# Laboratory Exercise 2

# Using Logic Instructions with the ARM Processor

Logic instructions are needed in many embedded applications. Logic instructions are useful for manipulation of bit strings and for dealing with data at the bit level where only a few bits may be of special interest. They are essential in dealing with input/output tasks. In this exercise we will consider some typical uses. We will use the Nios II Soft processor in the DE0-Nano or DE0-Nano-SoC Computer.

#### Part I

In this part you will implement a Nios II assembly language program that counts the longest string of 1's in a word of data. For example, if the word of data is 0x103fe00f, then the required result is 9.

Perform the following:

- 1. Create a new folder to hold your Monitor Program project for this part. Create a file called *part1.s*, and type the assembly language code shown in Figure 1 into this file. This code uses an algorithm involving shift and AND operations to find the required result—make sure that you understand how this works.
- 2. Make a new Monitor Program project in the folder where you stored the *part1.s* file. Use the computer that corresponds to your board, either the DE0-Nano Computer or DE0-Nano-SoC Computer for this project.
- 3. Compile and load the program. Fix any errors that you encounter (if you mistyped some of the code). Once the program is loaded into memory in the computer, single step through the code to see how the program works.

#### Part II

Perform the following.

- 1. Make a new folder and make a copy of the file *part1.s* in that new folder. Give the new file a name such as *part2.s*.
- 2. In the new file *part2.s*, take the code which calculates the number of consecutive 1's and make it into a subroutine called ONES. Have the subroutine use register r3 to receive the input data and register r2 for returning the result.
- 3. Add more words in memory starting from the label TEST\_NUM. You can add as many words as you like, but include at least 10 words. To terminate the list include the word 0 at the end—check for this 0 entry in your main program to determine when all of the items in the list have been processed.

- 4. In your main program, call the newly-created subroutine in a loop for every word of data that you placed in memory. Keep track of the longest string of 1's in any of the words, and have this result in register r15 when your program completes execution.
- 5. Make sure to use breakpoints or single-stepping in the Monitor Program to observe what happens each time the ONES subroutine is called.

/* Program that	at counts consecutive 1's */	
	.text	
	.global _start	
_start:		
	movia r10, TEST_NUM	#load the data word into r10
	ldw r10, 0(r10)	
	movi r15, 0	#r15 will hold the result
LOOP:	beq r10, r0, END	#loop until the data contains no more 1's
	srli r11, r10, 1	#perform SHIFT LEFT , followed by AND
	and r10, r10, r11	
	addi r15, r15, 1	#count length so far
	br loop	
END:	br end	
TEST_NUM:	.word 0x103fe00f	
	.end	

Figure 1: Assembly-language program that finds the largest string of 1's.

## Part III

One might be interested in the longest string of 0's, or even the longest string of alternating 1's and 0's. For example, the binary number 101101010001 has a string of 6 alternating 1's and 0's.

Write a new assembly language program that determines the following:

- Longest string of 1's in a word of data—put the result into register R5
- Longest string of 0's in a word of data—put the result into register R6
- Longest string of alternating 1's and 0's in a word of data—put the result into register R7 (Hint: What happens when an n-bit number is XORed with an n-bit string of alternating 0's and 1's?)

Make each calculation in a separate subroutine called ONES, ZEROS, and ALTERNATE. Call each of these subroutines in the loop that you wrote in Part III, and keep track of the largest result for each calculation, from your list of data.

## Part IV

In this part you are to extend your code from Part III so that the results produced are shown on the Terminal window of the Aardvark Monitor Program. Each result should be displayed as a two-digit decimal number preceded by the name of the register the number corresponds to. You may want to use the approach discussed in Part IV of Exercise 1 to convert the numbers in registers R5, R6, and R7 from binary to decimal.

The DE0-Nano and DE0-Nano SoC Computers can communicate with the Aardvark Monitor Program's Terminal through the JTAG UART. The programming interface of the JTAG UART consists of two 32-bit registers, as shown in Figure 2. On the DE0-Nano-SoC, the register mapped to address 0xFF201000 is called the *Data* register and the register mapped to address 0xFF201004 is called the *Control* register. On the DE0-Nano, these registers are mapped to 0x10001000 and 0x10001004 respectively. The following code examples will use the DE0-Nano-SoC as an example. If you are using a DE0-Nano, simply change the base address from 0xFF201000 to 0x10001000.

The JTAG UART includes a 64-character FIFO that stores data waiting to be transmitted to the host computer. ASCII character data is loaded into this FIFO by performing a write to bits 7-0 of the Data register in Figure 2. Note that writing into this register has no effect on received data. The amount of space, WSPACE, currently available in the transmit FIFO is provided in bits 31-16 of the Control register. If the transmit FIFO is full, then any characters written to the Data register will be lost. A subroutine that transmits a character to the host computer is shown in Figure 3.

Address	31 · · · 16	15	14 · · · 11 1	098	$7 \cdots 1 0$	
0xFF201000	RAVAIL	RVALID	Unus	ed	DATA	Data register
0xFF201004	WSPACE	Unu	ised A	C WI RI	WE RE	Control register

Figure 2: The JTAG UART Programming Interface for the DEO-Nano-SoC.

An example of code that shows the contents of registers on the Terminal window is illustrated in Figure 4. The code in the figure shows only the steps needed to display the contents of register r15 on the Terminal. Extend the code to display the contents of all three registers.

/**********	**************	*******
* Subroutine to	o send a character to the JTAG UART	
* $r3 = characteristics rate rate rate rate rate rate rate rate$	er to send	
******	*************	*************************************/
PUT_JTAG:	movia r10, 0xFF201000	#DE0-Nano-SoC JTAG UART base address
	stw r3, 0(r10)	#send the character
END_PUT:	ret	

Figure 3: A subroutine that sends an ASCII character to the Monitor Program Terminal.

code for Part III (	not shown)	
/* Display r15, r1	6 and r17 on the Terminal */	
DISPLAY:	movia r3. PRINT_R15	
	call PRINT STR	#print the string
		in prime and burning
SHOW R5.	mov r 10 r 15	#display content of r15 on the Terminal
5110 10 103.	call PRINT REG	#print the decimal digits
		"print the deciniar digits
	code for printing "r16." and deci	mal value of r16 (not shown)
	code for printing 110. and deel	inal value of 110 (not shown)
	code for printing "r17." and deci	mal value of $r17$ (not shown)
	code for printing 117. and deel	
/* Subroutine to s	end a null-terminated string to the	TAGUART
* input: r2 oc	ontains the address of the string */	
DDINT CTD.	marine address of the string 7	#DEO Nama Sac ITAC UADT hasa address
PRINT_STR:		#DEU-INANO-SOC JIAG UARI base address
JIAG_LOOP:	$100 r^2, 0(r^3)$	#get the next character
END DDINT		
END_PRINT:	ret	
1x Submarting to ?	'maint'' the desired value of a resi	stor to the ITAC Terminal
/* Subroutine to	print the decimal value of a regi	ster to the JTAG Terminal
* input: r3 is	the register to be converted to dec	cimal and displayed */
PRINT_REG:	mov r19, ra	#save return address
	call DIVIDE	#ones digit will be in r2; tens digit in r3
	movia r6, 0xFF201000	#DE0-Nano-SoC JTAG UART base address
	mov ra, r19	#restore return address
PRINT_R15:	. <b>asciz</b> "r15: "	
	<b>.skip</b> 2 // pad with 2	bytes to maintain word alignment
PRINT_R16:	. <b>asciz</b> "r16: "	
	.skip 2 // pad with 2	bytes to maintain word alignment
PRINT_R17:	. <b>asciz</b> "r17: "	
	.skip 2 // pad with 2	bytes to maintain word alignment

Figure 4: A code fragment for showing registers in decimal on the Terminal window.

For your Monitor Program project for this part, in the screen shown in Figure 5, make sure to select JTAG\_UART as the *Terminal device*. Without this setting no character output will appear on the Terminal window when your code writes to the JTAG UART.

Specify system parameters Host connection: DE-SoC [USB-1] Processor: Nios2 Don't reset the processor when loading a program (ARM only) Terminal device: JTAG_UART	Project Settings W	lizard 📃
System parameters Host connection: DE-SoC [USB-1] Processor: Nios2 Don't reset the processor when loading a program (ARM only) Terminal device: JTAG_UART	Specify syste	em parameters
Host connection: DE-SoC [USB-1]   Processor: Nios2 Don't reset the processor when loading a program (ARM only) Terminal device: JTAG_UART	System parameter	15
Processor: Nios2  Don't reset the processor when loading a program (ARM only) Terminal device: JTAG_UART	Host connection:	DE-SoC [USB-1]   Refresh
Terminal device:  Tage Tage Tage Tage Tage Tage Tage Tag	Processor:	Nios2 🗸
Terminal device: JTAG_UART		Don't reset the processor when loading a program (ARM only)
	Terminal device:	JTAG_UART
Z Back   Nevt N   Finish   Can		Rack Next Sinish Cancel

Figure 5: Specifying the Terminal device.

Copyright © 1991-2016 Intel Corporation. All rights reserved. Intel, The Programmable Solutions Company, the stylized Intel logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Intel Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Intel products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights. Intel warrants performance of its semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel Corporation. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

This document is being provided on an "as-is" basis and as an accommodation and therefore all warranties, representations or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed.