

Laboratory Exercise 3

Using Logic Instructions

In this exercise we will examine the use of logic instructions in the Nios II instruction set. 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, as will be illustrated in Laboratory Exercise 4.

The background knowledge needed to do this exercise can be acquired from the tutorial *Introduction to the Altera Nios II Soft Processor*, which can be found in the University Program section of the Altera web site.

We will use the DE-series Basic Computer with your application programs being loaded in the SDRAM memory.

Part I

Consider the Nios II assembly-language program in Figure 1. It examines a word of data and determines the maximum number of consecutive 1s in that word. For example, the word 0x937a (1001001101111010) has a maximum of four consecutive 1s. The code in the figure calculates the number of consecutive 1s for the data 0x90abcdef. This code is available in the file *lab3_part1.s*. Note that this code is based on a clever (not obvious) idea that we can count the number of consecutive 1s by shifting the number and ANDing it with the shifted result until the number consists of all 0s. Make sure that you understand how this idea works.

```
.include "nios_macros.s"

.equ TEST_NUM, 0x90abcdef /* The number to be tested */



.global _start
_start:
    movia r8, TEST_NUM    /* Load r8 with the number to be tested */
    mov r9, r8            /* Copy the number to r9 */

STRING_COUNTER:
    mov r10, r0           /* Clear the counter to zero */
STRING_COUNTER_LOOP:
    beq r9, r0, END_STRING_COUNTER /* Loop until r9 contains no more 1s */
    srli r11, r9, 0x01     /* Calculate the number of 1s by shifting the number */
    and r9, r9, r11        /* and ANDing it with the shifted result. */
    addi r10, r10, 0x01    /* Increment the counter. */
    br STRING_COUNTER_LOOP
END_STRING_COUNTER:
    mov r12, r10           /* Store the result into r12 */

END:
    br END                /* Wait here when the program has completed */
.end
```

Figure 1. Assembly-language program that counts consecutive 1s.

Perform the following:

1. Create a directory *lab3* and a subdirectory *lab3_part1*. Copy the file *lab3_part1.s* into this directory.
2. Using the Altera Monitor Program, create a new project, *part1*, in the directory *lab3_part1*. Specify that the DE-series Basic Computer and the assembly language program in the file *lab3_part1.s* should be used, using the procedure introduced in Lab 1.
3. Compile and load the program.
4. The Monitor Program will display the disassembled view of the code loaded in the memory. Note that the pseudoinstruction **movia** in the original program has been replaced with a pair of machine instructions, **orhi** and **ori**. This is different from the exercise in Part 2 of Lab 1 where the **movia** pseudoinstruction was replaced by the pair **orhi** and **addi**. Using the **ori** instruction instead of **addi** is a more intuitive choice. We forced this choice by including the macros in the “nios_macros.s” file.
5. Run the program. When the program is running, you will not be able to see any changes (such as the contents of registers or memory locations) in the monitor windows, because the monitor program cannot communicate with the processor system on the DE-series board. But, if you stop the program the present state of these components will be displayed. Do so and observe that the program has stopped executing at the last Branch instruction which is loaded in the memory location 0x28. Also note that the number of consecutive 1s in our test number is 4 as indicated by the contents of register *r12*.
6. Return to the beginning of the program by clicking on the icon . Now, single step through the program by clicking on the icon . Watch how the instructions change the data in the processor's registers.
7. Set the Program Counter to 0x08, which will bypass the first two instructions which load the test number into register *r8*. Also, set the value in register *r8* to 0xabcdef90. How many consecutive 1s are there in this number? Run the program to see if you are right.

Part II

In this part, we will examine one more time how instructions are formed.

Perform the following:

1. Use the *Nios II Processor Reference Handbook*, which is available on Altera's website, to determine the machine-code representation of the following assembly-language instructions: **and r13, r8, r12** and **sra r8, r8, r13**.
2. Reload your program (by selecting **Actions > Load**). Then, execute the program once, stopping at the end.
3. Use the Altera Monitor Program's memory-fill functionality to place these two instructions into memory locations 0x0 and 0x4. We should note that you will see these values updated in the Memory view of the Monitor Program, but you will not see them updated in the Disassembly view.
4. Set the Program Counter to 0x0. What will happen this time? To verify your answer, single step the instructions you placed at addresses 0x0 and 0x4 (to see their effect) and then execute the rest of the program.
5. Using the memory-fill feature change the last Branch instruction to point to the start of the program instead of to itself. Put a breakpoint at this instruction. Run the program and observe the state of registers when the breakpoint is reached. Keep rerunning the program until the number of 1s in the data being tested remains constant. What is the final state?
6. Now repeat steps 1 to 5, but use the instruction **srl r8, r8, r13** instead of **sra r8, r8, r13**. That is, reload your program and then place the machine-code representation of the assembly-language instructions **and r13, r8, r12** and **srl r8, r8, r13** into memory locations 0x0 and 0x4. What is the difference in behavior? Explain the results you observed.

Part III

The program in Figure 1 uses a clever idea of ANDing the test data with its shifted version to determine the number of consecutive 1s. Develop another program that accomplishes the same task but which is not based on this idea. Run and test your program.

Part IV

One might be interested in the longest string of alternating 1s and 0s. For example, the binary number 101101010001 has a string of 6 alternating 1s and 0s, as highlighted here: 101**10101**0001. Assume that the two end bits can be part of the longest string. For example, 1010 has 4 consecutive bits of alternating 1s and 0s.

Write a program that determines the following in a 32-bit word:

- Longest string of 1s - write the result to register *r12*
- Longest string of 0s - write the result to register *r14*
- Longest string of alternating 1s and 0s - write the result to register *r16* (Hint: What happens when an n-bit number is XORed with an n-bit string of alternating 0s and 1s?)

Run and test your program.

Part V

In this part we want to convert an 8-digit binary coded decimal (BCD) number, which is stored as ASCII-encoded characters, into a packed form where each digit occupies four bits. ASCII code for decimal digits represents each digit as a byte in which the upper four bits are set to 0x3 and the lower four bits are the value of the digit. Assume that the 8-digit ASCII-encoded number is stored in 8 bytes at addresses *BCD_ASCII* to *ASCII_BCD* + 7 in the order from the most-significant to the least-significant digit (as they would be stored if obtained from a keyboard input device). The derived 32-bit packed BCD number is to be stored at address *BCD_PACKED*.

Write a program that accomplishes the desired conversion. Run your program and test it with some sample data.

Preparation

As your preparation do the following:

1. Determine the machine code representation for the three instructions in Part II
2. Write the assembly language programs for Parts III to V