UNIVERSITY OF WALES, ABERYSWYTH
Department of Computer Science

CS25510 - COMPUTER HARDWARE
LABORATORY WORK

THR 68HC11 Simulator - INTRODUCTION

1. Login to a PC machine connected to the I.S. network and create a directory for your Assembly Language programs within your home directory.
2. From the Computer Science courseware menu, select **THR 68HC11 Simulator**.
3. Once loaded, you are presented with a **Commands** window and a **CPU registers** window. All windows are scalable and moveable.
4. Click on the **?** button for help information on the simulator.
5. **Before** writing your first 68HC11 program, go to the help section: **Overview of the buttons and menu items**, and learn what the individual buttons do.
6. After learning the functions assigned to each of the buttons, go to the help section: **Editor and Assembler**. Read how the **editor** and the **assembler** operate.
7. You should be in a position now to write your first 68HC11 Assembly Language program. Using the **editor**, type in the following program:

```
* This is a simple program that outputs an
* 8-bit binary count on the Port B of the 68HC11

PORTB   EQU    $1004  ; Port B data register

        ORG    $E000  ; start of ROM (program)
        CLR    PORTB  ; clears Port B to $00
MAIN    INC    PORTB  ; Adds 1 to Port B
        BRA    MAIN   ; creates an infinite loop

* Note:
* Even though Port B is an output, the ‘inc PORTB’ reads
* Port B value, increments it, then writes the value back.
* This is a common behaviour for output registers for
* the 68HC11 and is a very useful feature.
```

8. Assemble this program and correct for any typographical errors you may have introduced.
9. Open the **Port registers** window (obtained via the **View** menu), and change PORTB from hexadecimal to binary (right mouse click on PORTB).
10. Using the **step** button, run your program in single step mode and observe how PORTB changes. Also notice any changes that occur in the **CPU registers** window.
11. Click on the **run** button and observe the simulator at maximum speed. To stop the program execution, click on the **stop** button.
12. **Reset** the 68HC11 simulator, familiarise yourself with running and stopping the program, and then **save** your program in the directory that you created at the beginning of the session.
13. Using the **editor**, modify your program to the following:

```
* 8-bit binary count on the Port B of the 68HC11
* with a delay loop.

PORTB  EQU    $1004  ; Port B data register

       ORG    $E000  ; start of ROM (program)
       CLR    PORTB  ; clears Port B to $00
MAIN   INC    PORTB  ; Adds 1 to Port B
       LDX    #$00ff ; set up delay timing
DELAY  DEX           ; X = X - 1
       BNE    DELAY  ; if X != 0, then loop
       BRA    MAIN   ; creates an infinite loop
```

14. Assemble to program, click on the **run** button and observe the speed with which PORTB changes. To stop the program execution, click on the **stop** button.
15. **Reset** the 68HC11 simulator, familiarise yourself with running and stopping the program.
16. Experiment with different delay values. Why is the # symbol used before the hexadecimal delay value?
17. From the **View** menu, click on the **TH Rijswijk IO Box**. Use the simulator help facility to familiarise yourself with how this box can be used to simulate an external electronic circuit or process.
18. Re-run your program using the **TH Rijswijk IO Box** and observe how the PORTB LED's (light emitting diodes) change.
19. **Save** your program in the directory that you created at the beginning of the session.
20. Using the **editor**, create the following program:

```
* This is a program that adds the contents of address
* $0000 to the contents of address $0001 and stores the resulting
* sum in the memory location at address $0002. If there is an
* overflow the contents of address $0003 is set to $FF else
* address $0003 is cleared.

         ORG    $0000  ; start of RAM (data)
OPER1    RMB    1      ; reserve 1 memory byte for operand 1
OPER2    RMB    1      ; reserve 1 memory byte for operand 2
SUM      RMB    1      ; reserve 1 memory byte for sum
OVERFL   RMB    1      ; reserve 1 memory byte for overflow signal

FALSE    EQU    $00    ; equate constant FALSE to $00

         ORG    $E000  ; start of ROM (program)
START    LDAB   #FALSE ; be optimistic (assume no overflow occurs)
         LDAA   OPER1  ; load operand 1 in accumulator A
         ADDA   OPER2  ; add operand 2 to accumulator A
         BVC    NOV    ; branch if no overflow occurs
         COMB          ; oops overflow: invert accumulator B
NOV      STAA   SUM    ; store result of addition
         STAB   OVERFL ; store overflow signal
         STOP          ; nice way to stop without running wild

         ORG    $FFFE  ; reset vector
         FDB    START  ; set to start of program
```

21. **Assemble** the program. If there are no errors then click on **Memory List...** in the **View** menu. You will be asked for a **Starting address for memory list**, just click **OK**.
22. In the **Memory List** window you will see OPER1, OPER2, SUM, OVERFL and FALSE. Double click on OPER1 and OPER2 and enter **7d** and **02** respectively.
23. Run the program and observe the contents of address SUM and OVERFL
24. Reset the simulator and re-run using values **7d** and **03** for OPER1 and OPER2 respectively.
25. Observe the contents of address SUM and OVERFL. **Reset** the simulator and re-run in single **step** mode. Look out for any changes in the **Condition Code register**.
26. How are **V** and **C** related in the Condition Code register?