

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):

```
switch_case.s
57 to_upper_case:
58     subb    $32, %bl    # make current character uppercase
59     movb    %bl, (%r9)  # store result as pointer
60     retq
61
62 # Lower case function
63 to_lower_case:
64     addb    $32, %bl    # make current character lowercase
65     movb    %bl, (%r9)  # store result as pointer
66     retq
67
68     .data
69 mystr: .ascii "foobar\n"
70     .equ mylen, (. - mystr)
71
```

```
~/cpssc260-hw8$ gcc -c -g switch_case.s
~/cpssc260-hw8$ ld switch_case.o -o switch_case
~/cpssc260-hw8$ ./switch_case
FOOBAR
~/cpssc260-hw8$
```

```
Shell x switch_case.s x + ... Shell x +
switch_case.s
57 to_upper_case:
58 subb $32, %bl # make current character uppercase
59 movb %bl, (%r9) # store result as pointer
60 retq
61
62 # Lower case function
63 to_lower_case:
64 addb $32, %bl # make current character lowercase
65 movb %bl, (%r9) # store result as pointer
66 retq
67
68 .data
69 mystr: .ascii "FOOBAR\n"
70 .equ mylen, (. - mystr)
71

Shell x switch_case.s x + ... Shell x +
~/cpssc260-hw8$ gcc -c -g switch_case.s
~/cpssc260-hw8$ ld switch_case.o -o switch_case
~/cpssc260-hw8$ ./switch_case
foobar
~/cpssc260-hw8$

Shell x switch_case.s x + ... Shell x +
~/cpssc260-hw8$ gcc -c -g switch_case.s
~/cpssc260-hw8$ ld switch_case.o -o switch_case
~/cpssc260-hw8$ ./switch_case
FOO_BAR!
~/cpssc260-hw8$
```

Problem 4 (Screenshots):

```
sort.c x selection_sort.s x + ... Shell x +
sort.c > f main
2
3 void selection_sort(int array[], int length);
4 void c_sort(int array[], int length);
5
6 int main(){
7
8     // Test arrays for both programs
9     int array1[] = {2, 5, 7, 2, 4, 5, 9, 1};
10    int array2[] = {1, 3, 9, 2, 4};
11    int array3[] = {8, 4, 1, 3, 9, 3, 5, 6, 0, 2};
12
13    // ARRAY 1 Assembly Sort
14    selection_sort(array1, 8);
15    for (int i = 0; i < 8; i++){
16        printf("%d ", array1[i]);
17    }
18    printf("\n");
19
20    // ARRAY 2 Assembly Sort
21    selection_sort(array2, 5);
22    for (int i = 0; i < 5; i++){
23        printf("%d ", array2[i]);
24    }
25    printf("\n");
26
27    // ARRAY 3 Assembly Sort
28    selection_sort(array3, 10);
29    for (int i = 0; i < 10; i++){
30        printf("%d ", array3[i]);
31    }
32    printf("\n");
33
34    return 0;
35 }

~/cpssc260-hw8$ gcc -g -z execstack sort.c selection_sort.s -o sort
~/cpssc260-hw8$ ./sort
1 2 2 4 5 5 7 9
1 2 3 4 9
0 1 2 3 3 4 5 6 8 9
~/cpssc260-hw8$
```

Problem 5 (Screenshots and Observations):

Object Dump Screenshots:

```
Shell x +
000000000401270 <c_sort>:
401270: 85 f6          test    %esi,%esi
401272: 0f 8e a0 00 00 00  jle     401310 <c_sort+0xa8>
401278: 41 54          push   %r12
40127a: 45 31 d2       xor     %r10d,%r10d
40127d: 55            push   %rbp
40127e: 48 8d 6f 04    lea     0x4(%rdi),%rbp
401282: 53            push   %rbx
401283: 49 89 e9       mov     %rbp,%r9
401286: 48 89 fb       mov     %rdi,%rbx
401289: 0f 1f 80 00 00 00 00  nopl    0x0(%rax)
401290: 45 8b 59 fc    mov     -0x4(%r9),%r11d
401294: 44 89 d0       mov     %r10d,%eax
401297: 4c 89 ca       mov     %r9,%rdx
40129a: 4d 63 c2       movslq  %r10d,%r8
40129d: 44 89 df       mov     %r11d,%edi
4012a0: 83 c0 01       add     $0x1,%eax
4012a3: 39 c6         cmp     %eax,%esi
4012a5: 7e 16         jle     4012bd <c_sort+0x4d>
4012a7: 8b 0a         mov     (%rdx),%ecx
4012a9: 48 83 c2 04    add     $0x4,%rdx
4012ad: 39 cf         cmp     %ecx,%edi
4012af: 7e ef         jle     4012a0 <c_sort+0x30>
4012b1: 4c 63 c0       movslq  %eax,%r8
4012b4: 83 c0 01       add     $0x1,%eax
4012b7: 89 cf         mov     %ecx,%edi
4012b9: 39 c6         cmp     %eax,%esi
4012bb: 7f ea         jg      4012a7 <c_sort+0x37>
4012bd: 4a 8d 04 83    lea     (%rbx,%r8,4),%rax
4012c1: 49 83 c1 04    add     $0x4,%r9
4012c5: 8b 10         mov     (%rax),%edx
4012c7: 41 89 51 f8    mov     %edx,-0x8(%r9)
4012cb: 44 89 18       mov     %r11d,(%rax)
4012ce: 41 8d 42 01    lea     0x1(%r10),%eax
4012d2: 39 c6         cmp     %eax,%esi
4012d4: 74 0a         je      4012e0 <c_sort+0x70>
4012d6: 41 89 c2       mov     %eax,%r10d
4012d9: eb b5         jmp     401290 <c_sort+0x20>
4012db: 0f 1f 44 00 00  nopl    0x0(%rax,%rax,1)
4012de: 4e 8d 64 95 00  lea     0x0(%rbp,%r10,4),%r12
4012e5: 48 8d 2d 18 0d 00 00  lea     0xd18(%rip),%rbp      # 402004 <_IO_std
in_used+0x4>
4012ec: 0f 1f 40 00    nopl    0x0(%rax)
4012f0: 8b 13         mov     (%rbx),%edx
4012f2: 48 89 ee       mov     %rbp,%rsi
4012f5: bf 01 00 00 00 00  mov     $0x1,%edi
4012fa: 31 c0         xor     %eax,%eax
4012fc: 48 83 c3 04    add     $0x4,%rbx
401300: e8 5b fd ff ff  call    401060 <__printf_chk@plt>
401305: 4c 39 e3       cmp     %r12,%rbx
401308: 75 e6         jne     4012f0 <c_sort+0x80>
40130a: 5b           pop     %rbx
40130b: bf 0a 00 00 00 00  mov     $0xa,%edi
401310: 5d           pop     %rbp
401311: 41 5c         pop     %r12
401313: e9 18 fd ff ff  jmp     401030 <putchar@plt>
401318: bf 0a 00 00 00 00  mov     $0xa,%edi
40131d: e9 0e fd ff ff  jmp     401030 <putchar@plt>
```

Testing Results Screenshot:

```
sort.c x selection_sort.s x +
sort.c > f main
1 #include <stdio.h>
2
3 void selection_sort(int array[], int length);
4 void c_sort(int array[], int length);
5
6 int main(){
7
8     // Test arrays for both programs
9     int array1[] = {2, 5, 7, 2, 4, 5, 9, 1};
10    int array2[] = {1, 3, 9, 2, 4};
11    int array3[] = {8, 4, 1, 3, 9, 3, 5, 6, 0, 2};
12
13    printf("C Program Sort:\n");
14    c_sort(array1, 8);
15    c_sort(array2, 5);
16    c_sort(array3, 10);
17

~/cpssc260-hw8$ gcc -g -z execstack sort.c selection_sort.s -o sort
~/cpssc260-hw8$ ./sort
C Program Sort:
1 2 2 4 5 5 7 9
1 2 3 4 9
0 1 2 3 3 4 5 6 8 9
~/cpssc260-hw8$
```

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.