EE2 Computer Architecture
Laboratory Exercise 1

Getting started with

ARM Software Design Toolkit (SDT) Version 2.2

Objectives

- To introduce the Window-based ARM simulator environment.
- To introduce ARM programming in assembly language

Where to start

- ARM SDT is an easy to use window-based environment for writing and debugging software for the ARM processor. This program is installed on all the machines on all undergraduate teaching laboratories on Levels 1 and 3.

- ARM SDT includes an assembler - a program that translate assembly language into machine instructions, a C-compile, and a symbolic debugger/simulator. The assembly language source is assumed to be in a file `filename.s`. You must first create the source program either using the built-in editor or the excellent free Programmer's File Editor (`pfe32.exe`) which is downloadable from my course home page: (http://www.ee.ic.ac.uk/pcheung/teaching/ee2_computing).

Exercise 1 - "Hello world!"

- Invoke ARM Project Manager program by clicking on the ICON  .

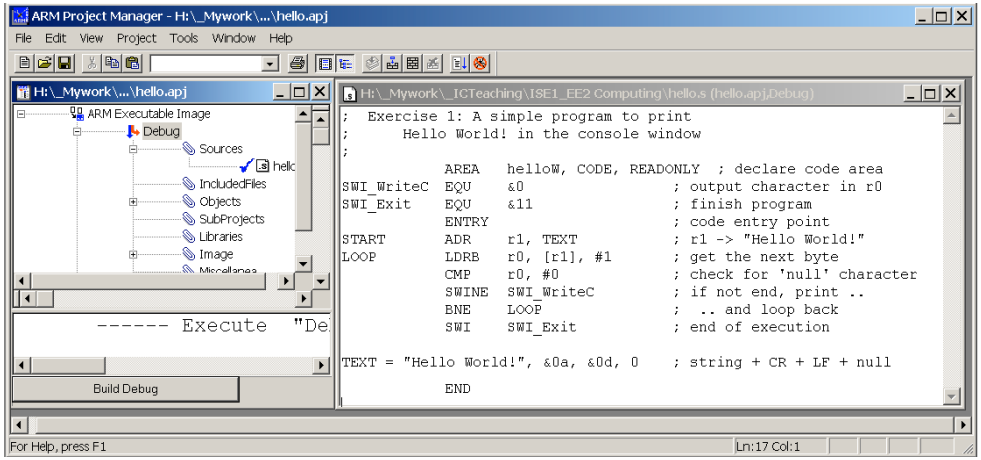
- Use the pulldown menu `File` → `New`, create a new assembler program. You will see an Editor window. Type in the following assembly language program save it as `hello.s`. This is a simple program that produces the message "Hello world!" in the output window.

```
;  Exercise 1: A simple program to print
;       Hello World! in the console window
;
            AREA    helloW, CODE, READONLY  ; declare code area
SWI_WriteC  EQU     &0                  ; output character in r0
SWI_Exit    EQU     &11                 ; finish program
            ENTRY                       ; code entry point
START       ADR     r1, TEXT            ; r1 -> "Hello World!"
LOOP        LDRB    r0, [r1], #1        ; get the next byte
            CMP     r0, #0              ; check for 'null' character
            SWINE   SWI_WriteC          ; if not end, print ..
            BNE     LOOP                ;  .. and loop back
            SWI     SWI_Exit            ; end of execution

TEXT = "Hello World!", &0a, &0d, 0    ; string + CR + LF + null

            END
```

- Next create a project `hello.apj` using the `Project`→`New` command. Add the file `hello.s` to the project. This automatically creates the necessary command line instructions to assemble and link the file `hello.s` to form the executable file `hello`.

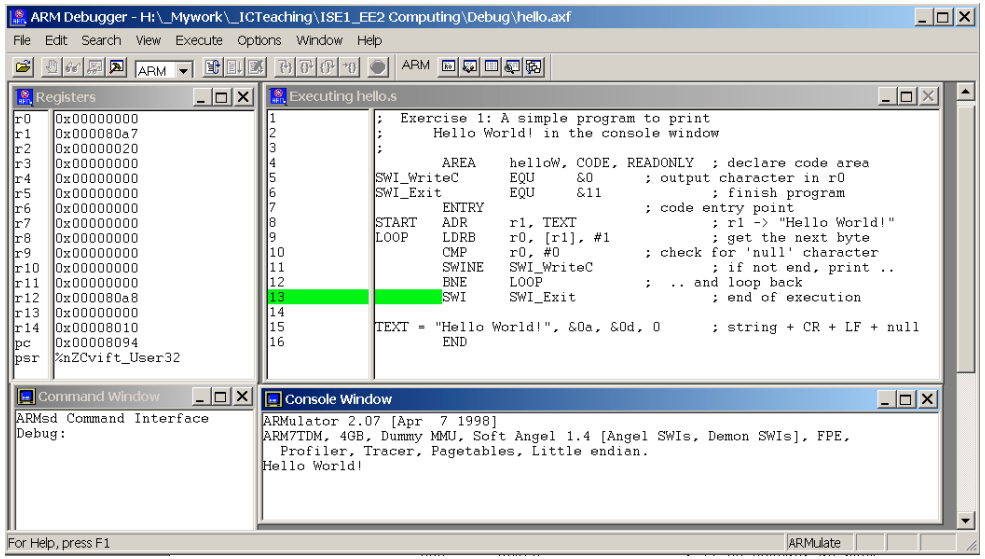
- You should see the following screen dump:

-



- To debug the program, use the command `Project -> Debug` `hello.apj` command to invoke the debugger/simulator. You should see a window as shown below.

- Execute and test the program using single stepping and notice how the register window values change with each instruction.



Exercise 2 - Reporting Time

- Now try this second example program. Make sure that you understand what you are doing. In particular, single step through the `HexOut` subroutine to make sure that you understand every single instruction.

```
            AREA    Example, CODE, READONLY
SWI_WriteC  EQU     0
SWI_ReadC   EQU     4
SWI_Clock   EQU     0x61                ; report elapse time in cent-seconds
            ENTRY                               ; mark first instruction
                                                ; to execute
start       NOP
            SWI     SWI_Clock           ; read timer
            CMP     r1, r0
            BEQ     start               ; if no change, go back
            MOV     r1, r0
            BL      HexOut              ; … else output it as hex
            MOV     r0, #&0a            ; output CR
            SWI     SWI_WriteC
            MOV     r0, #&0d            ; output LF
            SWI     SWI_WriteC
            B       start

; Subroutine HexOut - Output 32-bit word as 8 hex digits as ASCII characters
; Input parameters:  r1 contains the 32-bit word to output
; Return parameters: none
; Registers changed: none
;
HexOut    STMED     r13!, {r0-r2, r14}  ; save working registers on stack
          MOV       r2, #8              ; r2 has nibble (4-bit digit) count = 8
Loop      MOV       r0, r1, LSR #28     ; get top nibble
          CMP       r0, #9              ; if nibble <= 9, then
          ADDLE     r0, r0, #"0"        ;       convert to ASCII numeric char
          ADDGT     r0, r0, #"A"-10     ; else  convert to ASCII alphabet char
          SWI       SWI_WriteC          ; print character
          MOV       r1, r1, LSL #4      ;  shift  left  4  bits  to  get  to  next
nibble
          SUBS      r2, r2, #1          ; decrement nibble count
          BNE       Loop                ; if more, do next nibble
          LDMED     r13!, {r0-r2, pc}   ; retrieve working registers from stack
                                        ;   … and return to calling program
          END
```

Exercise 3 - Subroutine StrLen

The subroutines in Exercises 3 & 4 are useful for future use.

Write and test a subroutine to count the number of characters in a null-terminated string. The subroutine interface is:

```
; Subroutine StrLen - Return the length of a null-terminated string
; Input parameters:  r1 contains the address of the string
; Return parameters: r0 contains the length of string including null character
; Registers changed: r0
```


Exercise 4 - Subroutine StrOut

Write and test a subroutine to output a null terminated string in the console window. The subroutine interface is:

```
; Subroutine StrOut - Output a null-terminated string to console window
; Input parameters:  r1 contains the address of the string
; Return parameters: none
; Registers changed: none
```

Related documents

- Reference CARD for ARM assembly language
- ARM System Call Summary
- Notes for Lectures 3-6

These can be downloaded from the course web page:

http://www.ee.ic.ac.uk/pcheung/teaching/ee2_computing