Clear [✓]
Zero Flag (Z)	Set [ ]	Clear [✓]

Now change num_1 to 0xF6 and num_2 to 0xEC and run the program again. The values for num_1 and num_2 will have to be changed in the program and the program will have to be reassembled and reloaded into memory. Run the program and fill in the blanks below.

What are the contents of Accumulator A after program execution? $E2

Show the states of the following bits of the Condition Code Register (CCR) after execution.

Carry Flag (C)	Set [✓]	Clear [ ]
Overflow Flag (V)	Set [✓]	Clear [ ]
Negative Flag (N)	Set [ ]	Clear [✓]
Zero Flag (Z)	Set [ ]	Clear [✓]

Change num_2 to 0x57 and run the program again. Fill in the blanks below.

What are the contents of Accumulator A after program execution? $4D

Show the states of the following bits of the Condition Code Register (CCR) after execution.

Carry Flag (C) Set [✓] Clear [ ]
Overflow Flag (V) Set [✓] Clear [ ]
Negative Flag (N) Set [ ] Clear [✓]
Zero Flag (Z) Set [ ] Clear [✓]

Change num_1 to 0xE9 and num_2 to 0x89 and run the program again. Fill in the blanks below.

What are the contents of Accumulator A after program execution? $72

Show the states of the following bits of the Condition Code Register (CCR) after execution.

Carry Flag (C) Set [✓] Clear [ ]
Overflow Flag (V) Set [✓] Clear [ ]
Negative Flag (N) Set [ ] Clear [✓]
Zero Flag (Z) Set [ ] Clear [✓]

Change the program so that both num_1 and num_2 are 0x3F and change the ADDA instruction to a SUBA instruction. Reassemble and run the program. Fill in the blanks below.

What are the contents of Accumulator A after program execution? $00

Show the states of the following bits of the Condition Code Register (CCR) after execution.

Carry Flag (C) Set [ ] Clear [✓]
Overflow Flag (V) Set [ ] Clear [✓]
Negative Flag (N) Set [ ] Clear [✓]
Zero Flag (Z) Set [✓] Clear [ ]

Change the above program so that NUM_2 is 0x90. Run the program and fill in the blanks below.

What are the contents of Accumulator A after program execution? $70

Show the states of the following bits of the Condition Code Register (CCR) after execution.

Carry Flag (C) Set [ ] Clear [✓]
Overflow Flag (V) Set [ ] Clear [✓]
Negative Flag (N) Set [✓] Clear [ ]
Zero Flag (Z) Set [ ] Clear [✓]

Laboratory 3.2.2:

Write a short program that loads accumulator A with `0x12` and defines a variable `VAR_1` as a byte of storage initialized with `0x30`. The code to define a byte is given below. The program should then use the `CMPA` instruction to compare the value in accumulator A to `VAR_1`. The `CMPA` instruction compares the two values by "subtracting" `VAR_1` from the value in accumulator A, but does not change either value. Run the program and fill in the blanks below.

What are the contents of accumulator A after program execution? $12

Show the states of the following bits of the Condition Code Register (CCR) after execution.

Carry Flag (C)	Set [ ]	Clear [✓]
Overflow Flag (V)	Set [ ]	Clear [✓]
Negative Flag (N)	Set [ ]	Clear [✓]
Zero Flag (Z)	Set [ ]	Clear [✓]

Laboratory 3.2.3:

Enter, build, and run the following programs and determine the value in accumulator A when the `NOP` instruction is reached.

```
        ldaa   #$D3
        adda   #$F2
        bvs    done
        ldaa   #0
done:   nop
```

What are the contents of Accumulator A after program execution? $C5

```
        ldaa   #$D3
        adda   #$F2
        bcs    done
        ldaa   #0
done:   nop
```

What are the contents of Accumulator A after program execution? $00

```
        ldaa   #$41
        adda   #$5A
        bvs    done
        ldaa   #0
done:   nop
```

What are the contents of Accumulator A after program execution? $00