Purpose: In this assignment we will be learning instructions to copy/set data and perform basic arithmetic operations.

To begin, download the provided assignment 2 template file. The file contains variables already declared in memory, some example instructions, as well as comments to organize your code.

For the purposes of this assignment, all variables will include their size in their name, but this is not the normal assembly naming convention.

mov instructions (mov, movzx, movsx)

The mov instruction allows us to copy values to registers and memory (variables). The movzx and movsx instructions behave like the mov instruction except that they allow a smaller source to be moved into a larger destination. The movzx instruction is used with unsigned integers and movsx is for signed integers. See the textbook for more details.

1. Write the instruction(s) to set the value of doubleAnswer1 to 400,000.
2. Write the instruction(s) to copy the value of doubleVariable2 to doubleAnswer2. Hint: You'll need to use two mov instructions.
3. Write the instruction(s) to extend an unsigned integer value from byteVariable3 to quadAnswer3.
4. Write the instruction(s) to extend an unsigned integer value from doubleVariable4 to quadAnswer4. Hint: You cannot use the movzx instruction for this.
5. Write the instruction(s) to extend a signed integer value from wordVariable5 to quadAnswer5.

add instruction

The add instruction determines the sum of two operands and stores the value in the first operand. It works with any combination of unsigned and signed integer values, assuming it does not overflow the data size. The adc instruction can be used in those situations. See the textbook for more details.

6. Write the instruction(s) to add wordVariable6a to wordVariable6b and store the result in wordAnswer6.
7. Write the instruction(s) to add the immediate value 2 to doubleAnswer7. Hint: This should take only a single instruction.

sub instruction

The sub instruction determines the difference of two operands and stores the value in the first operand. It works with any combination of unsigned and signed integer values, assuming it does not overflow the data size. See the textbook for more details.

8. Write the instruction(s) to subtract quadVariable8b from quadVariable8a and store the result in quadAnswer8.
9. Write the instruction(s) to subtract wordVariable9 from doubleVariable9 and store the result in doubleAnswer9. Hint: You'll need to extend wordVariable9.

inc and dec instructions

The inc instruction adds 1 to the provided register or memory operand. Dec subtracts 1 from the operand.

mul Instruction

The mul instruction multiplies unsigned integer values. One of the multiplicands must be placed into the appropriately sized portion of the rax register. The answer will always be split into two registers (either al+ah or ax/eax/rax + dx/edx/rdx). See the textbook for more details.

10. Write the instruction(s) to multiply byteVariable10a by byteVariable10b and store the result in wordAnswer10. Hint: This should be doable with just 3 instructions.

11. Write the instruction(s) to multiply doubleVariable11a by doubleVariable11b and store the result in quadAnswer11.

12. Write the instruction(s) to multiply quadVariable12a by quadVariable12b and store the result in doubleQuadAnswer12.

imul instruction

The imul instruction multiplies signed values. The imul instruction allows a variable number of operands and you are free to use any of the variations that you wish. See the textbook for more details.

13. Write the instruction(s) to multiply byteVariable13 by -3 and store the result in wordAnswer13.

div instruction

The div instruction divides the contents of either ax or a combination of ax/eax/rax + dx/edx/rdx depending on the size of the provided operand. See the textbook for more details.

Since div works with unsigned values, the upper register must either be filled or set to all zeroes.

14. Write the instruction(s) to divide byteVariable14a by byteVariable14b using unsigned division. Store the answer in byteAnswer14 and any remainder in byteRemainder14. Hint: The upper portion of ax needs to be set before dividing.

15. Write the instruction(s) to divide doubleVariable15 by wordVariable15 using unsigned division. Store the answer in wordAnswer15. Hint: Make sure each portion of doubleVariable15 ends up in the right register.

16. Write the instruction(s) to divide wordVariable16a by wordVariable16b using unsigned division. Store the remainder in wordRemainder16.

idiv instruction

The signed version of division, idiv, works exactly the same except it checks to see if the value is negative. When extending values for signed division, care must be taken to either extend with all 0's for unsigned or all 1's for signed. See the textbook for more details.

conversion instructions (cbw, cwd, cdq, cqo)

These four instructions are specifically designed to assist in setting up signed division with idiv. Each will take the specified value in the rax register and extend it to the appropriate size with the correct sign in either the ah or rdx register.

17. Write the instruction(s) to divide byteVariable17 by the value 5 using signed division. Store the answer in byteAnswer17.

18. Write the instruction(s) to **multiply** wordVariable18a by the value wordVariable18b using signed multiplication. Then divide the result by wordVariable18c. Store the answer in wordAnswer18. Hint: This set of operations should take exactly 4 instructions.

19. Write the instruction(s) to divide doubleVariable19a by doubleVariable19b using signed division. Store the answer in doubleAnswer19 and the remainder in doubleRemainder19.

20. Write the instruction(s) to divide quadVariable20 by doubleVariable20 using signed division. Store the remainder in doubleRemainder20.

Testing

A debugger script is provided for testing purposes. At any time you can open the debugger, utilize a breakpoint to suspend the program and run the command "source debug.txt" to read in commands to examine each answer and remainder variable. The debug script should only be run once all of your instructions have been executed. Set a debug point before the final output to see the results of your work.

The output from the debug script will be sent to the file answers.txt.

Submission

Once you are satisfied with the program, upload the assembly source code (.asm) file to the assignment #2 page on the class website.