Booleans & 64-Bit Mode

In 64-bit mode, instructions work similarly to how they do in 32-bit mode, but with some differences due to the larger register size.

Operand Size: When you operate on 64-bit registers or memory operands with a source operand that's smaller than 32 bits, all bits in the destination operand are affected eg.

```
316 mov rax, allones     ; RAX = FFFFFFFFFFFFFFFF
317 and rax, 80h         ; RAX = 0000000000000080
```

Here, the and operation affects all 64 bits of RAX.

32-Bit Operand: However, when you use a 32-bit constant or register as the source operand, only the lower 32 bits of the destination operand are modified. For instance:

```
320 mov rax, allones         ; RAX = FFFFFFFFFFFFFFFF
321 and rax, 80808080h       ; RAX = FFFFFFFF80808080
```

In this case, only the lower 32 bits of RAX are changed by the and operation.

Memory Operand: The same rules apply when the destination operand is in memory, not just in registers.

Special Handling for 32-Bit Operands: You need to be careful when dealing with 32-bit operands because they behave differently from other operand sizes in 64-bit mode.

Understanding these distinctions is crucial when writing assembly code for 64-bit systems.

==========

Questions:

==========

Question: How can you clear the high 8 bits of AX without changing the low 8 bits using a single 16-bit operand instruction?

Answer: You can clear the high 8 bits of AX by using the AND instruction with the 16-bit mask 00FFh. The instruction would look like and ax, 00FFh.

```
and ax, 00FFh
```

Question: How can you set the high 8 bits of AX without changing the low 8 bits using a single 16-bit operand instruction?

Answer: You can set the high 8 bits of AX by using the OR instruction with a 16-bit value.

```
or ax, FF00h
```

Question: What instruction can you use to reverse all the bits in EAX with a single instruction?

Answer: To reverse all the bits in EAX, you can use the XOR instruction with a mask where all bits are set to FFFFFFFFh. The instruction would be xor eax, FFFFFFFFh.

```
xor eax, FFFFFFFFh
```

Question: How can you set the Zero flag if the 32-bit value in EAX is even and clear the Zero flag if EAX is odd?

Answer: You can set the Zero flag if the 32-bit value in EAX is even and clear the Zero flag if EAX is odd using the TEST instruction and conditional jumps. Here's an example:

```
test eax, 1       ; Test if the least significant bit is set
jz even_label     ; Jump if Zero flag is set (EAX is even)
; Code for odd case here
even_label:
; Code for even case here
```

Question: How can you convert an uppercase character in AL to lowercase using a single instruction, but without modifying AL if it's already lowercase?

Answer: To convert an uppercase character in AL to lowercase without modifying it if it's already lowercase, you can use conditional instructions like this:

```
cmp al, 'A'       ; Compare AL with 'A'
jl not_uppercase ; Jump if AL is less than 'A' (not uppercase)
cmp al, 'Z'       ; Compare AL with 'Z'
jg not_uppercase ; Jump if AL is greater than 'Z' (not uppercase)
add al, 32        ; Convert uppercase to lowercase ('A'-'a' = 32)
not_uppercase:
; Continue with your code here
```

This code first checks if AL is between 'A' and 'Z' (inclusive) using CMP and conditional jumps (JL and JG). If it's within that range, it adds 32 to AL, converting the uppercase letter to lowercase.