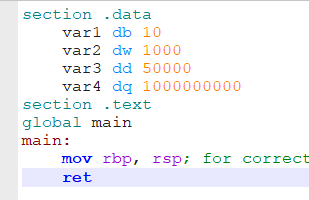
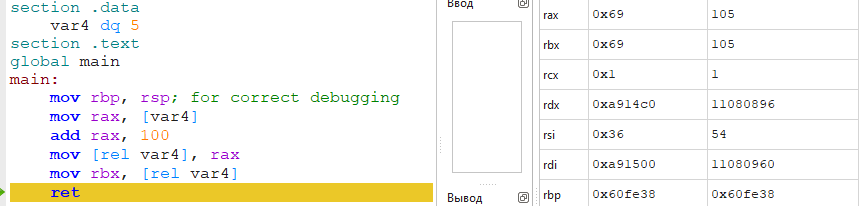
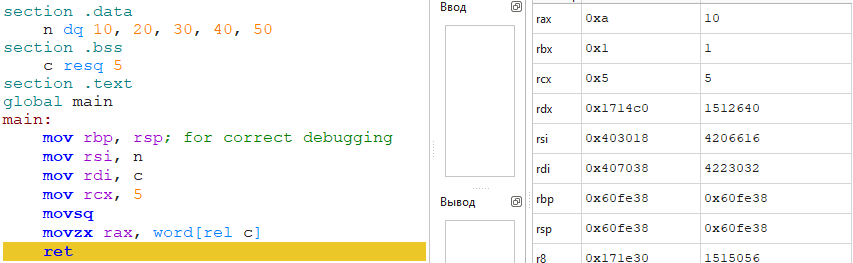
1.



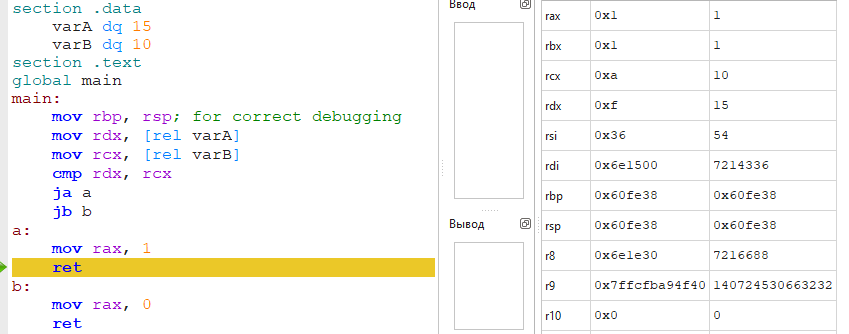
2.

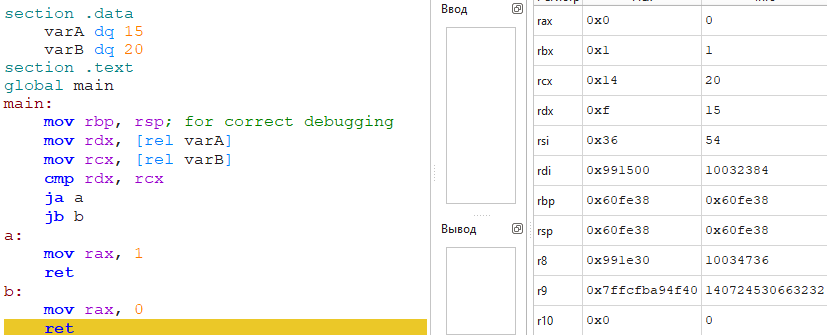


3.

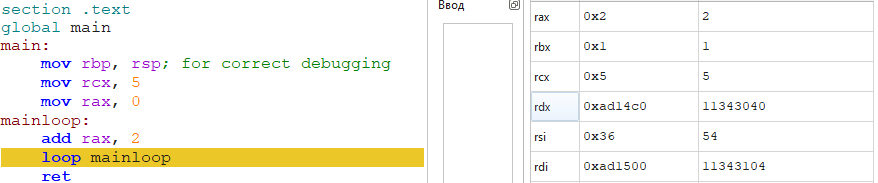


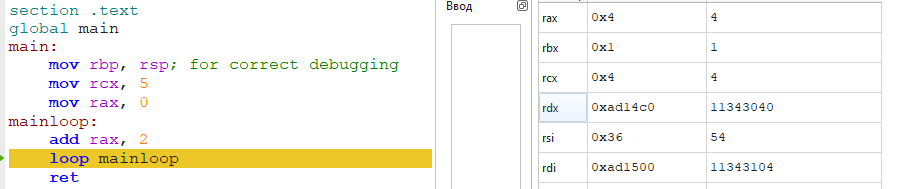
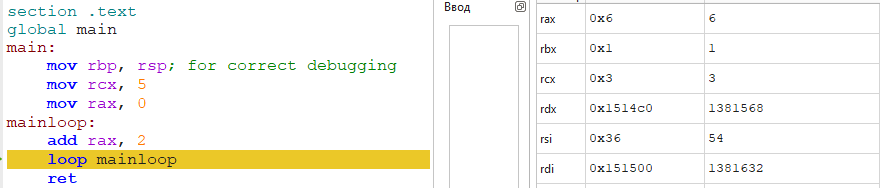
4.

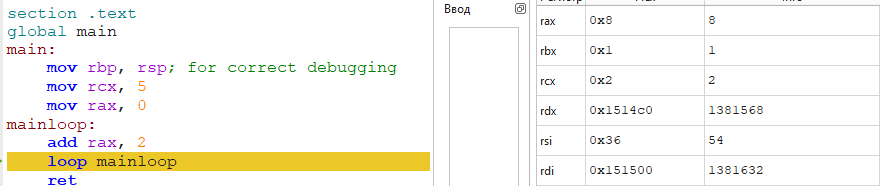


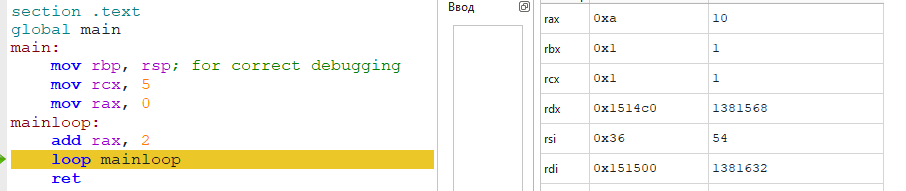


6.

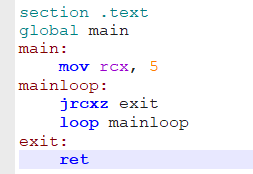






7.



**Контрольные вопросы**

1. Какие директивы используются для определения данных в NASM?:

* db (define byte): определяет целое число размером в 1 байт
* dw (define word): определяет целое число размером в 2 байта (слово)
* dd (define dword): определяет целое число размером в 4 байта (так называемое двойное слово
* dq (define qword/quad): определяет целое число размером в 8 байт (четверное слово)

1. Чем отличается секция .data от .bss?

* Секция data - инициализированные данные или секция, для чтения и записи, но не для выполнения. Секция bss - секция не инициализированных данных, допустимо чтение данных и запись в секцию

1. Как работает инструкция CMP и какие флаги она изменяет?:

* Инструкция CMP (от слова compare - сравнить) позволяет сравнить значения и установить флаги. Благодаря чему мы можем использовать результат сравнения для выполнения условного перехода
* CF (флаг переноса): устанавливается, если происходит беззнаковое переполнение, то есть при сумме с переносом или вычитании с заимствованием (например, при сумме чисел 0FFh и 01h). Если переполнение не происходит, то флаг не устанавливается.
* OF (флаг переполнения): установливается, если происходит переполнение со знаком, а именно когда переполняется бит, следующий за старшим знаковым битом. Например, при сумме чисел 7Fh и 01h. В двоичной системе это будет операция 0111 1111 + 0000 0001, результатом которой формально будет число 1000 0000. Таким образом, произошло изменение старшего знакового бита.
* SF (флаг знака): устанавливается, если старший бит результата установлен. В противном случае флаг знака сброшен (то есть флаг знака отражает состояние старшего бита результата).
* ZF (флаг нуля): устанавливается, если результат вычисления дает 0. Если результат ненулевой, флаг сброшен

1. Какие инструкции используются для условных переходов?:

* je: проверяет условие ZF == 1 и выполняет переход, если оба операнда равны. Фактически эквивалентна инструкции jz
* jne: проверяет условие ZF == 0 и выполняет переход, если оба операнда НЕ равны. Фактически эквивалентна инструкции jnz
* ja / jnbe: проверяет одновременно два условия СF == 0 и ZF == 0 (оба условия должны быть истинными). Выполняет переход, если первый операнд больше второго. Оба операнда беззнаковые.
* jae / jnb: проверяет условие СF == 0 и выполняет переход, если первый операнд больше или равен второму. Оба операнда беззнаковые. Аналогичен инструкции jnc
* jb / jnae: проверяет условие СF == 1 и выполняет переход, если первый операнд меньше второго. Оба операнда беззнаковые. Аналогичен инструкции jc.
* jbe / jna: проверяет одновременно два условия СF == 1 и ZF == 1 (достаточно, чтобы выполнялось хотя бы одно из этих условий). Выполняет переход, если первый операнд меньше или равен второму. Оба операнда беззнаковые.
* jg / jnle: проверяет одновременно два условия SF == OF и ZF == 0 (оба условия должны быть истинными). Выполняет переход, если первый операнд больше второго. Оба операнда со знаком.
* jge / jnl: проверяет условие SF == OF и выполняет переход, если первый операнд больше или равен второму. Оба операнда со знаком.
* jl / jnge: проверяет условие SF != OF (флаги SF и OF не должны быть равны) и выполняет переход, если первый операнд меньше второго. Оба операнда со знаком.
* jle / jng: проверяет одновременно два условия SF != OF и ZF == 1 (достаточно, чтобы выполнялось хотя бы одно из этих условий). Выполняет переход, если первый операнд меньше или равен второму. Оба операнда со знаком.

1. Как работает инструкция LOOP и в чем ее отличие от JRCXZ?

* Инструкция loop позволяют сократить определение цикла. Она уменьшает на 1 число в регистре RCX и переходит к определенной метке, если RCX не равен нулю

jrcxz - она проверяет значение RCX, и если оно рано 0, то переходит к определенной метке.