David Giacobbi

CPSC 260-01: Assignment 5 Written Answers

1. The register %rax stores the lower ax byte of $53; in other words, it stores the first 8 bit character with its memory address too. The register %rbx stores the high bh byte of $37, storing the second byte character of $37. Lastly, the %rip register stores the instruction pointer which tracks the current line of assembly code that is being executed. In each movb method, a specified character, according to size, is moved to the provided register.

2. The %rax register is the accumulator for the program. At first, the program stores the $65535 2-byte word into the accumulator. Then, the low byte of the accumulator is changed from ff to 01, resulting in the accumulator value to change from 0xffff to 0xff01.

3. Similar to problem 2, the %rax register acts as the accumulator. It first takes in the low byte character of 0x01 and stores it in the %rax register. However, once the 2-byte word $65535 is added into the accumulator, the larger value overwrites the smaller value completely, changing the %rax register to 0xffff.

4. The %rax register is the accumulator for the program. When it takes in the first 2-byte word $17, the value of the register is changed from 0x0 to 0x17. When it takes in the addw $-17, the %rax register is changed back to zero as -17+17 results in zero. Additionally, the eflags register changes from 0x202 to 0x257, tracking the current state of the processor as well as documenting the results of the addition that was performed. The %rbx register then takes in the 2-byte value $23 into the base register. The %rbx register returns to zero when $23 2-byte is subtracted from the register. This operation is tracked by the eflags register, changing its value from 0x257 to 0x246.

5. The %rax register is filled with the 2-byte word $65535, changing the value from 0x0 to 0xffff. The addw function then adds $2 to 0xffff, causing it to change back to 0x1 in the %rax register. Moreover, the eflags register changes from 0x202 to 0x213 to track the memory overload that happened in that register. Next the value 0x1 from the %rax accumulator register is moved to the %rbx base register, changing its value from 0x0 to

0x1. Lastly, $-2 is added to the %rbx base register, changing its value from 0x1 back to 0xffff. This alerted the eflag to acknowledge that the value is no longer overloaded in that register, changing the eflag from 0x213 to 0x286. The $-2 addition worked because the eflag remembered that the 0xffff value was overloaded in the %rax register; therefore, when the value moved to the %rbx register, the eflag tracked the overload when it was copied to the other register.