Solutions

Related Problem 1:

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

Related Problem 2:

- `movb $0xF, (%ebx)` // Cannot use %ebx as address register in x86-64 machine
  Correct: `movb $0xF, (%rbx)`
- `movl %rax, (%rsp)` // Mismatch between instruction suffix and register ID
  Correct: `movq %rax, (%rsp)`
- `movw (%rax),4(%rsp)` // Cannot have both source and destination be memory references
  Correct: `movw (%rax), %dx` followed by `movw %dx, 4(%rsp)`
- `movb %al,%sl` // No register named %sl
  Correct: `movb %al, %sil`
- `movl %eax,$0x123` // Cannot have immediate as destination
  Correct: `movl $0x123, %eax`
- `movl %eax,%dx` // Destination operand incorrect size
  Correct: `movw %ax, %dx` or `movl %eax, %edx`
- `movb %si, 8(%rbp)` // Mismatch between instruction suffix and register ID
  Correct: `movb %sil, 8(%rbp)`

Related Problem 3: *Answer:* https://godbolt.org/z/9G4696P6j

Related Problem 4: *Answer:* https://godbolt.org/z/ePox4Y7sv

Related Problem 5: *Answer:* https://godbolt.org/z/aq9h3G8h8

Related Problem 6:

Operand	Value (not used LEA)	Value (used LEA)
%rax	0x100	Not valid syntax for lea
0x104	0xAB	Not valid syntax for lea
$0x108	0x108	Not valid syntax for lea
(%rax)	0xFF	0x100
4(%rax)	0xAB	0x104
9(%rax,%rdx)	0x11	0x10C
260(%rcx,%rdx)	13	0x108
0xFC(,%rcx,4)	0xFF	0x100
(%rax,%rdx,4)	0x11	0x10C

Related Problem 7: *Answer:* https://godbolt.org/z/rT5cYaY6z

Related Problem 8: *Answer:* https://godbolt.org/z/z5PdPPbez

Related Problem 9:

CF	(`unsigned`) (b + a) < (`unsigned`) a	Unsigned overflow
ZF	((b + a) == 0)	Zero
SF	((b + a) < 0)	Negative
OF	(a < 0 == b < 0) && ((b + a) < 0 $\neq a<$ 0)	Signed overflow

Answer: https://godbolt.org/z/f9f3M49hs

Related Problem 10: *Answer:* https://godbolt.org/z/ndrn6bqWY

Related Problem 11: *Answer:* https://godbolt.org/z/5s5ffGPz5

Related Problem 12: *Answer:* https://godbolt.org/z/TWezeEM6d

Related Problem 13: *Answer:* https://godbolt.org/z/q7c1e45Mb

Related Problem 14:
As shown in Table 6 of the review document, in the bit representation, negative values are already greater than 6 and the maximum representable signed version of the same type. That is why the compiler interprets those bits as unsigned (from Lab 2, we know that bits are always the same; we just interpret them differently) so that it does not need an additional check. For example, `0b1000` is -8 as signed value. If I treat it as unsigned, it becomes 8 which is already greater than Maximum signed 4 bit value which is 7. Instead of having two if conditions, compilers just interprets them as unsigned.

Related Problem 15: *Answer:* https://godbolt.org/z/3GWE8bves

Related Problem 16:

The same reason explained in the **Related Problem 14**. `ja` interprets the values as unsigned to obviate the need for additional `< 0` cases.

Related Problem 17: *Answer:* https://godbolt.org/z/cnaGd5no3

Related Problem 18: *Answer:* https://godbolt.org/z/68b4fEc38