

ЭВМ и ПУ_Ассемблер_лаборат...

Файл

D:/YandexDisk/University/Computers_and_Peripherals/ЭВМ%20и%20ПУ...

22100

23

третьих, рассматривается ввод шестнадцатеричных чисел с клавиатуры.

В ходе работы производится знакомство с очень важными понятиями флагов состояния, стека и процедуры. Изучаются инструкции для работы с этими объектами, а также инструкции сдвига, цикла, условных переходов и некоторые другие.

Одной из целей работы является развитие навыков алгоритмизации задач и отладки программ.

Флаг переноса

Если выполнить сложение чисел 1 и FFFFh, то получим 10000h. Это число не может быть записано в шестнадцатитбитное слово, т.к. в нем помещаются только четыре шестнадцатеричные цифры. Единица в результате называется переполнением. Она записывается в специальную ячейку, называемую флагом переноса CF (от "Carry Flag"). Флаг содержит число, состоящее из одного бита, т.е. содержит или единицу или ноль. Если флаг содержит единицу, то говорят, что он "установлен", а если ноль – "сброшен".

Выполните загрузку чисел 1 и FFFFh в регистры BX и AX и запишите в память инструкцию ADD AX,BX. После этого протрассируйте инструкцию ADD. В конце второй строки распечатки, полученной с помощью команды R Debug, вы увидите восемь пар букв. Последняя пара выглядит как CY (от "Carry Ye" - перенос есть), т.е. флаг переноса установлен.

Установите IP в 100h и прибавьте единицу к нулю в AX, повторив трассировку инструкции сложения. Флаг переноса переустанавливается в каждой операции сложения, и так как на этот раз переполнения не будет, то флаг будет сброшен. С помощью команды R проверьте, что в качестве состояния флага CF листинг содержит NC ("от No Carry" – нет переноса).

Циклический сдвиг

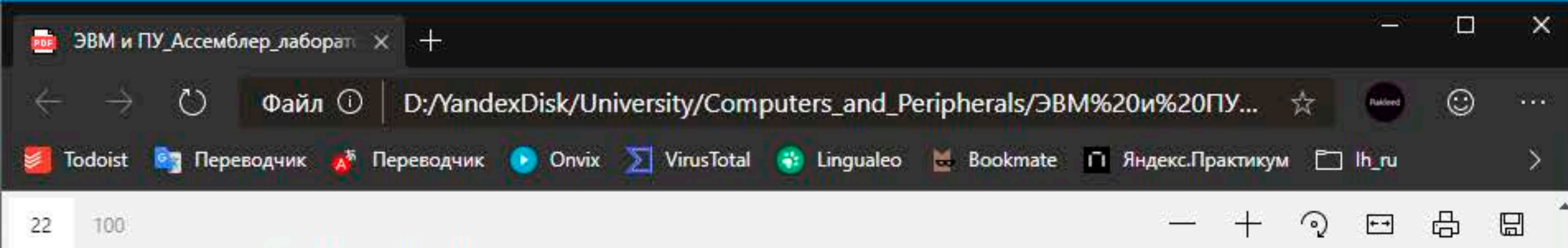
Допустим, что нам надо выполнить вывод на экран двоичного числа. За шаг мы печатаем только один символ, и нам надо произвести выборку всех битов двоичного числа, одного за другим, слева направо. Например, пусть требуемое число есть 10000000b. Если мы сдвинем весь этот байт влево на одну позицию, помещая единицу во флаг переноса и добавляя ноль справа,

Windows XP Professional - VMware Workstation

ФайлПравкаВидВиртуальная машинаВкладкиСправка

ГлавнаяWindows XP Professional

Для входа в эту виртуальную машину напрямую, переместите указатель мыши внутрь или нажмите Ctrl+G.



некоторые другие.

Одной из целей работы является развитие навыков алгоритмизации задач и отладки программ.

Флаг переноса

Если выполнить сложение чисел 1 и FFFFh, то получим 10000h. Это число не может быть записано в шестнадцатитбитное слово, т.к. в нем помещаются только четыре шестнадцатеричные цифры. Единица в результате называется переполнением. Она записывается в специальную ячейку, называемую флагом переноса CF (от "Carry Flag"). Флаг содержит число, состоящее из одного бита, т.е. содержит или единицу или ноль. Если флаг содержит единицу, то говорят, что он "установлен", а если ноль – "сброшен".

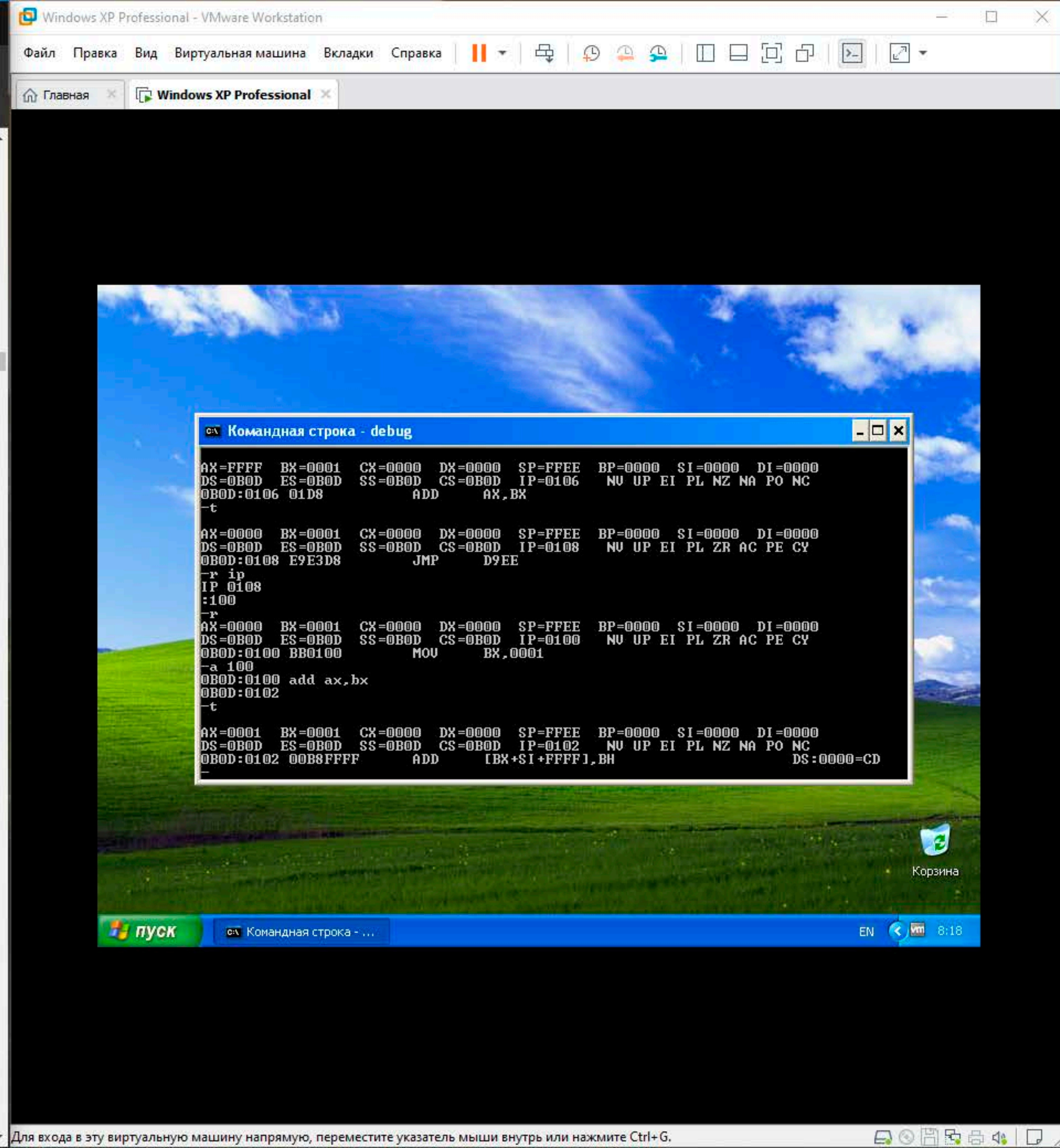
Выполните загрузку чисел 1 и FFFFh в регистры BX и AX и запишите в память инструкцию ADD AX, BX. После этого протрассируйте инструкцию ADD. В конце второй строки распечатки, полученной с помощью команды R Debug, вы увидите восемь пар букв. Последняя пара выглядит как CY (от "Carry Ye" - перенос есть), т.е. флаг переноса установлен.

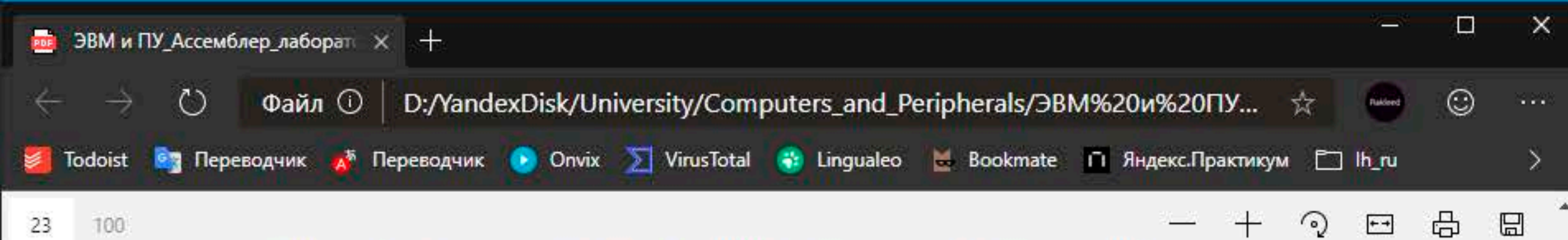
Установите IP в 100h и прибавьте единицу к нулю в AX, повторив трассировку инструкции сложения. Флаг переноса переустанавливается в каждой операции сложения, и так как на этот раз переполнения не будет, то флаг будет сброшен. С помощью команды R проверьте, что в качестве состояния флага CF листинг содержит NC ("от No Carry" – нет переноса).

Циклический сдвиг

Допустим, что нам надо выполнить вывод на экран двоичного числа. За шаг мы печатаем только один символ, и нам надо произвести выборку всех битов двоичного числа, одного за другим, слева направо. Например, пусть требуемое число есть 10000000b. Если мы сдвинем весь этот байт влево на одну позицию, помещая единицу во флаг переноса и добавляя ноль справа, и затем повторим этот процесс для каждой последующей цифры, во флаге переноса будут по очереди содержаться все цифры нашего двоичного числа.

Инструкция RCL (от "Rotate Carry Left" – циклический сдвиг влево с переносом) сдвигает крайний левый бит во флаг переноса (в примере это 1), в то время как бит, находившийся до этого во флаге переноса, сдвигается в





проверьте, но в качестве состояния флага... No Carry" – нет переноса).

Циклический сдвиг

Допустим, что нам надо выполнить вывод на экран двоичного числа. За шаг мы печатаем только один символ, и нам надо произвести выборку всех битов двоичного числа, одного за другим, слева направо. Например, пусть требуемое число есть 10000000b. Если мы сдвинем весь этот байт влево на одну позицию, помещая единицу во флаг переноса и добавляя ноль справа, и затем повторим этот процесс для каждой последующей цифры, во флаге переноса будут по очереди содержаться все цифры нашего двоичного числа.

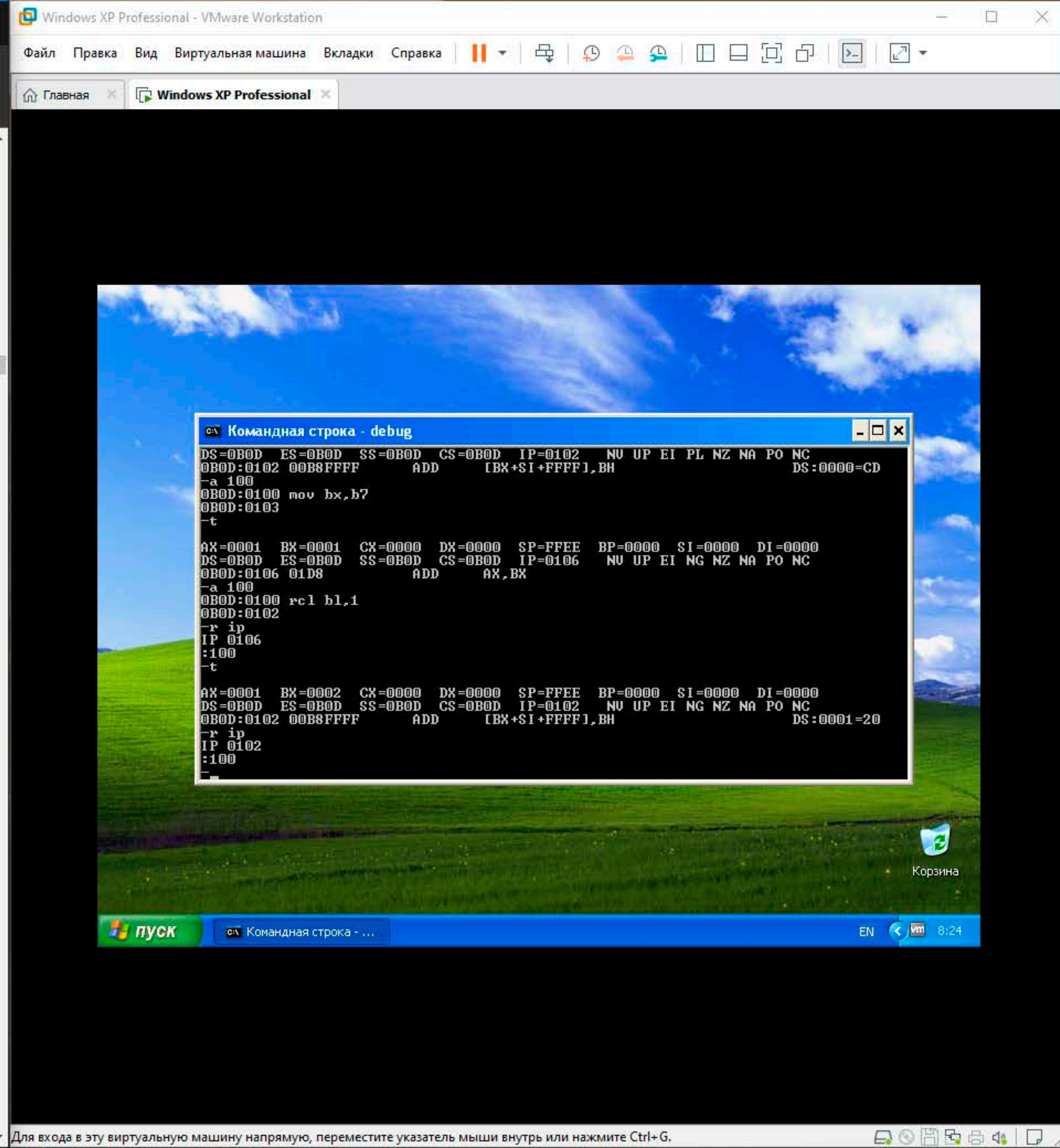
Инструкция RCL (от "Rotate Carry Left" – циклический сдвиг влево с переносом) сдвигает крайний левый бит во флаг переноса (в примере это 1), в то время как бит, находившийся до этого во флаге переноса, сдвигается в

24

крайне правую позицию (т.е. в нулевой бит). В процессе сдвига все остальные биты сдвигаются влево. После определенного количества циклических сдвигов (17 для слова, 9 для байта) биты возвращаются на их начальные позиции, и вы получаете исходное число.

В ы п о л н и т е с помощью Debug размещение по адресу 100h инструкции RCL BL,1, которая циклически сдвигает байт в BL влево на один бит, используя флаг переноса. Поместите в регистр BX число B7h и протрассируйте эту инструкцию несколько раз. Убедитесь, что после 9 циклов регистр BX содержит опять B7h.

Как напечатать двоичное значение флага переноса? Из таблицы кодов ASCII видно, что символ "0" есть 30h, а символ "1" есть 31h.. Таким образом, сложение флага переноса и 30h дает символ "0", когда флаг сброшен и символ "1", когда он установлен. Для выполнения такого сложения удобно использовать инструкцию ADC (от "Add with Carry" – сложение с переносом). Эта инструкция складывает три числа: два числа, как и инструкция ADD, ПЛЮС один бит из флага переноса.



ЭВМ и ПУ_Ассемблер_лаборат...

Файл

D:/YandexDisk/University/Computers_and_Peripherals/ЭВМ%20и%20ПУ...

23100

шаг мы печатаем только один символ, и нам надо произвести выборку всех битов двоичного числа, одного за другим, слева направо. Например, пусть требуемое число есть 10000000b. Если мы сдвинем весь этот байт влево на одну позицию, помещая единицу во флаг переноса и добавляя ноль справа, и затем повторим этот процесс для каждой последующей цифры, во флаге переноса будут по очереди содержаться все цифры нашего двоичного числа.

Инструкция RCL (от "Rotate Carry Left" – циклический сдвиг влево с переносом) сдвигает крайний левый бит во флаг переноса (в примере это 1), в то время как бит, находившийся до этого во флаге переноса, сдвигается в

24

крайне правую позицию (т.е. в нулевой бит). В процессе сдвига все остальные биты сдвигаются влево. После определенного количества циклических сдвигов (17 для слова, 9 для байта) биты возвращаются на их начальные позиции, и вы получаете исходное число.

Выполните с помощью Debug размещение по адресу 100h инструкции RCL BL,1, которая циклически сдвигает байт в BL влево на один бит, используя флаг переноса. Поместите в регистр BX число B7h и протрассируйте эту инструкцию несколько раз. Убедитесь, что после 9 циклов регистр BX содержит опять B7h.

Как напечатать двоичное значение флага переноса? Из таблицы кодов ASCII видно, что символ "0" есть 30h, а символ "1" есть 31h.. Таким образом, сложение флага переноса и 30h дает символ "0", когда флаг сброшен и символ "1", когда он установлен. Для выполнения такого сложения удобно использовать инструкцию ADC (от "Add with Carry" - сложение с переносом). Эта инструкция складывает три числа: два числа, как и инструкция ADD, ПЛЮС один бит из флага переноса.

Поместите в память после инструкции RCL BL,1 инструкцию ADC DL,30, которая выполнит сложение содержимого DL (0), 30h и флага переноса, поместив результат в DL. Записав далее инструкции, обеспечивающие вывод символа на экран и завершение программы, получим программу, выполняющую печать старшего бита регистра BL:

Windows XP Professional - VMware Workstation

ФайлПравкаВидВиртуальная машинаВкладкиСправка

ГлавнаяWindows XP Professional

Командная строка - debug

```
-r ip
IP 0102
:100
-t

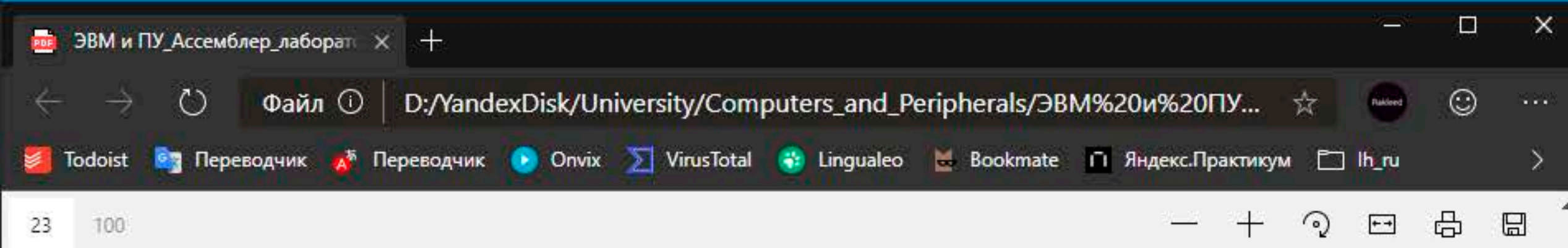
AX=0001 BX=0080 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B0D ES=0B0D SS=0B0D CS=0B0D IP=0102  OV UP EI NG NZ NA PO NC
0B0D:0102 00B8FFFF ADD     [BX+SI+FFFF],BH          DS:007F=00
-r ip
IP 0102
:100
-t

AX=0001 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B0D ES=0B0D SS=0B0D CS=0B0D IP=0102  OV UP EI NG NZ NA PO CY
0B0D:0102 00B8FFFF ADD     [BX+SI+FFFF],BH          DS:FFFF=00
-r ip
IP 0102
:100
-t

AX=0001 BX=0001 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B0D ES=0B0D SS=0B0D CS=0B0D IP=0102  NU UP EI NG NZ NA PO NC
0B0D:0102 00B8FFFF ADD     [BX+SI+FFFF],BH          DS:0000=CD
```

пусксх Командная строка - ...EN8:27

Для входа в эту виртуальную машину напрямую, переместите указатель мыши внутрь или нажмите Ctrl+G.



крайне правую позицию (т.е. в нулевой бит). В процессе сдвига все остальные биты сдвигаются влево. После определенного количества циклических сдвигов (17 для слова, 9 для байта) биты возвращаются на их начальные позиции, и вы получаете исходное число.

В ы п о л н и т е с помощью Debug размещение по адресу 100h инструкции RCL BL,1, которая циклически сдвигает байт в BL влево на один бит, используя флаг переноса. Поместите в регистр BX число B7h и протрассируйте эту инструкцию несколько раз. Убедитесь, что после 9 циклов регистр BX содержит опять B7h.

Как напечатать двоичное значение флага переноса? Из таблицы кодов ASCII видно, что символ "0" есть 30h, а символ "1" есть 31h. Таким образом, сложение флага переноса и 30h дает символ "0", когда флаг сброшен и символ "1", когда он установлен. Для выполнения такого сложения удобно использовать инструкцию ADC (от "Add with Carry" - сложение с переносом). Эта инструкция складывает три числа: два числа, как и инструкция ADD, ПЛЮС один бит из флага переноса.

П о м е с т и т е в память после инструкции RCL BL,1 инструкцию ADC DL,30, которая выполнит сложение содержимого DL (0), 30h и флага переноса, поместив результат в DL. Записав далее инструкции, обеспечивающие вывод символа на экран и завершение программы, получим программу, выполняющую печать старшего бита регистра BL:

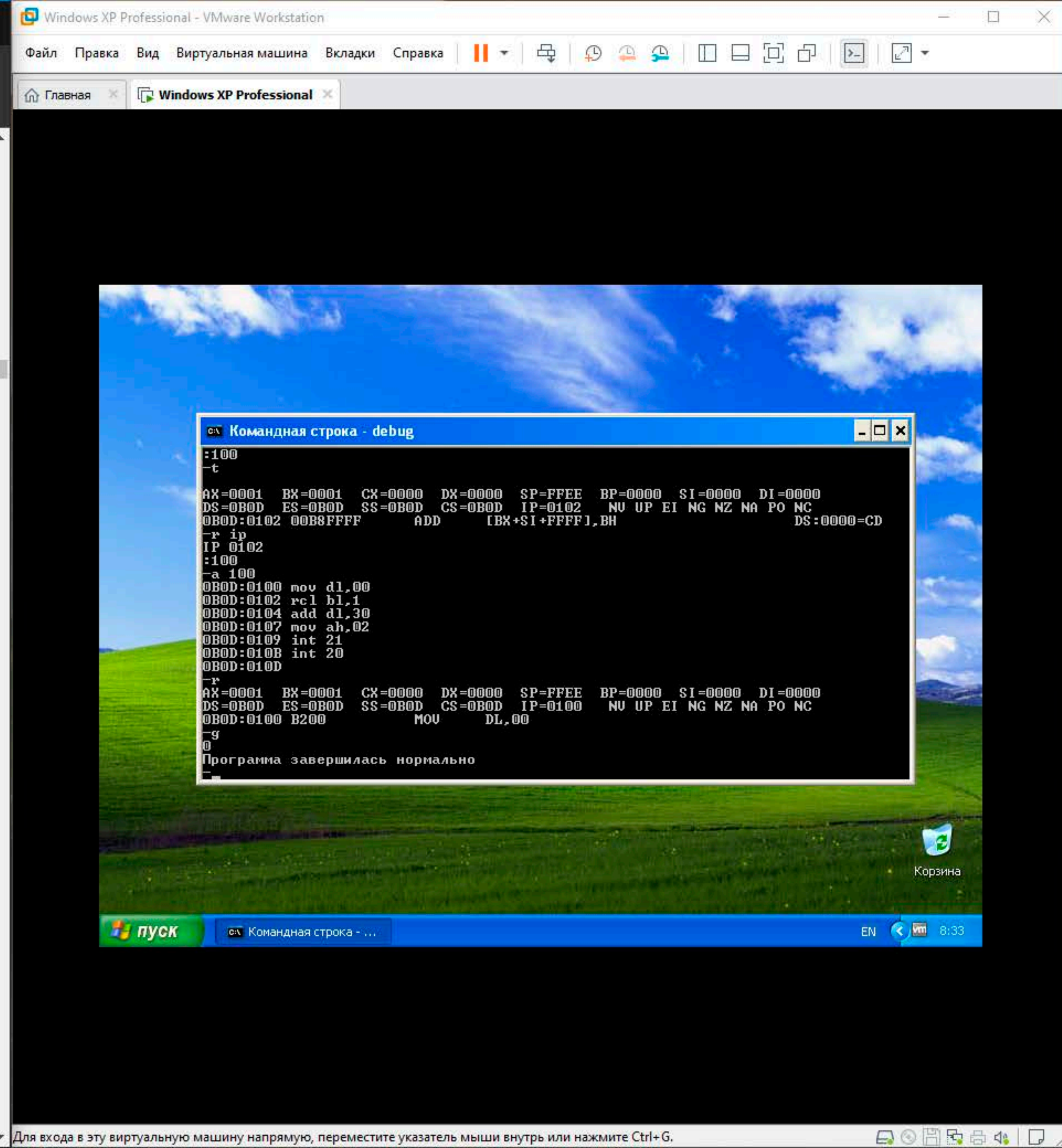
```
MOV    DL, 00
RCL    BL, 1
ADC    DL, 30
MOV    AH, 02
INT    21
INT    20
```

В ы п о л н и т е эту программу для обоих значений старшего бита BL. Для записи в BX используйте команду R Debug.

Организация циклов

Если мы хотим распечатать все биты BL, то мы должны повторить операции циклического сдвига и распечатки флага переноса CF восемь раз (число битов в BL). Неоднократное повторение одних и тех же операций называется циклом.

Соответствующий алгоритм приведен на рис. 1. На первом этапе переменной CX присваивается значение 8. Именно столько раз мы хотим



ЭВМ и ПУ_Ассемблер_лаборат...

Файл

D:/YandexDisk/University/Computers_and_Peripherals/ЭВМ%20и%20ПУ...

25100

полученное значение CX не равно 0, делается переход по адресу, заданному в качестве операнда инструкции LOOP. Таким образом, наличие данной инструкции обеспечивает реализацию двух этапов алгоритма, приведенного на рис. 1.

26

В качестве примера применения инструкции LOOP приведем программу, выполняющую вывод на экран четырех звездочек:

100	MOV	AH, 02
102	MOV	DL, 2A
104	MOV	CX, 4
107	INT	21
109	LOOP	107
10B	INT	20

В ы п о л н и т е трассировку данной программы, наблюдая за содержимым регистров IP и CX. При этом следует помнить, что не следует использовать команду T для инструкции INT. При достижении этой инструкции следует набрать команду G d, где d - адрес в памяти инструкции, следующей за инструкцией INT. Эта команда сообщит Debug о том, что надо продолжить выполнение программы до того момента, когда IP достигнет значения введенного адреса. Таким образом Debug будет выполнять инструкцию INT без трассировки и останавливаться при достижении инструкции, следующей за INT. Адрес d называется точкой останова. При достижении инструкции INT 20 вводится команда G.

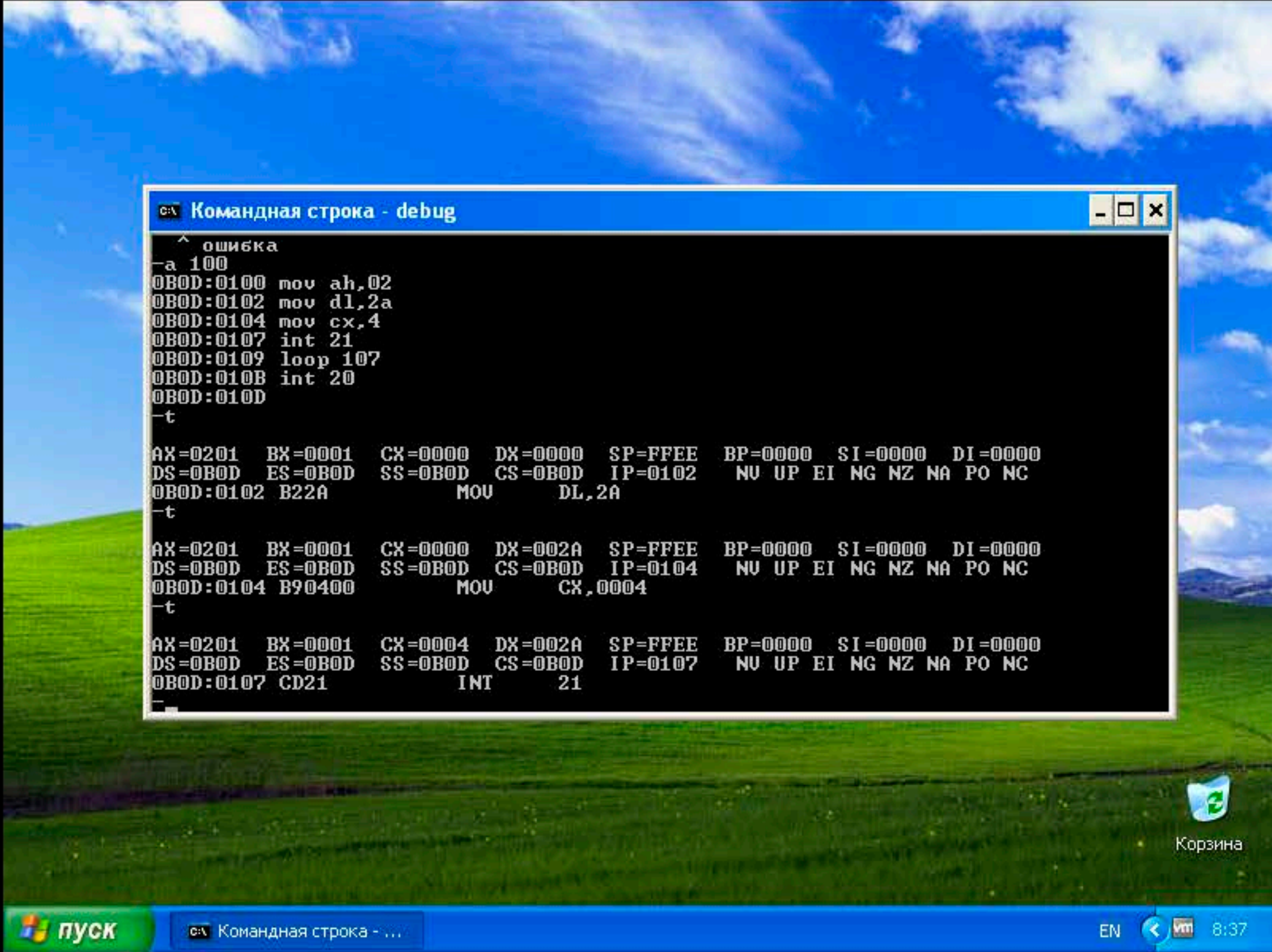
При применении команды G d необходимо знать адрес d. Исключить трассировку при достижении инструкции INT проще, если использовать команду Debug P (от "Proceed" – переходить, продолжать). Эта команда является удобным средством обхода инструкций, вызывающих подпрограммы DOS.

В ы п о л н и т е написание и ввод в память программы вывода двоичного содержимого байта (содержится в регистре BL) на экран

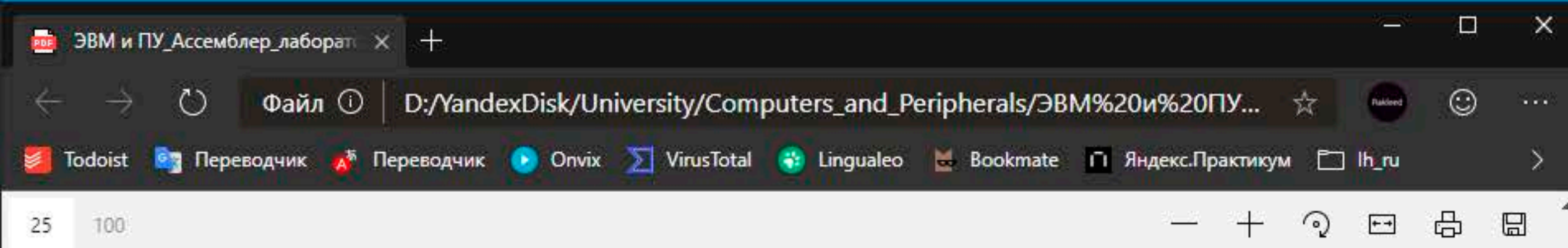
Windows XP Professional - VMware Workstation

ФайлПравкаВидВиртуальная машинаВкладкиСправка

ГлавнаяWindows XP Professional



Для входа в эту виртуальную машину напрямую, переместите указатель мыши внутрь или нажмите Ctrl+G.



В качестве примера применения инструкции LOOP приведем программу, выполняющую вывод на экран четырех звездочек:

100	MOV	AX, 02
102	MOV	DL, 2A
104	MOV	CX, 4
107	INT	21
109	LOOP	107
10B	INT	20

Выполните трассировку данной программы, наблюдая за содержимым регистров IP и CX. При этом следует помнить, что не следует использовать команду T для инструкции INT. При достижении этой инструкции следует набрать команду G d, где d - адрес в памяти инструкции, следующей за инструкцией INT. Эта команда сообщит Debug о том, что надо продолжить выполнение программы до того момента, когда IP достигнет значения введенного адреса. Таким образом Debug будет выполнять инструкцию INT без трассировки и останавливаться при достижении инструкции, следующей за INT. Адрес d называется точкой останова. При достижении инструкции INT 20 вводится команда G.

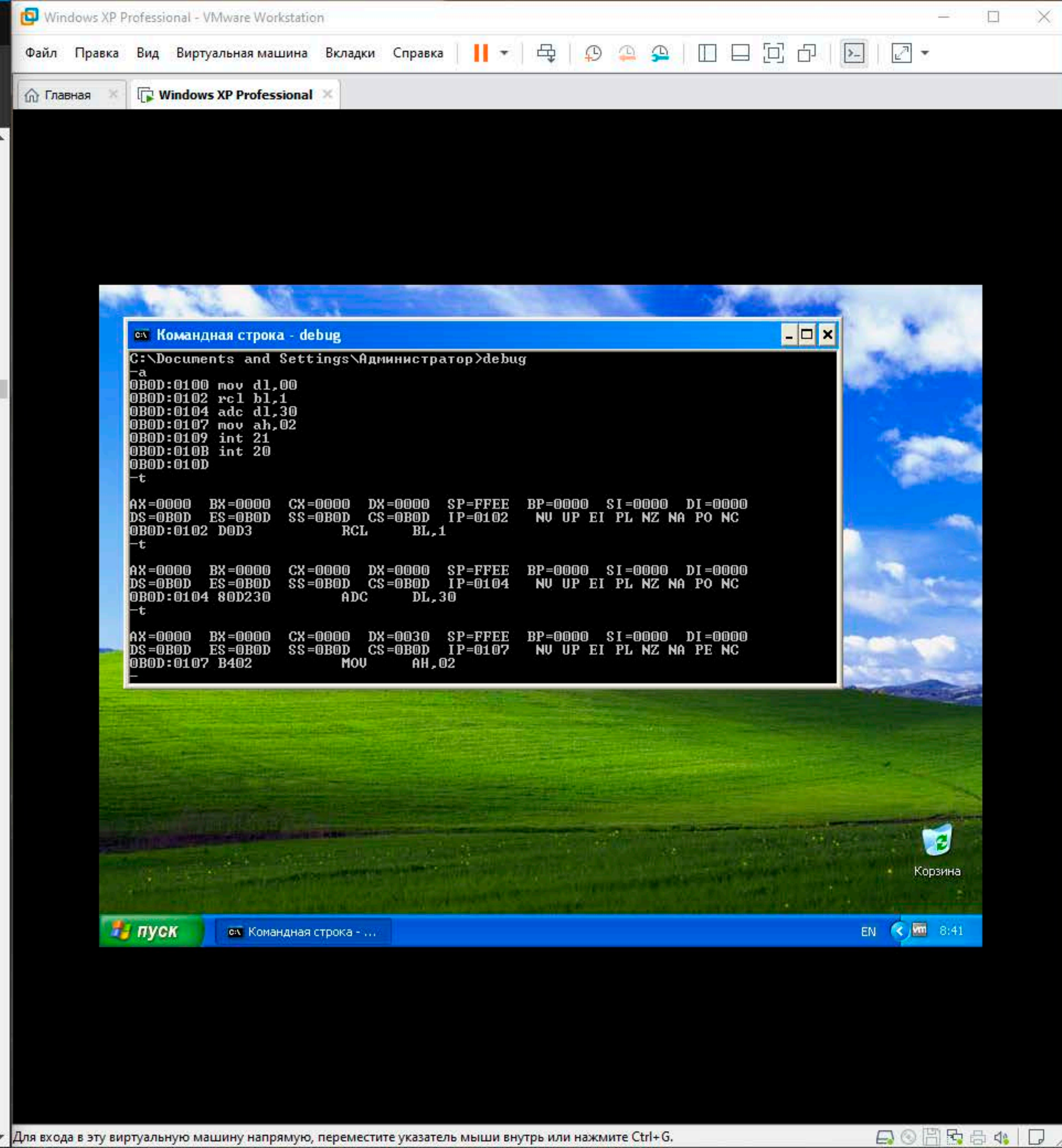
При применении команды G d необходимо знать адрес d. Исключить трассировку при достижении инструкции INT проще, если использовать команду Debug P (от "Proceed" - переходить, продолжать). Эта команда является удобным средством обхода инструкций, вызывающих подпрограммы DOS.

Выполните написание и ввод в память программы вывода двоичного содержимого байта (содержится в регистре BL) на экран (алгоритм на рис. 1).

Отладка программы

Она включает поиск ошибок в программе (тестирование программы) и их исправление. Пока наши программы достаточно просты, и каждая из них включает всего одну подпрограмму. Что касается отладки программ, состоящих из нескольких подпрограмм, то она будет рассматриваться позднее. Пока лишь заметим, что такая отладка может рассматриваться как последовательность отладок подпрограмм.

Тестирование программы выполняется при различных значениях ее входных данных. Если очередной прогон программы показал наличие в ней ошибки, то производится ее поиск. Как раз для такого поиска и предназначена трассировка программы.



С:\ Командная строка - debug

```

0B0D:010D
-t
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B0D ES=0B0D SS=0B0D CS=0B0D IP=0102  NU UP EI PL NZ NA PO NC
0B0D:0102 D0D3          RCL     BL,1
-t
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B0D ES=0B0D SS=0B0D CS=0B0D IP=0104  NU UP EI PL NZ NA PO NC
0B0D:0104 80D230        ADC     DL,30
-t
AX=0000 BX=0000 CX=0000 DX=0030 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B0D ES=0B0D SS=0B0D CS=0B0D IP=0107  NU UP EI PL NZ NA PE NC
0B0D:0107 B402          MOV     AH,02
-t
AX=0200 BX=0000 CX=0000 DX=0030 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B0D ES=0B0D SS=0B0D CS=0B0D IP=0109  NU UP EI PL NZ NA PE NC
0B0D:0109 CD21          INT     21
-g
0
Программа завершилась нормально
-

```

Пуск С:\ Командная строка - ... EN 8:41

Если программа длинная, то ее пошаговая трассировка не очень удобна. В этом случае сначала желательно локализовать ошибку, определив содержащий ее фрагмент программы. Далее этот фрагмент исследуется

ЭВМ и ПУ_Ассемблер_лаборат...

Файл

D:/YandexDisk/University/Computers_and_Peripherals/ЭВМ%20и%20ПУ...

26100

Тестирование программы выполняется при различных значениях ее входных данных. Если очередной прогон программы показал наличие в ней ошибки, то производится ее поиск. Как раз для такого поиска и предназначена трассировка программы.
Если программа длинная, то ее пошаговая трассировка не очень удобна. В этом случае сначала желательно локализовать ошибку, определив содержащий ее фрагмент программы. Далее этот фрагмент исследуется

27

более подробно. Для локализации ошибки мы выбираем несколько точек останова. Для выбора этих адресов, разбивающих программу на фрагменты, используется листинг программы, а также ее блок-схема. Далее, с помощью команды G d производится анализ работы программы в каждой из точек останова. Если в очередной точке останова результаты работы программы неверны, то данная точка завершает искомый фрагмент.

В ы п о л н и т е отладку введенной ранее в память программы вывода двоичного содержимого байта. Тестирование программы проведите с помощью команды G, предварительно загружая с помощью команды R в регистр BX различные пары шестнадцатеричных цифр. (Debug не имеет команд для работы с однобайтовыми регистрами и поэтому BL заполняется как часть BX.) В случае обнаружения ошибки выполните пошаговую трассировку, или используйте точки останова.

Флаги состояния

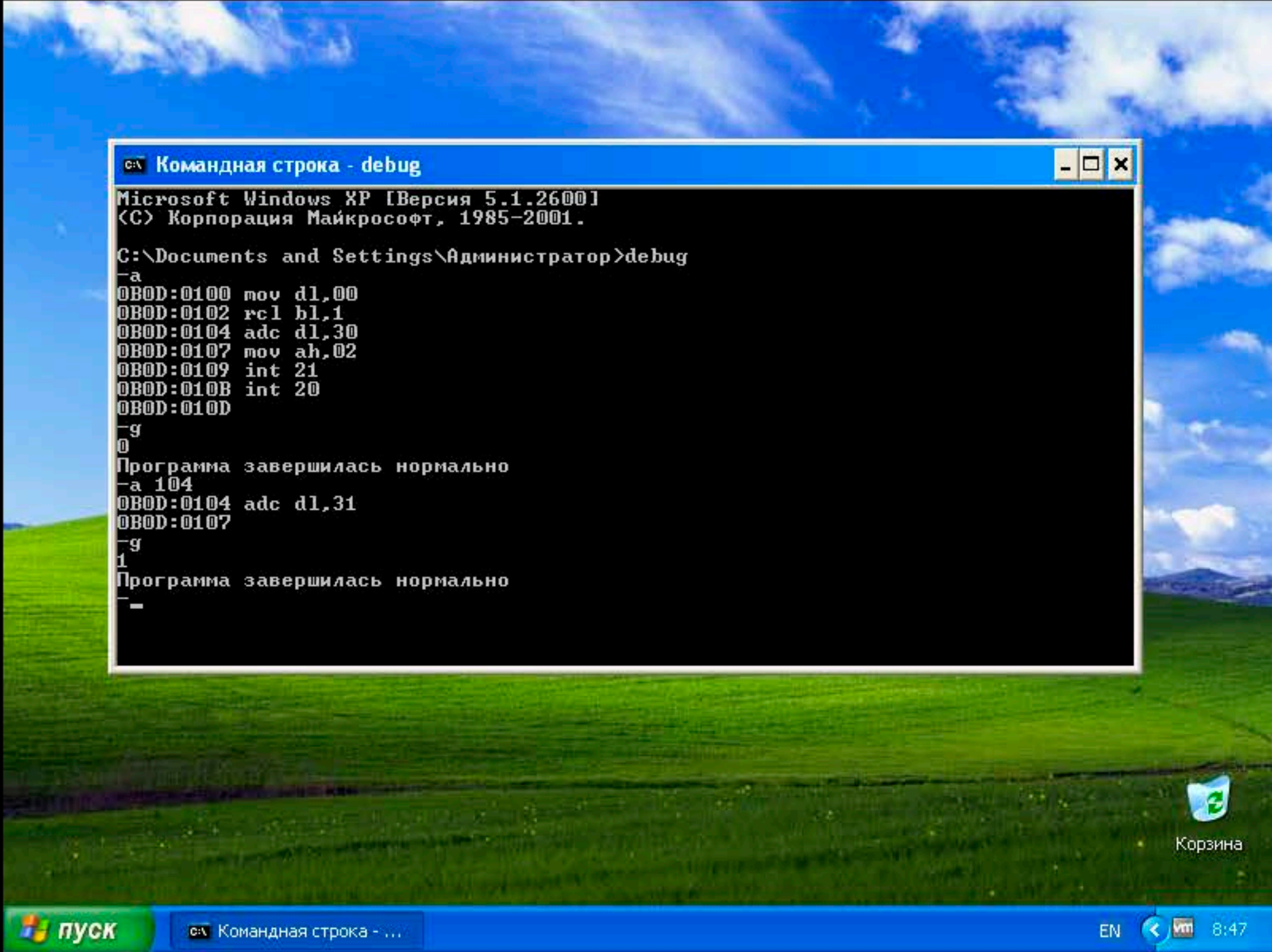
Кроме флага переноса CF существуют другие флаги состояния. Рассмотрим три из них, которые описывают результат последней арифметической операции.

Допустим, что мы выполнили инструкцию вычитания SUB. Одним из флагов состояния, устанавливаемых в зависимости от результата этой инструкции, является флаг нуля ("Zero Flag"). Если результат инструкции SUB есть 0, то флаг нуля будет установлен в 1. На распечатке регистров это значение обозначается как ZR (от "Zero" – нуль). Если результат арифметической операции не равен нулю, то флаг нуля сбрасывается в 0 – NZ (от "Not Zero" – не ноль).

Windows XP Professional - VMware Workstation

ФайлПравкаВидВиртуальная машинаВкладкиСправка

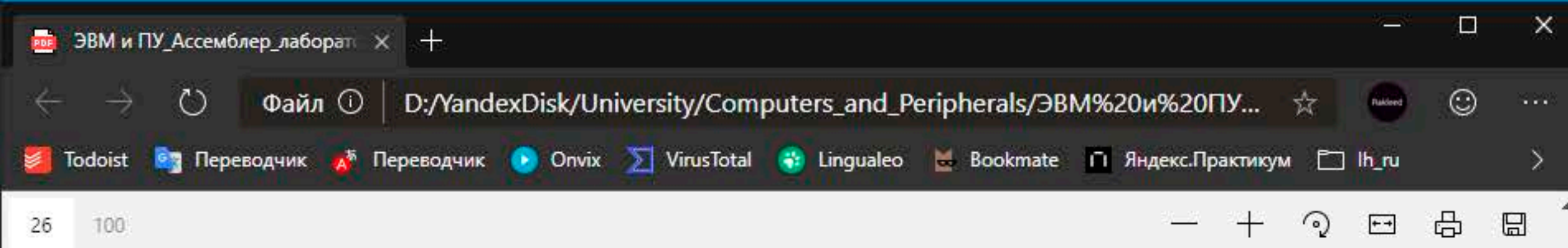
ГлавнаяWindows XP Professional



```
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.

C:\Documents and Settings\Администратор>debug
-a
0B0D:0100 mov dl,00
0B0D:0102 rcl bl,1
0B0D:0104 adc dl,30
0B0D:0107 mov ah,02
0B0D:0109 int 21
0B0D:010B int 20
0B0D:010D
-g
0
Программа завершилась нормально
-a 104
0B0D:0104 adc dl,31
0B0D:0107
-g
1
Программа завершилась нормально
-
```

Для входа в эту виртуальную машину напрямую, переместите указатель мыши внутрь или нажмите Ctrl+G.



SUB есть 0, то флаг нуля будет установлен в 1. На распечатке регистров это значение обозначается как ZR (от "Zero" – нуль). Если результат арифметической операции не равен нулю, то флаг нуля сбрасывается в 0 – NZ (от "Not Zero" – не ноль).

Введите в память инструкцию SUB AX,BX. Протрассируйте ее с одинаковыми и с разными числами в регистрах AX и BX, наблюдая за состоянием флага нуля (ZR или NZ).

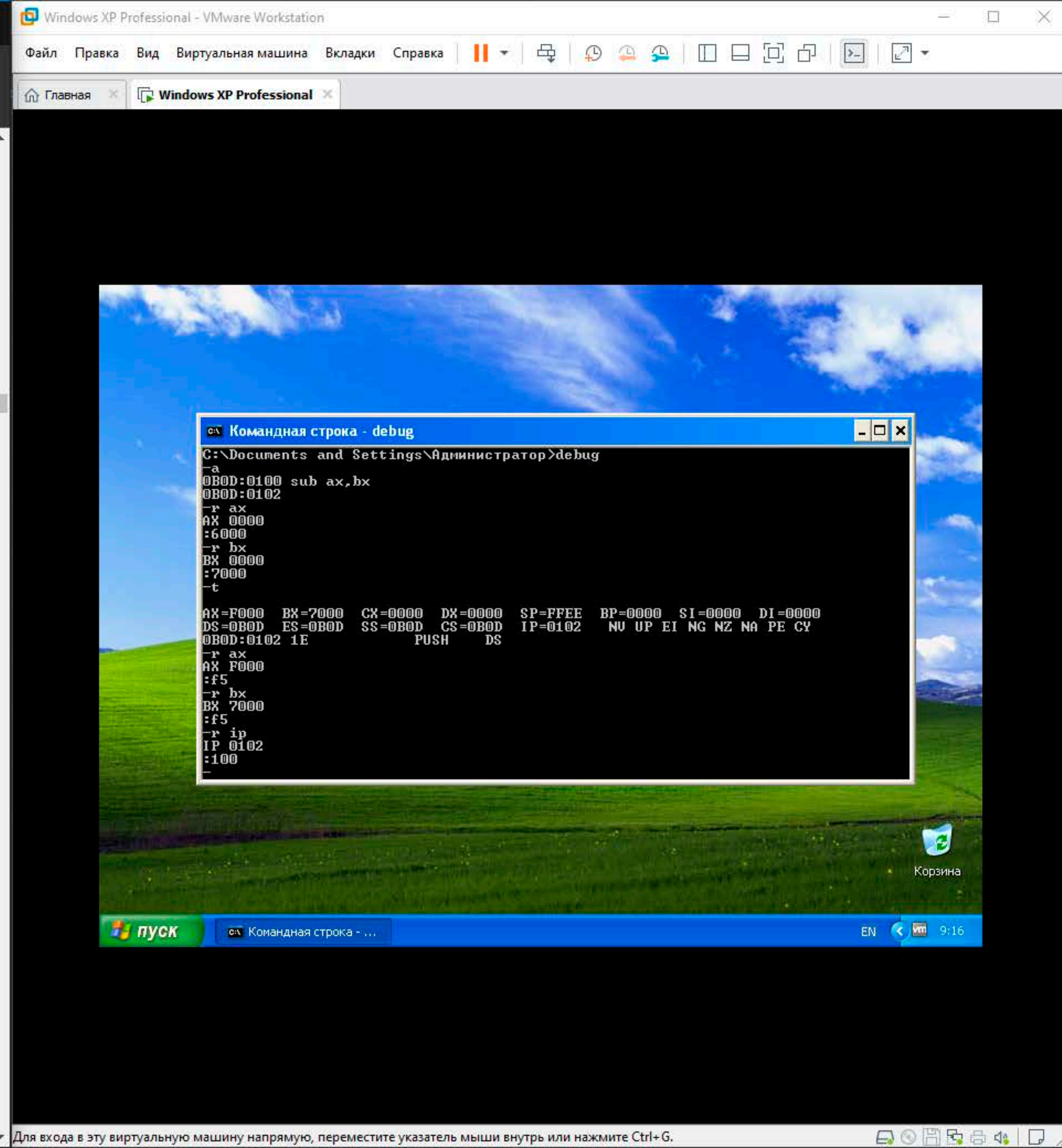
Флаг знака ("Sign Flag") принимает значение 1 (на распечатке регистров NG – от "Negative"), если результат предыдущей арифметической операции отрицательный. Если результат неотрицательный (ноль или положительный), то флаг знака принимает значение 0 (на распечатке PL – "Plus"). **Выполните** трассировку инструкции SUB AX,BX при разных содержимых AX и BX и наблюдая за флагом знака.

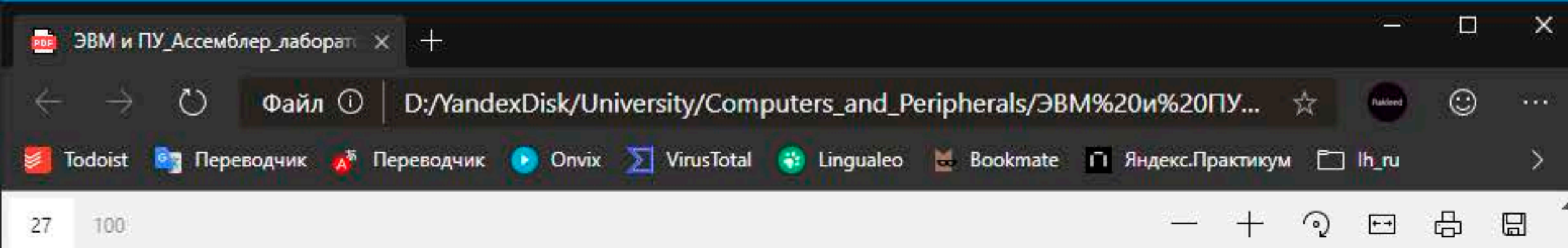
Флаг переполнения устанавливается в 1 в том случае, если знаковый бит изменился в той ситуации, когда этого не должно было произойти. Например, если мы сложим два положительных числа 7000h и 6000h, то получим отрицательное число D000h (представление в дополнительном коде числа –12288). Это ошибка, т.к. результат переполняет слово. На листинге регистров переполнение обозначается как OV ("Overflow" – переполнение). Если предыдущая арифметическая инструкция не дала переполнения, то флаг сбрасывается в 0. На распечатке регистров это значение обозначается как NV ("No Overflow"). **Проверьте**

установку флага переполнения протрассировав инструкцию SUB или ADD.

Использование инструкции SUB для сравнения двух чисел неудобно, т.к. эта инструкция производит изменение первого из чисел. Другая инструкция, CMP (от "Compare" – сравнение), производит сравнение двух чисел без их изменения. Результат сравнения используется только для установки флагов.

Загрузите в регистры AX и BX одинаковые числа, например F5h и протрассируйте инструкцию CMP AX,BX. При этом убедитесь, что установлен флаг нуля (ZR), но оба регистра сохранили свое значение –





бит изменился в той ситуации, когда этого не должно было произойти. Например, если мы сложим два положительных числа 7000h и 6000h, то получим отрицательное число D000h (представление в дополнительном коде числа -12288). Это ошибка, т.к. результат переполняет слово. На листинге регистров переполнение обозначается как OV ("Overflow" – переполнение). Если предыдущая арифметическая инструкция не дала переполнения, то флаг сбрасывается в 0. На распечатке регистров это значение обозначается как NV ("No Overflow"). *Проверьте*

28

установку флага переполнения протрассировав инструкцию SUB или ADD.

Использование инструкции SUB для сравнения двух чисел неудобно, т.к. эта инструкция производит изменение первого из чисел. Другая инструкция, CMP (от "Compare" – сравнение), производит сравнение двух чисел без их изменения. Результат сравнения используется только для установки флагов.

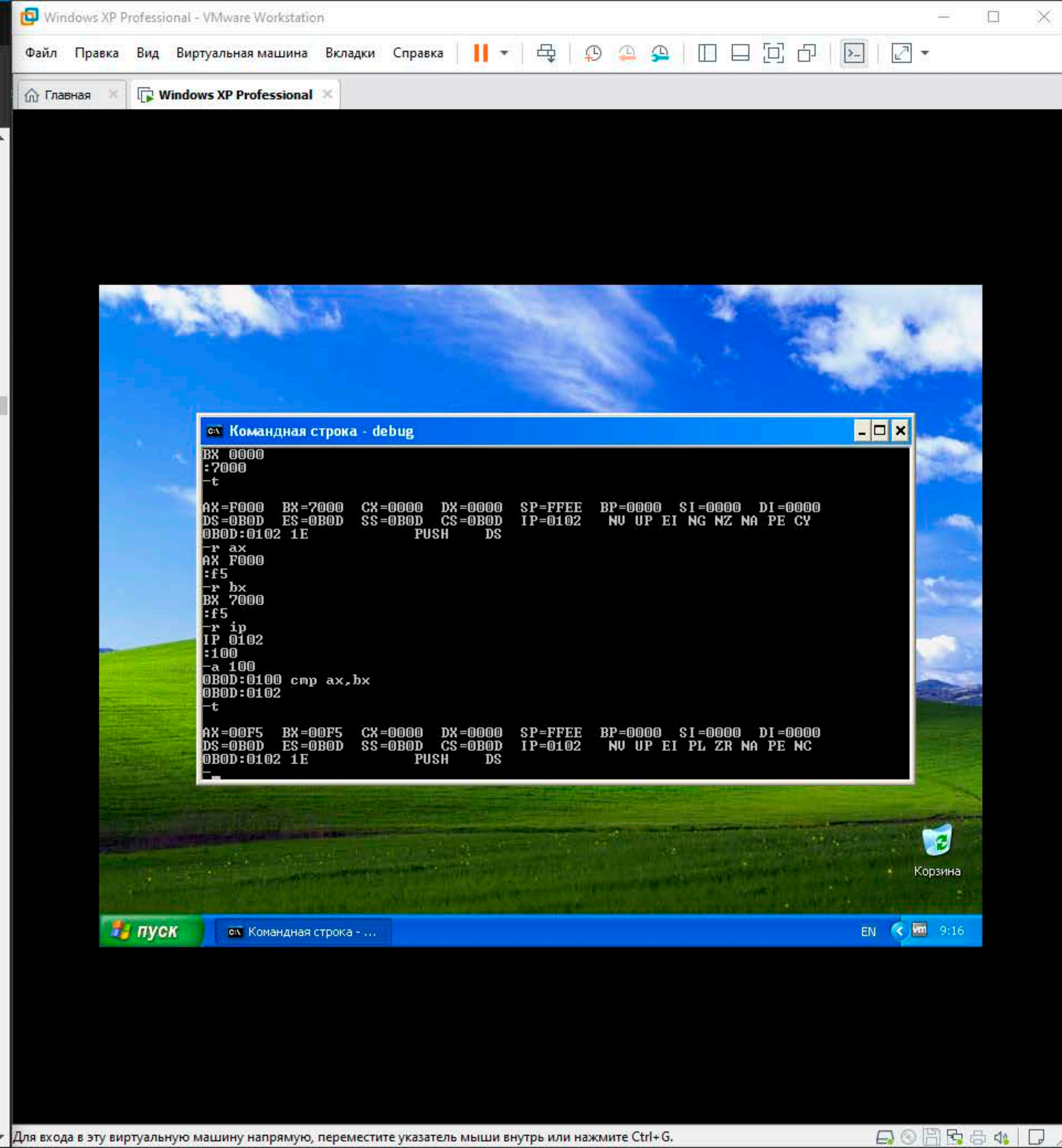
Загрузите в регистры AX и BX одинаковые числа, например F5h и протрассируйте инструкцию CMP AX,BX. При этом убедитесь, что установлен флаг нуля (ZR), но оба регистра сохранили свое значение – F5h.

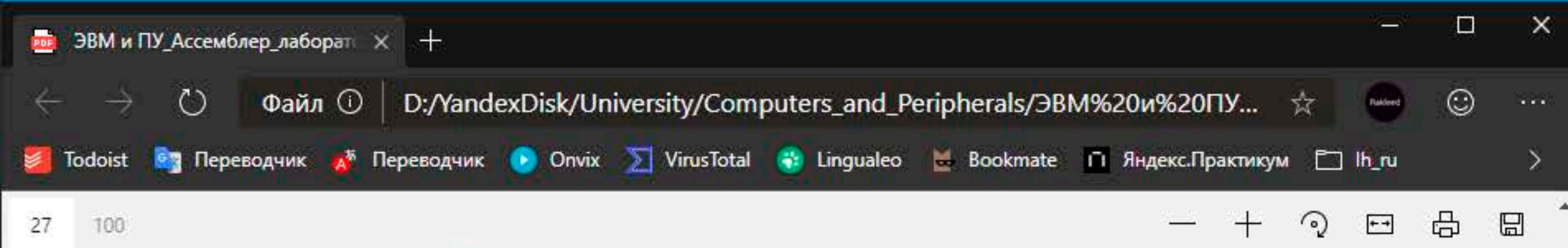
Инструкции условного перехода

Флаги состояния устанавливаются для того, чтобы можно было менять ход выполнения программы в зависимости от текущей ситуации. Анализ флагов состояния и соответствующие переходы в программе выполняют инструкции, называемые инструкциями условного перехода.

Инструкция JZ (от "Jump if Zero" – перейти, если ноль) проверяет флаг нуля, и если он установлен (ZR), то выполняется переход на новый адрес. Таким образом, если мы вслед за инструкцией SUB напишем, например, JZ 15A, то нулевой результат вычитания означает, что ЦП начнет выполнять инструкцию, находящуюся по адресу 15A, а не следующую по порядку инструкцию.

Противоположной по отношению к JZ является инструкция JNZ ("Jump





установки флагов.

Загрузите в регистры AX и BX одинаковые числа, например F5h и протрассируйте инструкцию CMP AX,BX. При этом убедитесь, что установлен флаг нуля (ZR), но оба регистра сохранили свое значение – F5h.

Инструкции условного перехода

Флаги состояния устанавливаются для того, чтобы можно было менять ход выполнения программы в зависимости от текущей ситуации. Анализ флагов состояния и соответствующие переходы в программе выполняют инструкции, называемые инструкциями условного перехода.

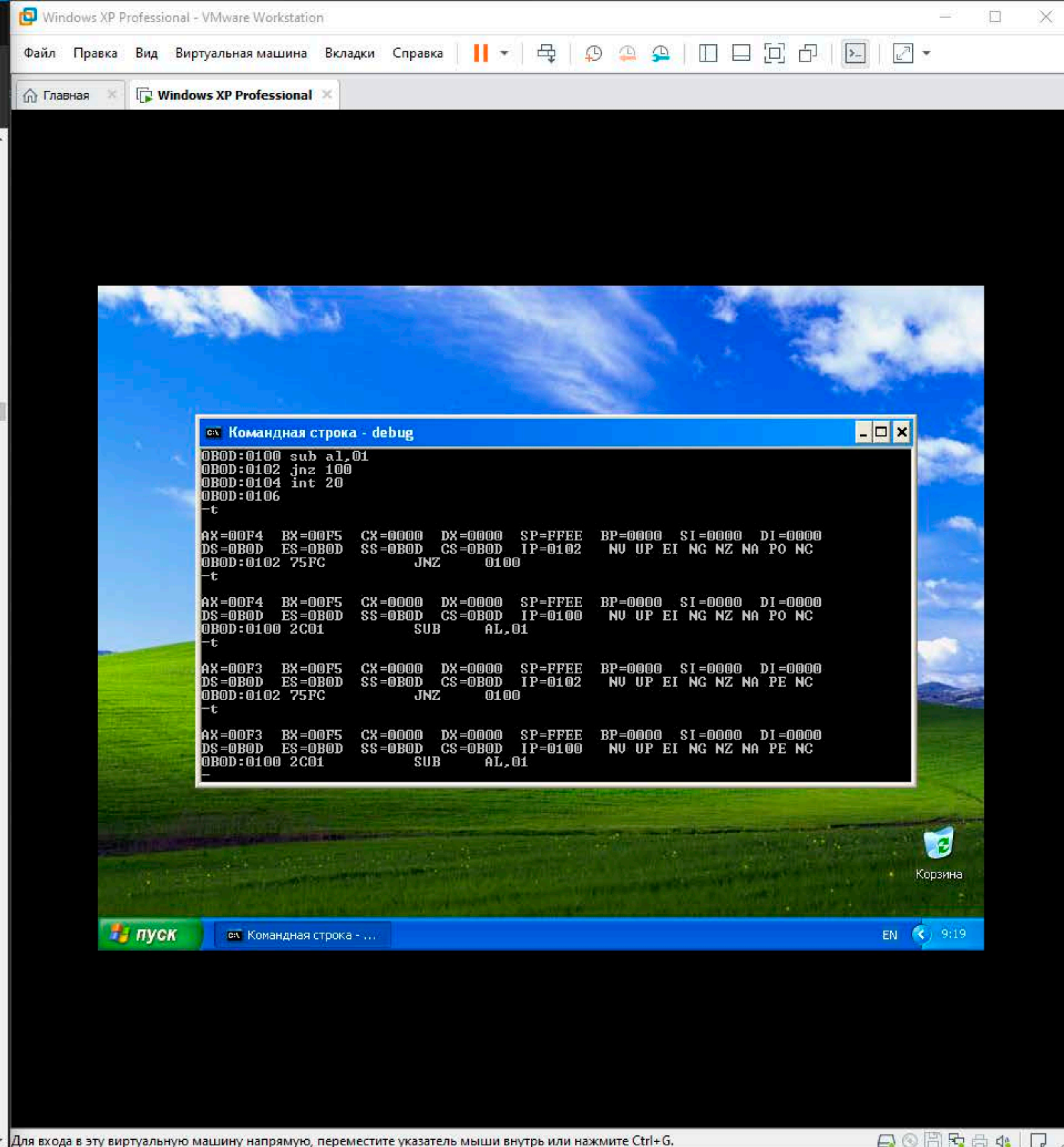
Инструкция JZ (от "Jump if Zero" – перейти, если ноль) проверяет флаг нуля, и если он установлен (ZR), то выполняется переход на новый адрес. Таким образом, если мы вслед за инструкцией SUB напомним, например, JZ 15A, то нулевой результат вычитания означает, что ЦП начнет выполнять инструкцию, находящуюся по адресу 15A, а не следующую по порядку инструкцию.

Противоположной по отношению к JZ является инструкция JNZ ("Jump if Not Zero" – перейти, если не ноль). В следующей простой программе из числа вычитается единица до тех пор, пока в результате не получится ноль:

100	SUB	AL,01
102	JNZ	100
104	INT	20

Поместите небольшое число в AL и протрассируйте программу, чтобы увидеть, как работает условное ветвление. При достижении последней инструкции введите команду G.

Инструкция условного перехода JA (от "Jump if Above" – перейти, если больше) осуществляет переход на указанный в инструкции адрес, если по результатам предыдущей инструкции флаг переноса CF сброшен (на листинге CF = NC). Флаг нуля ZF также должен быть сброшен. Данная инструкция обычно записывается сразу за инструкцией сравнения (CMP) двух беззнаковых чисел. Если первое сравниваемое число больше второго, то инструкция JA осуществляет переход.



Инструкции условного перехода

Флаги состояния устанавливаются для того, чтобы можно было менять ход выполнения программы в зависимости от текущей ситуации. Анализ флагов состояния и соответствующие переходы в программе выполняют инструкции, называемые инструкциями условного перехода.

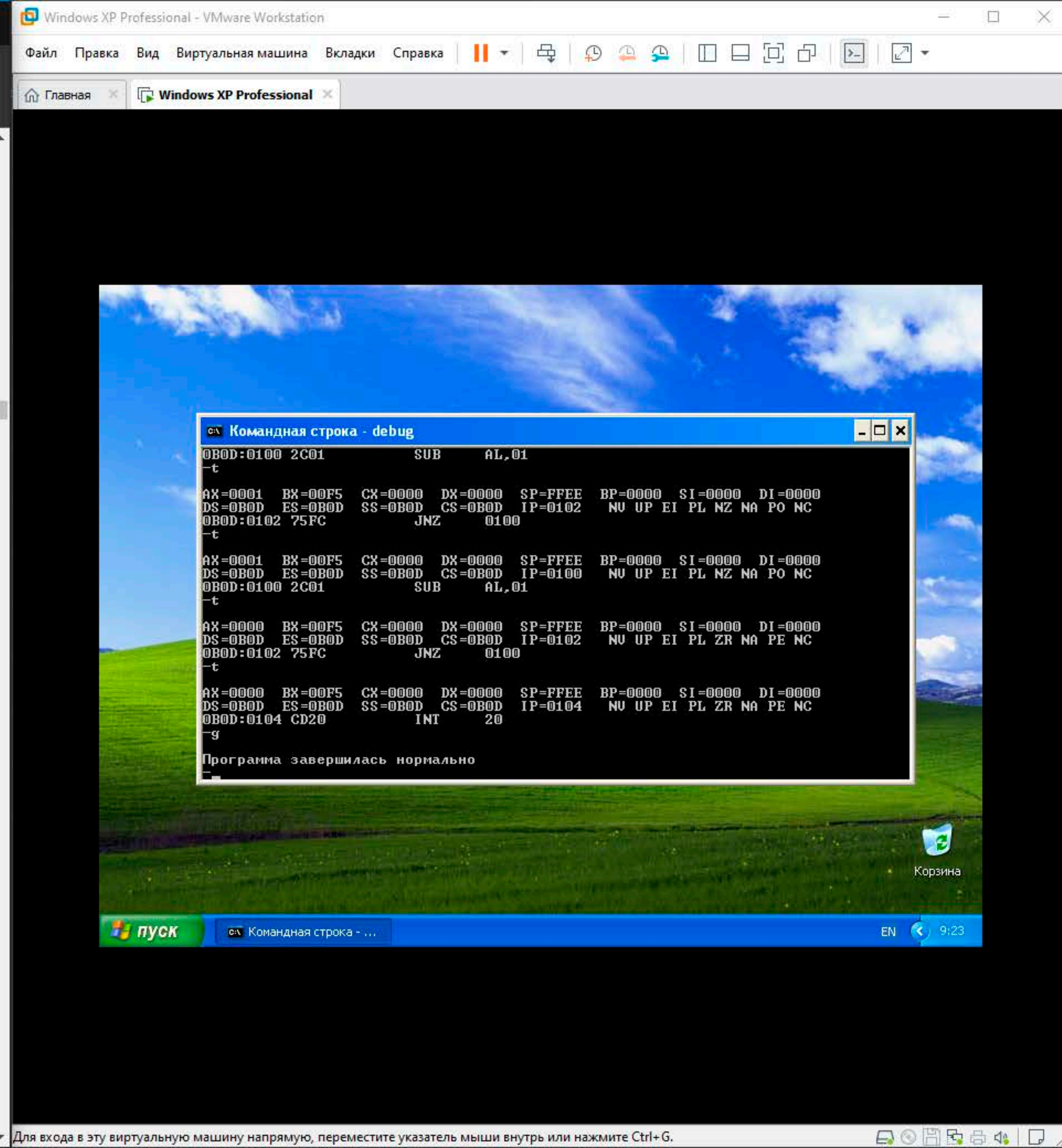
Инструкция JZ (от "Jump if Zero" – перейти, если ноль) проверяет флаг нуля, и если он установлен (ZR), то выполняется переход на новый адрес. Таким образом, если мы вслед за инструкцией SUB напишем, например, JZ 15A, то нулевой результат вычитания означает, что ЦП начнет выполнять инструкцию, находящуюся по адресу 15A, а не следующую по порядку инструкцию.

Противоположной по отношению к JZ является инструкция JNZ ("Jump if Not Zero" – перейти, если не ноль). В следующей простой программе из числа вычитается единица до тех пор, пока в результате не получится ноль:

100	SUB	AL,01
102	JNZ	100
104	INT	20

Поместите небольшое число в AL и протрассируйте программу, чтобы увидеть, как работает условное ветвление. При достижении последней инструкции введите команду G.

Инструкция условного перехода JA (от "Jump if Above" – перейти, если больше) осуществляет переход на указанный в инструкции адрес, если по результатам предыдущей инструкции флаг переноса CF сброшен (на листинге CF = NC). Флаг нуля ZF также должен быть сброшен. Данная инструкция обычно записывается сразу за инструкцией сравнения (CMP) двух беззнаковых чисел. Если первое сравниваемое число больше второго, то инструкция JA осуществляет переход.



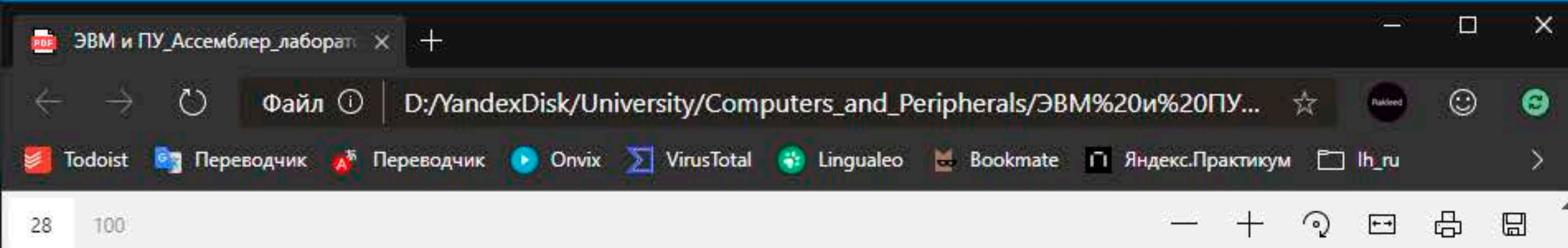


Рис. 2. Алгоритм программы вывода одной шестнадцатеричной цифры

Текст программы вывода шестнадцатеричной цифры:

100	MOV	DL, BL
102	CMP	DL, 09
105	JA	10C
107	ADD	DL, 30
10A	JMP	10F
10C	ADD	DL, 37
10F	MOV	AH, 02
111	INT	21
113	INT	20

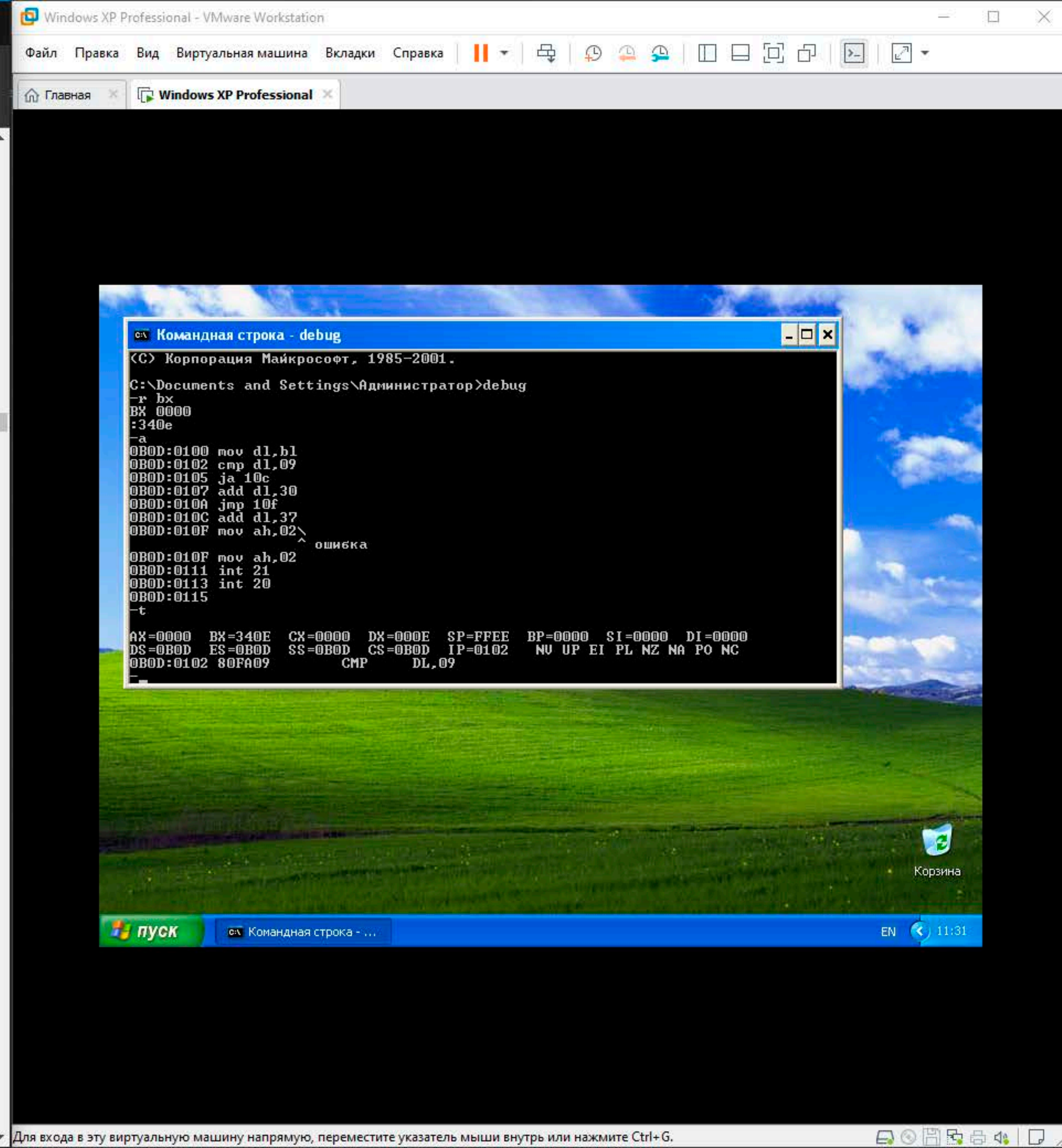
Для передачи значения шестнадцатеричной цифры на вход программы используется регистр BL. Инструкция CMP вычитает два числа ((DL)–9h), чтобы установить флаги, но она не изменяет регистр DL. Поэтому, если содержимое DL больше чем 9, инструкция JA 10C осуществляет переход к

инструкции по адресу 10C.

Запишите приведенную выше программу в ОП и протрассируйте ее, предварительно записав в BL шестнадцатеричное число, состоящее из одной цифры. Не забывайте использовать или команду G с указанием точки останова, или команду P, когда запускаете инструкции INT. Затем проверьте правильность работы программы, используя команду G, предварительно загружая в BX граничные данные: 0; 9; Ah и Fh.

Общее правило: при тестировании любой программы рекомендуется проверить граничные данные.

Вывод старшей цифры двузначного шестнадцатеричного числа



ЭВМ и ПУ_Ассемблер_лаборат...

Файл

D:/YandexDisk/University/Computers_and_Peripherals/ЭВМ%20и%20ПУ...

29100

107ADDDL, 30

10AJMP10F

10CADDDL, 37

10FMOVAH, 02

111INT21

113INT20

Для передачи значения шестнадцатеричной цифры на вход программы используется регистр BL. Инструкция CMP вычитает два числа ((DL)–9h), чтобы установить флаги, но она не изменяет регистр DL. Поэтому, если содержимое DL больше чем 9, инструкция JA 10C осуществляет переход к

30

инструкции по адресу 10C.

Запишите приведенную выше программу в ОП и протрассируйте ее, предварительно записав в BL шестнадцатеричное число, состоящее из одной цифры. Не забывайте использовать или команду G с указанием точки останова, или команду P, когда запускаете инструкции INT. Затем проверьте правильность работы программы, используя команду G, предварительно загружая в BX граничные данные: 0; 9; Ah и Fh .

Общее правило: при тестировании любой программы рекомендуется проверить граничные данные.

Вывод старшей цифры двузначного шестнадцатеричного числа

Одна шестнадцатеричная цифра занимает четыре бита (четыре бита часто называют полубайтом или тетрадой). Двузначное шестнадцатеричное число занимает восемь бит (один байт). Это может быть, например, регистр BL. При выводе числа на экран сначала выводится старшая цифра, а затем – младшая. Соответствующий укрупненный алгоритм приведен на рис. 3. При этом детальный алгоритм каждого из двух этапов “Вывод цифры на экран” приведен на рис. 2.

Windows XP Professional - VMware Workstation

ФайлПравкаВидВиртуальная машинаВкладкиСправка

ГлавнаяWindows XP Professional

Командная строка - debug

→ r bx

BX 340E

: 0

→ g

Программа завершилась нормально

→ r bx

BX 0000

: 9

→ g

Программа завершилась нормально

→ r bx

BX 0009

: a

→ g

Программа завершилась нормально

→ r bx

BX 000A

: f

→ g

Программа завершилась нормально

пуск

Командная строка - ...

EN11:34

Корзина

Для входа в эту виртуальную машину напрямую, переместите указатель мыши внутрь или нажмите Ctrl+G.

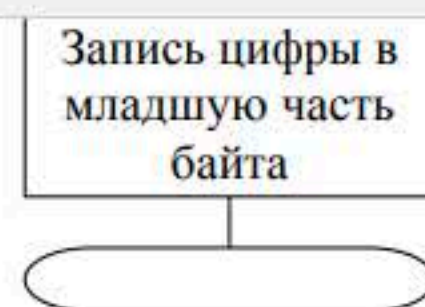
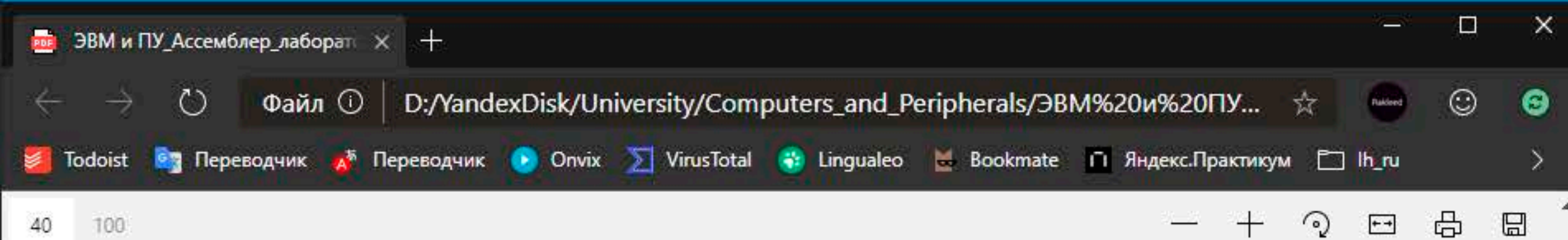


Рис. 9. Алгоритм ввода двузначного шестнадцатеричного числа

Задание

Разработайте программу, вызываемую из DOS, которая выполняет:

- 1) ввод с клавиатуры двух 4-х значных шестнадцатеричных чисел, которые записываются в качестве содержимого регистров BP и DI;
- 2) вывод на экран содержимого регистров, заполненных на шаге 1, в виде двоичных чисел;
- 3) вывод на экран содержимого регистров, заполненных на шаге 1, в виде шестнадцатеричных чисел.

Пример информации на экране:

ВВЕДИТЕ СОДЕРЖИМОЕ РЕГИСТРА BP F46B<Enter>
ВВЕДИТЕ СОДЕРЖИМОЕ РЕГИСТРА DI 5A0C<Enter>

(BP) = 1111010001101011 (DI) = 0101101000001100
(BP) = F46B (DI) = 5A0C

Примечание 1. В отличие от приводимых в описании работы программ, некоторые 8-битные регистры следует заменить на 16-битные.

Примечание 2. Рекомендуется дополнительно разработать процедуру, выполняющую ввод шестнадцатеричного числа в 16-битный регистр, процедуру вывода содержимого такого регистра в двоичном виде, а также

