David Giacobbi

CPSC 260: Assignment #8 Responses

**Problem 1:**

|  |  |  |
| --- | --- | --- |
| movl | %eax | (%rsp) |
| movw | (%rax) | %dx |
| movb | $0xFF | %bl |
| movb | (%rsp, %rdx, 4) | %dl |
| movq | (%rdx) | %rax |
| movw | %dx | (%rax) |

\*answers are highlighted in yellow

**Problem 2:**

|  |  |  |  |
| --- | --- | --- | --- |
| movb | $0xF | (%ebx) | %ebx is not an address register, can’t be dereferenced |
| movl | %rax | (%rsp) | Instruction suffix and register ID do not match in size |
| movw | (%rax) | 4(%rsp) | Memory references can’t be both the source and destination |
| movb | %al | %sl | %sl does not exist in register library |
| movq | %rax | $0x123 | Immediate/literal values cannot be destination |
| movl | %eax | $rdx | Wrong size for the destination operand |
| movb | %si | 8(%rbp) | Instruction suffix and register ID do not match |

\*answers are highlighted in yellow

**Problem 3 (Screenshots):**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**A screenshot of a computer

Description automatically generated**

**Problem 4 (Screenshots):**

**A screenshot of a computer program

Description automatically generated**

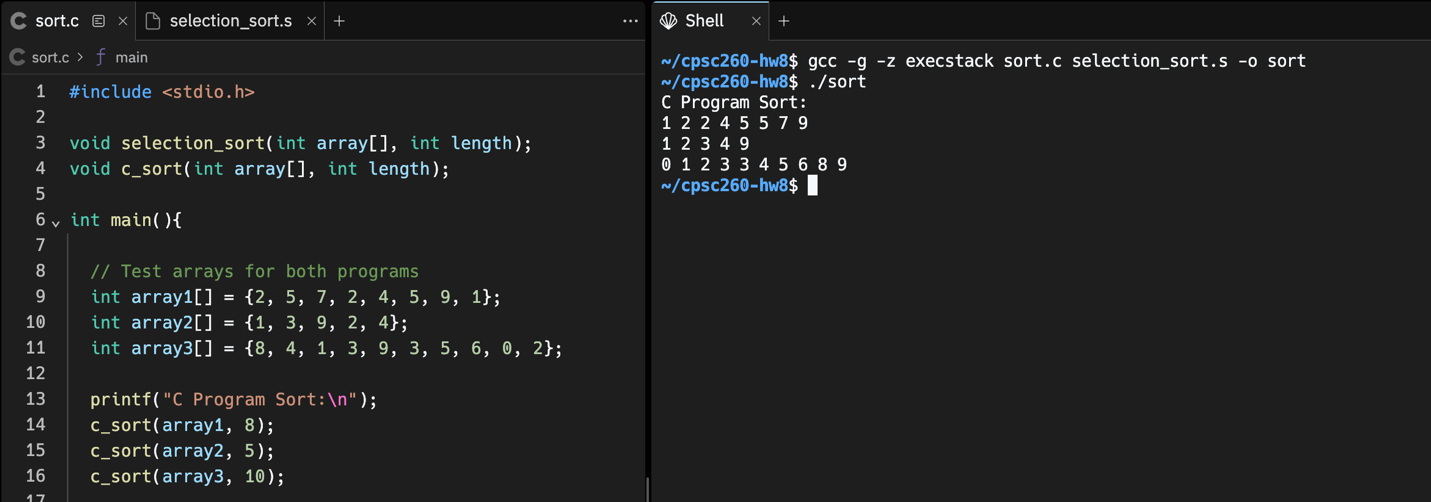
**Problem 5 (Screenshots and Observations):**

Object Dump Screenshots:

**A screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated**

Testing Results Screenshot:

Observations between Object Dump and My Code:

One of the first observations that I made between my own code and the C-compiled assembly is that it decided to push registers onto the stack as they were needed as variables in the function. I think the reasoning behind this is that I had time to think about the structure of my function and what registers I would use ahead of time. Since the compiler is simply reading in lines of my C code, it does not have this luxury and, therefore, pushes available registers onto the stack before altering them.

Another observation was the compiler’s use of the lea function when loading effective addresses. An interesting tactic that I noticed was the use of a negative shift in line 401290. It calls the dereferenced %r9 register with a -4 shift: -0x4(%r9), %r11d. I have never worked with a shift like this and I am curious to know how this could possibly optimize my own function.

Lastly, I noticed the use of a nopl assembly call, which I have not seen before. This would be another call to investigate its purpose as it was often associated with the dereferencing of a register on multiple occasions. Overall, though there are some instances of optimization, the object dump revealed that my own code and its implementation were roughly the same length and, given the restraints of selection sort, roughly follow the same path but maybe use different registers in completing the same task.