5.4 Exercises
1. Display bytes from memory locations at 0x4020 to 0x4027.

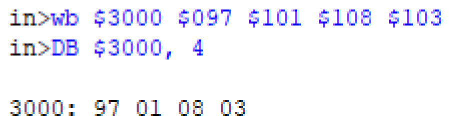
4000: CF 20 00 10 EF CE 00

2. What are the contents (in hex, binary, decimal both signed and unsigned and ASCII) of memory location at 0x4024?

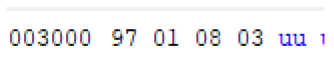
Hex: EF

5.6 Questions Determine the ASCII character codes for the first four (4) letters of your UserID (i.e. the user name you type in response to the ”Login:” prompt on the workstations.) Modify memory locations 0x3000-0x3003 so that they contain the 4 ASCII codes and ensure that the next memory location (0x3004) contains 0x00. Show the sequence of commands required and the resulting memory contents.

aelg
097 101 108 103



```
in>wb $3000 $097 $101 $108 $103
in>DB $3000, 4

3000: 97 01 08 03
```



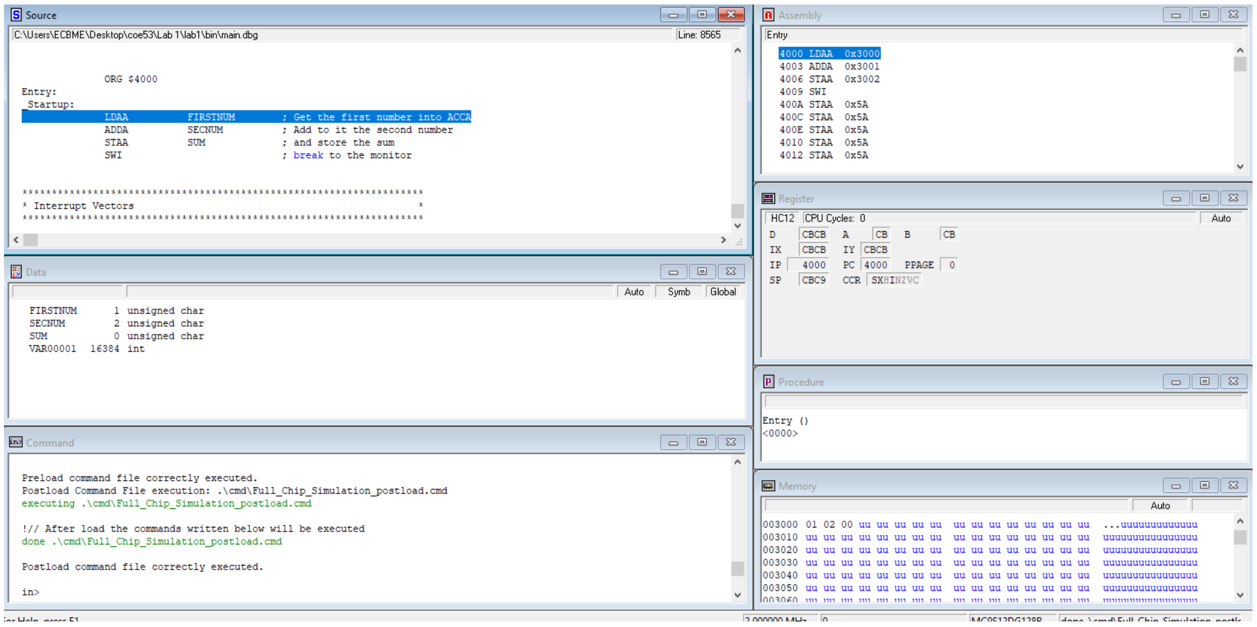
```
003000 97 01 08 03 uu ı
```

7.1 Exercise Manually convert each instruction into its corresponding operation code, and then enter the program into the microcomputer. In detail: • Use the HC12 Reference Manual to look up the ‘operation code’ for the INCA instruction and the SWI instruction. These instructions have a ‘one byte’ op-code, which keeps things simple. Other instructions could require more bytes. • Decide on a suitable starting address (the ‘Origin’) for the program. The RAM location $3000 is a suitable choice. • Use the WB command to enter the sequence of three bytes, starting at the chosen origin. • Use the DB command to verify that the instructions have been stored correctly. Next, you need to set up the machine registers to run the program: • Use the RS command to set the accumulator A (also referred to as ACCA) to 00. • Use the RS command to set the PC to the value of the Origin address • Use the RD command to examine the register contents and ensure that they are correct. Now run the program by executing the Go (G) command. The computer should break to the monitor with the program counter pointing at the next byte that would be executed. The accumulator ACCA should contain the expected result. Check that this is the case. Note that the original program that calculates Fibonacci Numbers still resides in the memory and can be invoked in the same way. Now you can close the True-Time Simulator & Real-Time Debugger window. This ‘Hand Assembly’ process is far too tedious and error-prone to use for entering anything more than a few lines of program code.

and only if (A) was $FF before the operation.

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
INCA	INH	42	O	O

```
in>wb $3000 $42 $43 $44
in>DB $3000, 3

3000: 42 43 44

in>RS A=00

in>RS PC=$3000

in>RD
A=0x0 B=0xCB CCR=0xD0 D=0xCB IX=0xCBCB IY=0xCBCB SP=0xCBC9
PC=0x3000 PPAGE=0x0 IP=0x3000


in>G
STARTED
RUNNING

in>S
STOPPING
HALTED

in>RD
A=0x0 B=0x5 CCR=0xC1 D=0x5 IX=0x12 IY=0x8 SP=0x1FFE
PC=0x4026 PPAGE=0x0 IP=0x4026
```



```
C:\Users\ECBME\Desktop\coe53\Lab 1\lab1\bin\main.dbg    Line: 8565

            ORG $4000
Entry:
 _Startup:
            LDAA        FIRSTNUM        ; Get the first number into ACCA
            ADDA        SECNUM          ; Add to it the second number
            STAA        SUM             ; and store the sum
            SWI                         ; break to the monitor


****************************************************************
* Interrupt Vectors                                            *
****************************************************************
```

```
Entry
  4000 LDAA  0x3000
  4003 ADDA  0x3001
  4006 STAA  0x3002
  4009 SWI
  400A STAA  0x5A
  400C STAA  0x5A
  400E STAA  0x5A
  4010 STAA  0x5A
  4012 STAA  0x5A
```

```
HC12  CPU Cycles: 0
D   CBCB   A   CB   B   CB
IX  CBCB   IY  CBCB
IP  4000   PC  4000   PPAGE  0
SP  CBC9   CCR SXHINZVC
```

```
FIRSTNUM      1 unsigned char
SECNUM        2 unsigned char
SUM           0 unsigned char
VAR00001  16384 int
```

```
Entry ()
<0000>
```

```
Preload command file correctly executed.
Postload Command File execution: .\cmd\Full_Chip_Simulation_postload.cmd
executing .\cmd\Full_Chip_Simulation_postload.cmd

!// After load the commands written below will be executed
done .\cmd\Full_Chip_Simulation_postload.cmd

Postload command file correctly executed.

in>
```

```
003000 01 02 00 uu uu uu uu uu  uu uu uu uu uu uu uu uu  ...uuuuuuuuuuuuu
003010 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
003020 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
003030 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
003040 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
003050 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
003060 uu uu uu uu uu uu uu uu  uu uu uu uu uu uu uu uu  uuuuuuuuuuuuuuuu
```