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, READ	ONLY ; declare code area		
SWI_WriteC	EQU	0.3	; output character in r0		
SWI_Exit	EQU	&11	; finish program		
	ENTRY		; code entry point		
START	ADR	rl, 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!", &Oa, &Od, O ; 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:

٠

ISE1/EE2 Computer System - Lab 1

🔛 ARM Project Manager - H:_Mywork\\hello.apj						
File Edit View Project Tools Window Help						
H: Mywork \hello.apj	<pre>H. Mywork\ Exercise Hello SWI_WriteC SWI_Brit STRT LOOP TEXT = "Hell</pre>	ICTead 1: A s World AREA EQU EQU EQU EQU EQU EQU ENTRY ADR LDRB CMP SWINE BNE SWI 0 Worl END	<pre>ing\ISE1_EE2_Comput imple program to ! in the console helloW, CODE, RH &0 &11 r1, TEXT r0, [r1], #1 r0, #0 SWI_WriteC LOOP SWI_EXit d!", &0a, &0d, 0</pre>	<pre>ing/helos(heloapjDebug) print window SADONLY ; declare code area ; output character in r0 ; finish program ; code entry point ; r1 -> "Hello World!" ; get the next byte ; check for 'null' character ; if not end, print ; and loop back ; end of execution ; string + CR + LF + null </pre>		
For Help, press F1 In:17 Col:1						

- 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.

🔝 ARM Debugger - H:_Mywork_ICTe	eaching\ISE1_EE2 Computing\Debug\hello.axf					
File Edit Search View Execute Options Window Help						
Registers Image: Constraint of the second seco	Image: Securing hellos 1 ; Exercise 1: A simple program to print 2 ; Hello World! in the console window 3 ; 4 AREA 5 SWI_WriteC EQU & 0; output character in r0 6 SWI_Exit 7 ; code entry point 8 START ADR r1, TEXT ; r1 -> "Hello World!" 9 LOOP LDRB r0, [r1], #1 ; get the next byte 10 CMP r0, #0 ; check for 'null' character 11 BNE LOOP ; and loop back 13 SWI_SWI_Exit ; end of execution 14 TEXT = "Hello World!", &Oa, &Od, 0 ; string + CR + LF + null END 16 END					
or Help, press F1 ARMulate ARMulate						

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, RE	ADONLY
SWI WriteC	EQU	0	
SWI ReadC	EQU	4	
SWI_Clock	EQU ENTRY	0x61	; report elapse time in cent-seconds ; mark first instruction ; to execute
start	NOP		
	SWT	SWI Clock	: read timer
	CMP	r1 r0	,
	BEO	start	· if no change, go back
	MOV	r1 r0	, 11 no onange, go baon
	BL	HexOut	else output it as her
	MOV	r0 #60a	, merse oueput it us nex
	SWT	SWI WriteC	, output on
	MOV	r0#e0d	: Output LE
	SWT	SWI WriteC	, output h
	B	start	
	2	beare	
; Subroutin ; Input pan ; Return pan ; Registers	he HexOut rameters: arameters: s changed:	- Output 32-bit word rl contains the 32-b none none	as 8 hex digits as ASCII characters it word to output
HerOut	STMED	r_{131} { r_{0} - r_{2} r_{14} }	· save working registers on stack
nexoue	MOV	r2 #8	; r2 has nibble (4-bit digit) count = 8
Loop	MOV	r0, r1, LSR #28	; get top nibble
100P	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
	SWT	SWI WriteC	; print character
	MOV	r1. r1. LSL #4	; shift left 4 bits to get to next
nibble		11, 11, 101 11	, 51110 1010 1 5105 00 900 00 1010
	SUBS	r2, r2, #1	; decrement nibble count
	BNE	LOOD	; 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

ISE1/EE2 Computer System - Lab 1

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