

Саратовский государственный университет им. Н.Г.Чернышевского
Факультет компьютерных наук и информационных технологий

Машинно-зависимые языки программирования

Лабораторная работа №3

Выполнил
студент 211 группы
Коновалов Александр

Содержание

1	Задания	2
1.1	Задание 3.1.	2
1.2	Задание 3.2.	2
2	Программы на языке ассемблера	3
2.1	Задание 3.1.	3
2.2	Задание 3.2.	4
3	Запуск программы	6
4	Контрольные вопросы	8
4.1	Чем отличается деление на байт от деления на слово? (где должно располагаться делимое, куда попадут частное от деления и остаток от деления)	8
4.2	Каков механизм действия команды <code>str</code> ? В паре с какими командами она обычно используется?	8
4.3	На какие флаги реагируют команды условного перехода для чисел со знаком и для чисел без знака?	8
4.4	С помощью команд условного и безусловного перехода выполните программную реализацию алгоритма ветвления для определения наименьшего числа из двух заданных. Алгоритм изображен в виде блок-схемы, приведенной на рис.3.3.	10
4.5	Каков механизм работы команды организации цикла <code>LOOP</code> ?	11
4.6	Как с помощью команды сдвига можно умножить знаковое число, хранящееся в <code>AX</code> , на 2 в n -ой степени?	11
4.7	Как с помощью команды сдвига проверить содержимое регистра <code>BX</code> на четность? .	11

1 Задания

Сначала программы должны печатать фамилию, имя и номер группы студента и переходить на новую строку. Используя рассмотренное упражнение, выполните следующие задания:

1.1 Задание 3.1.

В регистре AX задано число от 0 до 65535. Выведите это число на экран. (Проверить программу для числа более 2600.)

1.2 Задание 3.2.

Используя 32-битные регистры процессора (EAX, EBX, EDX), напишите программу, выводящую на экран число 65536. Число 65536 изначально поместить в регистр EAX.

2 Программы на языке ассемблера

2.1 Задание 3.1.

```
1  .model small ;Модель памяти SMALL использует сегменты размером не более 64Кб
2  .stack 100h ;Сегмент стека размером 100h (256 байт)
3  .386 ;Разрешение трансляции команд процессора 386
4  .data ;Начало сегмента данных
5  student db 'Коновалов Александр 211', 0Dh, 0Ah, '$'
6  .code ;Начало сегмента кода
7  start: ;Точка входа в программу start
8
9  ;Предопределенная метка @data обозначает
10 ;адрес сегмента данных в момент запуска программы,
11 mov AX, @data ;который сначала загрузим в AX,
12 mov DS, AX ;а затем перенесем из AX в DS
13
14 call print_student
15
16 mov AX,65535 ;В регистре AX задано число от 0 до 65535
17 mov BX,10 ;аносим основание системы счисления, равное 10, в регистр BX
18
19 mov CX,0 ;1. Номер младшего разряда числа, равный 0, заносим в регистр
   ↪ CX.
20
21 div_num:
22 mov DX,0 ;2. Заносим 0 в регистр DX (после деления в регистр DX
   ↪ попадет остаток от деления, перед делением надо его обнулить, т.к.
   ↪ делимое у нас DX:AX). Также DX содержит старшую значимую часть
   ↪ делимого(у нас всегда 0)
23 div BX ;3. Выполняем деление DX:AX на BX: неполное частное от
   ↪ деления помещается в AX (div), а остаток (mod) - в DX.
24 push DX ;4. Сохраняем содержимое DX в стеке (нам надо
   ↪ сохранить очередную цифру в стек)
25 inc CX ;5. Увеличиваем значение счетчика в регистре CX на
   ↪ 1. (инкремент)
26 cmp AX,0 ;6.Проверка условия: если неполное частное равно 0, то
   ↪ переход на пункт 7, иначе переход на пункт 2.
27 jne div_num ;переход к метке div_num, если метка ZF = 0(числа не
   ↪ равны)
28
29 loop_: ;7. Организуем цикл loop, в котором:
30 pop DX ;а. извлекаем из стека слово данных в регистр DX.
31 shr DX, 8 ;б. выполняем логический сдвиг вправо над DX на 8 (SHR
   ↪ On1,счетчик), в результате которого получаем DH=0, DL = DH.(требуется
   ↪ только для деления на байт памяти, а у нас слово!)
32 call print ;с. вызываем процедуру print вывода цифры на экран.
33 loop loop_ ;Если CX <> 0 то осуществить переход на loop_
34
35 mov ax,4C00h ;Завершение программы
```

```

36 int 21h
37
38 print_student proc
39     mov AH,09h
40     mov DX,offset student
41     int 21h
42     ret
43 print_student endp
44
45 print proc ;Процедура print вывода на экран одной цифры
46     push AX ;1. Сохраняем содержимое AX в стеке.
47     mov AH,02h ;2. Заносим в регистр AH значение 02h.
48     add DL, 30h ;3. Выполняем преобразование в символьную форму находящейся в
    ↪ регистре DL цифры.
49     int 21h ;4. Вызываем прерывание 21h.
50     pop AX ;5. Восстанавливаем AX из стека.
51     ret ;6. Выполняем возврат из процедуры в точку вызова.
52 ret ;Возврат в программу
53 print endp ;Конец процедуры
54
55 end start ;Конец программы

```

2.2 Задание 3.2.

```

1 .model small ;Модель памяти SMALL использует сегменты размером не более 64Кб
2 .stack 100h ;Сегмент стека размером 100h (256 байт)
3 .386 ;Разрешение трансляции команд процессора 386
4 .data ;Начало сегмента данных
5 student db 'Коновалов Александр 211', 0Dh, 0Ah, '$'
6 .code ;Начало сегмента кода
7 start: ;Точка входа в программу start
8
9 ;Предопределенная метка @data обозначает
10 ;адрес сегмента данных в момент запуска программы,
11 mov AX, @data ;который сначала загрузим в AX,
12 mov DS, AX ;а затем перенесем из AX в DS
13
14 call print_student
15
16 mov EAX,65536 ;В регистре EAX задано число 65536 (можно от 0 до
    ↪ 4294967295)
17 mov EBX,10 ;аносим основание системы счисления, равное 10, в регистр BX
18
19 mov CX,0 ;1. Номер младшего разряда числа, равный 0, заносим в регистр
    ↪ CX.
20
21 div_num:

```

```

22  mov EDX,0          ;2. Заносим 0 в регистр DX (после деления в регистр DX
    ↳ попадет остаток от деления, перед делением надо его обнулить, т.к.
    ↳ делимое у нас DX:AX). Также DX содержит старшую значимую часть
    ↳ делимого(у нас всегда 0)
23  div EBX            ;3. Выполняем деление DX:AX на BX: неполное частное
    ↳ от деления помещается в AX (div), а остаток (mod) - в DX.
24  push EDX           ;4. Сохраняем содержимое DX в стеке (нам надо
    ↳ сохранить очередную цифру в стек)
25  inc CX             ;5. Увеличиваем значение счетчика в регистре CX на
    ↳ 1.(инкремент)
26  cmp EAX,0          ;6.Проверка условия: если неполное частное равно 0, то
    ↳ переход на пункт 7, иначе переход на пункт 2.
27  jne div_num        ;переход к метке div_num, если метка ZF = 0(числа не
    ↳ равны)
28
29  loop_:             ;7. Организуем цикл loop, в котором:
30  pop EDX            ;а. извлекаем из стека слово данных в регистр DX.
31  ;shr DX, 8         ;б. выполняем логический сдвиг вправо над DX на 8 (SHR
    ↳ On1,счетчик), в результате которого получаем DH=0, DL = DH.(требуется
    ↳ только для деления на байт памяти, а у нас слово!)
32  call print         ;с. вызываем процедуру print вывода цифры на экран.
33  loop loop_         ;Если CX <> 0 то осуществить переход на loop_
34
35  mov ax,4C00h ;Завершение программы
36  int 21h
37
38  print_student proc
39      mov AH,09h
40      mov DX,offset student
41      int 21h
42      ret
43  print_student endp
44
45  print proc ;Процедура print вывода на экран одной цифры
46      push EAX        ;1. Сохраняем содержимое AX в стеке.
47      mov AH,02h      ;2. Заносим в регистр AH значение 02h.
48      add DL, 30h     ;3. Выполняем преобразование в символьную форму находящейся в
    ↳ регистре DL цифры.
49      int 21h         ;4. Вызываем прерывание 21h.
50      pop EAX         ;5. Восстанавливаем AX из стека.
51      ret             ;6. Выполняем возврат из процедуры в точку вызова.
52  ret ;Возврат в программу
53  print endp ;Конец процедуры
54
55  end start ;Конец программы

```

3 Запуск программы

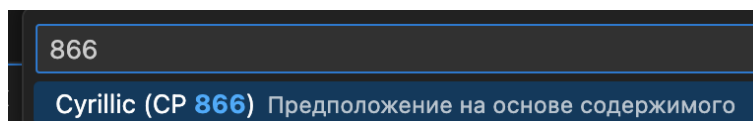


Рис. 1: Сохранение в нужной кодировке при помощи VS Code

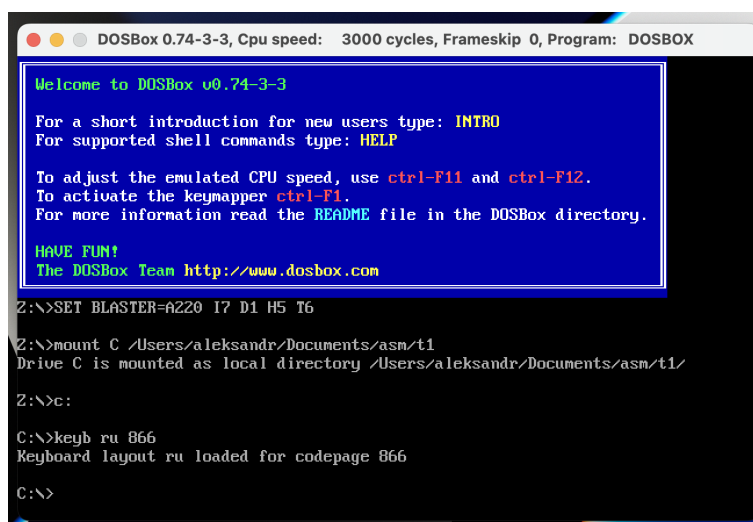
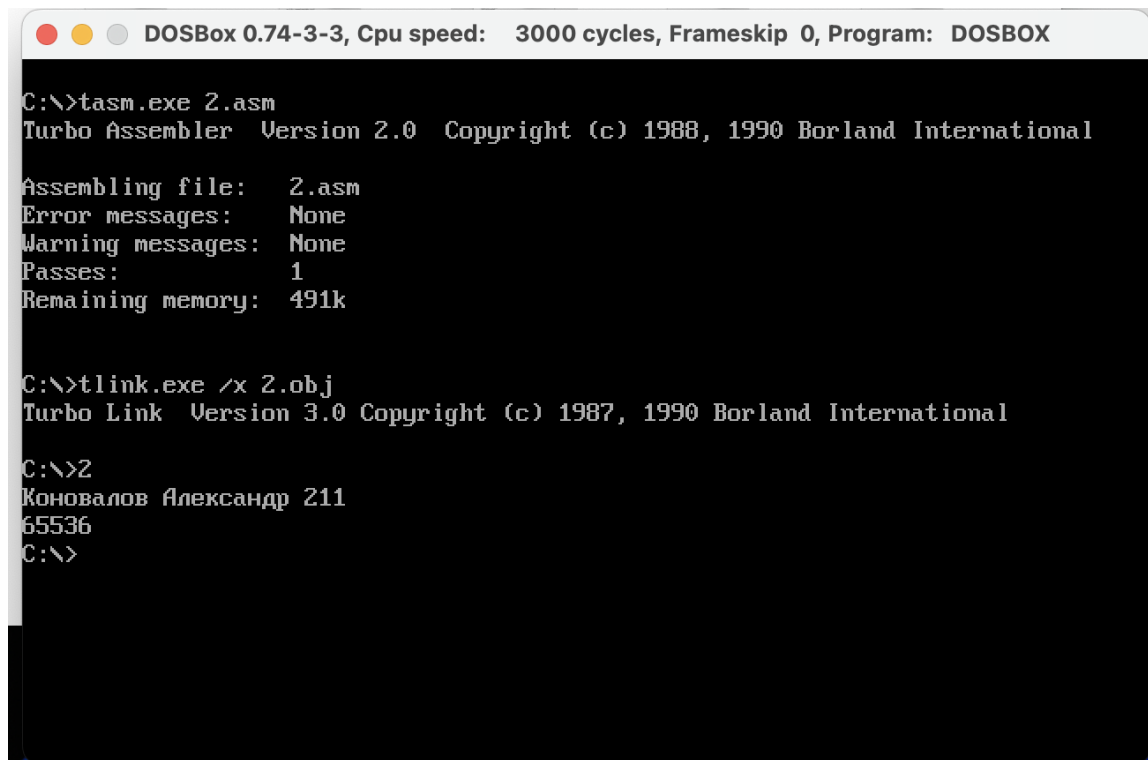


Рис. 2: Применение нужной кодировки в DosBox



Рис. 3: Запуск 1 программы



DOSBox 0.74-3-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

```
C:\>tasm.exe 2.asm
Turbo Assembler Version 2.0 Copyright (c) 1988, 1990 Borland International

Assembling file: 2.asm
Error messages:  None
Warning messages: None
Passes:         1
Remaining memory: 491k

C:\>tlink.exe /x 2.obj
Turbo Link Version 3.0 Copyright (c) 1987, 1990 Borland International

C:\>2
Коновалов Александр 211
65536
C:\>
```

Рис. 4: Запуск 2 программы

4 Контрольные вопросы

4.1 Чем отличается деление на байт от деления на слово? (где должно располагаться делимое, куда попадут частное от деления и остаток от деления)

Один из операндов (делимое) всегда в два раза длиннее операнда делителя. Ниже приведены схемы, иллюстрирующие команды деления.

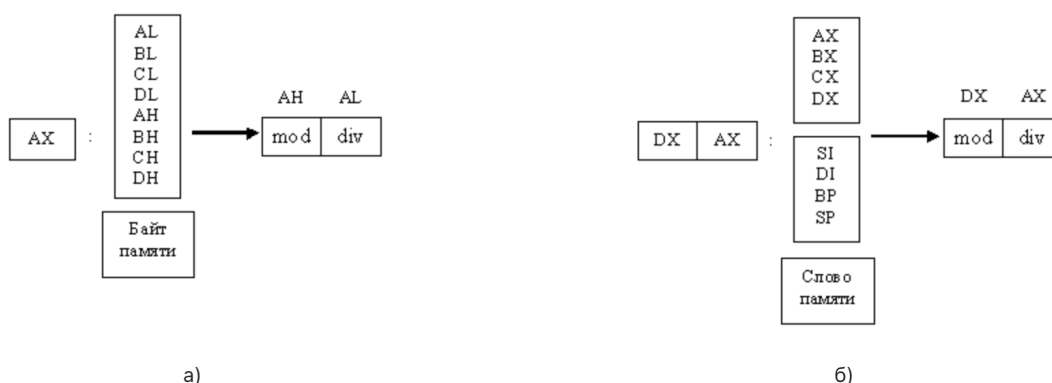


Рис. 5: Схема выполнения операции деления на а - байт; б - слово

Байтовая команда делит 16-битовое делимое на 8-битовый делитель. Делимое находится в регистре AX. В результате деления получается два числа: частное помещается в регистр AL, а остаток – в AH.

Команда, работающая со словами, делит 32-битовое делимое на 16-битовый делитель. Делимое находится в паре регистров DX:AX, причем регистр DX содержит старшую значимую часть, а регистр AX – младшую. Команда деления помещает частное в регистр AX, а остаток в DX.

4.2 Каков механизм действия команды cmp? В паре с какими командами она обычно используется?

Команда сравнения CMP сравнивает два числа, вычитая второе из первого. Инструкция CMP не сохраняет результат, а лишь устанавливает в соответствии с результатом флаги состояния. Основное назначение команды CMP – это организация ветвлений (условных переходов) в ассемблерных программах. Используется, к примеру, с командами условного перехода (Jcc).

4.3 На какие флаги реагируют команды условного перехода для чисел со знаком и для чисел без знака?

Первая группа команд Jcc (кроме JCXZ/JECXZ) проверяет текущее состояние регистра флагов (не изменяя его) и в случае соблюдения условия осуществляет переход на смещение, указанное в качестве операнда. Флаги, проверяемые командой, кодируются в ее мнемонике, например: JC – переход, если установлен CF. Сокращения «L» (less – меньше) и «G» (greater – больше) применяются для целых со знаком, а «A» (above – над) и «B» (below – под) для целых без знака. Ниже в таблице показаны команды условного перехода и проверяемые ими флаги.

Мнемоника	Флаги					Комментарии
	OF	CF	ZF	PF	SF	
Проверка флагов						
JE/JZ	X	X	1	X	X	
JP/JPE	X	X	X	1	X	
JO	1	X	X	X	X	
JS	X	X	X	X	1	
JNE/JNZ	X	X	0	X	X	
JNP/JPO	X	X	X	0	X	
JNO	0	X	X	X	X	
JNS	X	X	X	X	0	
Арифметика со знаком						
JL/JNGE	a	X	X	X	b	a не равно b (SF<>OF)
JLE/JNG	a	X	1	X	b	ZF или a не равно b
JNL/JGE	a	X	X	X	b	a равно b
JNLE/JG	a	X	0	X	b	не ZF и (a равно b)
Арифметика без знака						
JB/JNAE/JS	X	1	X	X	X	
JBE/JNA	X	1	1	X	X	CF или ZF
JNB/JAE	X	0	X	X	X	
JNBE/JA	X	0	0	X	X	не CF и не ZF

Рис. 6: Команды условного перехода.

буква X в любой позиции означает, что команда не проверяет флаг. Цифра 0 означает, что флаг должен быть сброшен, а цифра 1 означает, что флаг должен быть установлен, чтобы условие было выполнено (переход произошел).

Команды условного перехода можно разделить на три подгруппы:

1. Непосредственно проверяющие один из флагов на равенство 0 или 1.
2. Арифметические сравнения со знаком. Существуют 4 условия, которые могут быть проверены: меньше (JL), меньше или равно (JLE), больше (JG), больше или равно (JGE). Эти команды проверяют одновременно три флага: знака, переполнения и нуля.
3. Арифметические без знака. Здесь также существует 4 возможных соотношения между операндами: меньше (JB), меньше или равно (JBE), больше (JA), больше или равно (JAE). Учитываются только два флага. Флаг переноса показывает какое из двух чисел больше. Флаг нуля определяет равенство.

4.4 С помощью команд условного и безусловного перехода выполните программную реализацию алгоритма ветвления для определения наименьшего числа из двух заданных. Алгоритм изображен в виде блок-схемы, приведенной на рис.3.3.

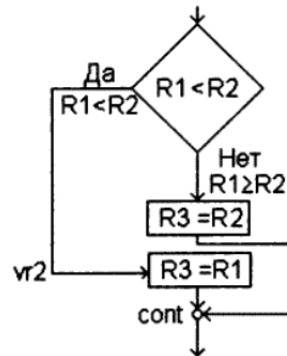


Рис.3.3. Организация ветвления на машинном уровне:

R1 - первое число хранится в регистре AX;

R2 - второе число хранится в регистре BX;

R3 - результат заносится в регистр DX;

vr2, cont - метки команд.

```

1  .model small ;Модель памяти SMALL использует сегменты размером не более 64Кб
2  .stack 100h ;Сегмент стека размером 100h (256 байт)
3  .386 ;Разрешение трансляции команд процессора 386
4  .data ;Начало сегмента данных
5  .code ;Начало сегмента кода
6  start: ;Точка входа в программу start
7
8  ;Предопределенная метка @data обозначает
9  ;адрес сегмента данных в момент запуска программы,
10 mov AX, @data ;который сначала загрузим в AX,
11 mov DS, AX ;а затем перенесем из AX в DS
12
13 mov AX,9 ;В регистре AX задано число от 0 до 65535
14 mov BX,2 ;аносим основание системы счисления, равное 10, в регистр BX
15
16 cmp AX,BX
17 JL vr2
18 mov DX, BX
19 jmp cont ; переход на конец программы
20
21 vr2:
22 mov DX, AX
23 jmp cont ; переход на конец программы
24 cont:
25 mov ax,4C00h ;Завершение программы
26 int 21h
27 end start ;Конец программы
  
```

4.5 Каков механизм работы команды организации цикла LOOP?

Все команды цикла используют регистр CX в качестве счетчика цикла. Простейшая из них – команда LOOP. Она в конце каждой итерации уменьшает содержимое CX на 1 и передает управление на метку (указанную в команде), если содержимое CX не равно 0. Если вычитание 1 из CX привело к нулевому результату, выполняется следующая команда.

Команда LOOPNE (цикл пока не равно) выходит из цикла, если не установлен флаг нуля или если в регистре CX получился 0. Команда LOOPE (цикл пока равно) выполняет обратную к описанной проверку флага нуля: цикл здесь завершается, если регистр CX достиг 0 или если установлен флаг 0.

Ниже приведен фрагмент программы, вычисляющей в цикле сумму цифр от 1 до 9, используя команду организации цикла LOOP.

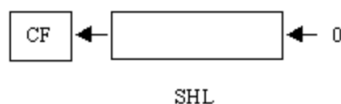
```
MOV AH,1 ;Занести в AH первую цифру 1
MOV AL,0 ;Подготовить регистр результата
MOV CX,9 ;Количество суммируемых цифр
met:
ADD AL,AH ;Прибавить к результату очередную цифру из AH
INC AH ;Увеличить AH на единицу
LOOP met ;Если CX <>0 то осуществить переход на met
```

4.6 Как с помощью команды сдвига можно умножить знаковое число, хранящееся в AX, на 2 в n-ой степени?

Для умножения числа на 2 в n-степени, нужно выполнить сдвиг содержимого регистра влево на n битов.

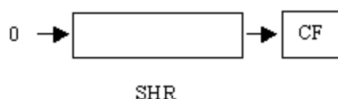
В Ассемблере есть команда логического сдвига влево shl. С её помощью можно выполнить данное умножение

```
SHL AX, n ; Сдвигаем AX влево на n бит = умножаем на 2 в n степени
```



4.7 Как с помощью команды сдвига проверить содержимое регистра BX на четность?

Выполнить логический сдвиг вправо на 1 бит(shr в Ассемблере), после сдвига младший разряд числа перейдет в флаг CF, проверяем его с помощью команды условного перехода



```
SHR BX, 1 ; Сдвиг BX вправо на 1 бит
JC not_even ; Если CF = 1 (нечетное), переходим по метке odd
```

; Здесь код для четного числа