# ECE 385 Experiment 6 Test Programs

For each test program, the point value and starting address (in hex) is given, followed by a list of instructions used in the test program, and a description of what the program does. *To begin each test, input the starting address on the switches and press "Run".*

There are seven tests; the first six form a hierarchical group, worth a total of 6 points, in which each test builds upon the instructions tested in the previous. Therefore, if the second test is demonstrated without demonstrating the first test, the points for both the first and second tests will be awarded; demonstrating the last test first will yield all 6 points. The seventh test is used to verify that the CPU properly acts upon each press of the Run and Continue buttons exactly once. This functionality comprises the final 1 point of the demo.

In the descriptions of the test programs, the term "Checkpoint" is used to refer to a pause instruction with a specified value. "Checkpoint 1" would be a pause instruction that displays x01 on the LEDs (excluding any I/O flags).

Basic I/O Test 1

| | |
|---|---|
| Points: | 1 |
| Program Start: | x0003 |
| Instructions Tested: | ANDi, LDR, STR, BR |

This program uses ANDi to clear R0 (i.e., ANDi R0, R0, 0), which is used as a base register for memory operations (using negative values for the offset). All test programs do this to set up for I/O operations. The program then reads in the data on the switches, writes this value to the hex display, and loops back (using BRnzp as an unconditional jump) to repeat the process indefinitely. When working correctly, the hex displays should appear to always display the value of the switches.

Basic I/O Test 2

| | |
|---|---|
| Points: | 1 (cumulative total: 2.0) |
| Program Start: | x0006 |
| Instructions Tested: | ANDi, LDR, STR, BR, PSE |

The code for this program is identical to the previous test, except that it uses pause instructions to ask for input and report output. The first pause instruction (checkpoint 1) will ask for input on the switches. The second and subsequent pauses will display x02 and both ask for input and report that an output is present on the hex display. When operating correctly, each press of the Continue button will transfer the value from the switches to the hex display, but the hex display will not change until Continue is pressed.

Self-Modifying Code Test

| | |
|---|---|
| Points: | 1 (cumulative total: 3.0) |
| Program Start: | x000B |
| Instructions Tested: | ANDi, ADDi, LDR, STR, BR, JSR, PSE |

This program is based upon the same loop as the last program, but inserts some additional operations. Before the loop begins, JSR 0 is executed. This serves to put the PC address in R7 without actually changing the value of PC. (This usage is common in the remaining tests as well). This PC value is used to load the data for the second pause instruction into a register. The loop then operates normally, except that in each iteration, the data for the pause instruction is incremented and stored back to the proper memory location. The result is that with each iteration of the loop, the pause instruction will display a value on the LEDs one greater than the value in the last iteration.

XOR Test

| | |
|---|---|
| Points: | 1 (cumulative total: 4.0) |
| Program Start: | x0014 |
| Instructions Tested: | AND, ANDi, NOT, LDR, BR, PSE |

This program performs XOR operation. In sLC-3, there is no dedicated XOR instruction, so the XOR is performed by multiple simple instructions (AND and NOT). The program will ask for input values (checkpoints 1 and 2), XOR them, and display the result on the hex display (checkpoint 3). It will then loop back to the top and repeat the process from checkpoint 1.

Multiplication Test (or, “Lab 5 in Software”)

| | |
|---|---|
| Points: | 1 (cumulative total:5.0) |
| Program Start: | x0031 |
| Instructions Tested: | AND, ANDi, ADD, ADDi. NOT, LDR, STR, BR, JSR, PSE |

This program performs multiplication, using a variation on the shift-and-add algorithm (using ADD Rx, Rx, Rx as a left-shift operation). The program will ask for input values (checkpoints 1 and 2), multiply them, and display the result on the hex display (checkpoint 3). It will then loop back to the top and repeat the process from checkpoint 1.

Sort Test

| | |
|---|---|
| Points: | 1 (cumulative total: 6.0) |
| Program Start: | x005A |
| Instructions Tested: | <all> |

This program is organized into four parts. The first part is a menu, containing function calls to the other three parts that are executed based on input. The menu contains a single pause

instruction, checkpoint –1 (displays xFF on the LEDs). Entering x0001 will call the “data entry” function, entering x0002 will call the “sort” function, and entering x0003 will call the “display” function. Any other value will simply cause the menu to loop back to the start without doing anything.

The data entry function allows the user to enter new data into the list to be sorted. The list is fixed at length 16, no more, no less. The function will display the current index (starting at 0) to be written to the list, and ask for data from the switches (checkpoint 1). It will do this 16 times, displaying the current index each time, before returning control to the menu loop.

The sort function will sort the values in the list using the Bubble Sort algorithm. No feedback is given to the user about the completion of the algorithm; it will return (seemingly) immediately to the menu.

The display function will display, in turn, each member of the list (checkpoint 2). Note that since the hex display will be used to display the data, the index of the value is displayed on the LEDs (LED<7:4>), using the self-modifying code technique used earlier to change the pause instruction in each iteration. A properly sorted list will be displayed in ascending order.
To test this program, the data entry function needs not be used, since sample values are pre-loaded into the list along with the program itself. These values, both in their original order and sorted order, are in Table 1. It is recommended that to demo, the display function should be run first to verify that the sample values are present and unsorted, the sort function should be run second, and the display function should be run again to verify that the sort was successful.

| Index | Before Sort | After Sort |
|---|---|---|
| 0 | x"00ef" | x"0001" |
| 1 | x"001b" | x"0003" |
| 2 | x"0001" | x"0007" |
| 3 | x"008c" | x"000d" |
| 4 | x"00db" | x"001b" |
| 5 | x"00fa" | x"001f" |
| 6 | x"0047" | x"0046" |
| 7 | x"0046" | x"0047" |
| 8 | x"001f" | x"004e" |
| 9 | x"000d" | x"006b" |
| A | x"00b8" | x"008c" |
| B | x"0003" | x"00b8" |
| C | x"006b" | x"00db" |
| D | x"004e" | x"00ef" |
| E | x"00f8" | x"00f8" |
| F | x"0007" | x"00fa" |

Table 1: Sample list values before and after sorting

"Act Once" Test

| | |
|---|---|
| Points: | 1 |
| Program Start: | x002A |
| Instructions Tested: | ANDi, ADDi, STR, JSR, JMP, PSE |

This program is a simple loop that counts up from zero, once per iteration, and displays the count on the hex display (checkpoint 1). When operating correctly, the counter will increase by one (no more, no less) with each press of Continue (i.e., the CPU will act once per press.) This test need only be run if the "act once" functionality of the run or continue buttons is called into question in the course of running other tests. At the TA's discretion, if this functionality appears to be present during observation of other tests, especially after the display function of the sort test, the points for this test may be awarded without running the test itself.