

Отчёт по лабораторной работе №6

Дисциплина: Архитектура компьютера

Луангсуваннавонг Сайпхачан

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
4.1	Символьные и численные данные в NASM	8
4.2	Выполнение арифметических операций в NASM	13
4.2.1	ответы на вопросы	19
5	Выполнение заданий для самостоятельной работы	21
6	Выводы	25
7	Ответы на вопросы для самопроверки	26
8	Список литературы	28

Список иллюстраций

4.1	Создание и перемещение в директории	8
4.2	Создание файла	8
4.3	Копирование файла	9
4.4	Редактирование файла	9
4.5	Запуск исполняемого файла	10
4.6	Редактирование файла	10
4.7	Запуск исполняемого файла	11
4.8	Создание файла	11
4.9	Редактирование файла	11
4.10	Запуск исполняемого файла	12
4.11	Редактирование файла	12
4.12	Запуск исполняемого файла	12
4.13	Редактирование файла	13
4.14	Запуск исполняемого файла	13
4.15	Создание файла	14
4.16	Редактирование файла	14
4.17	Запуск исполняемого файла	15
4.18	Редактирование файла	16
4.19	Запуск исполняемого файла	17
4.20	Создание файла	17
4.21	Редактирование файла	18
4.22	Запуск исполняемого файла	19
5.1	Создание файла	21
5.2	Редактирование файла	22
5.3	Запуск исполняемого файла	23

1 Цель работы

Целью данной лабораторной работы является освоение арифметических инструкций языка ассемблера NASM.

2 Задание

1. Символьные и численные данные в NASM
2. Выполнение арифметических операций в NASM
3. Выполнение заданий для самостоятельной работы

3 Теоретическое введение

Большинство инструкций на языке ассемблера требуют обработки операндов. Адрес операнда предоставляет место, где хранятся данные, подлежащие обработке. Это могут быть данные хранящиеся в регистре или в ячейке памяти. Далее рассмотрены все существующие способы задания адреса хранения операндов – способы адресации. Существует три основных способа адресации:

- Регистровая адресация - операнды хранятся в регистрах и в команде используются имена этих регистров, например: `mov ax, bx`.
- Непосредственная адресация - значение операнда задается непосредственно в команде, Например: `mov ax, 2`.
- Адресация памяти - операнд задает адрес в памяти. В команде указывается символическое обозначение ячейки памяти, над содержимым которой требуется выполнить операцию.

Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. ASCII – сокращение от American Standard Code for Information Interchange (Американский стандартный код для обмена информацией). Согласно стандарту ASCII каждый символ кодируется одним байтом.

Расширенная таблица ASCII состоит из двух частей. Первая (символы с кодами 0-127) является универсальной (см. Приложение.), а вторая (коды 128-255) предназначена для специальных символов и букв национальных алфавитов и на компьютерах разных типов может меняться.

Среди инструкций NASM нет такой, которая выводит числа (не в символьном

виде). По этому, например, чтобы вывести число, надо предварительно преобразовать его цифры в ASCII-коды этих цифр и выводить на экран эти коды, а не само число. Если же выводить число на экран непосредственно, то экран воспримет его не как число, а как последовательность ASCII-символов – каждый байт числа будет воспринят как один ASCII-символ – и выведет на экран эти символы.

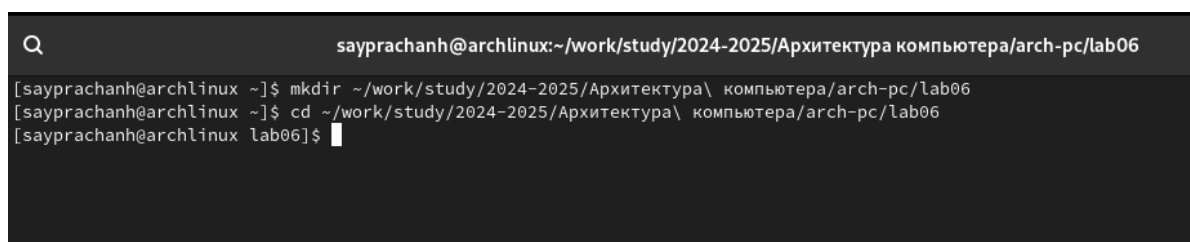
Аналогичная ситуация происходит и при вводе данных с клавиатуры. Введенные данные будут представлять собой символы, что сделает невозможным получение корректного результата при выполнении над ними арифметических операций.

Для решения этой проблемы не обходимо проводить преобразование ASCII символов в числа и обратно.

4 Выполнение лабораторной работы

4.1 Символьные и численные данные в NASM

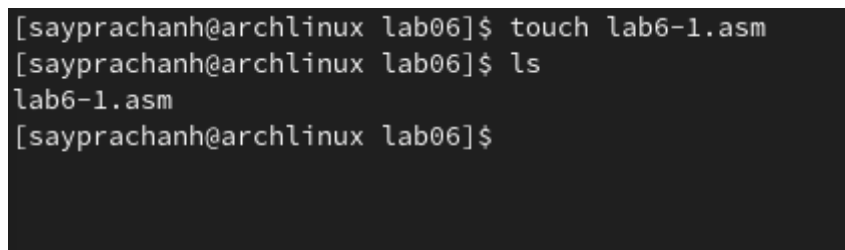
С помощью команды `mkdir` я создаю новую директорию, в которой буду создавать файлы с программами для лабораторной работы № 6. (Рис. 4.1). Я перехожу в созданную директорию с помощью команды `cd`



```
sayprachanh@archlinux:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06
[sayprachanh@archlinux ~]$ mkdir ~/work/study/2024-2025/Архитектура\ компьютера/arch-pc/lab06
[sayprachanh@archlinux ~]$ cd ~/work/study/2024-2025/Архитектура\ компьютера/arch-pc/lab06
[sayprachanh@archlinux lab06]$
```

Рис. 4.1: Создание и перемещение в директории

С помощью команды `touch`, я создаю файл `lab6-1.asm` (Рис. 4.2)



```
[sayprachanh@archlinux lab06]$ touch lab6-1.asm
[sayprachanh@archlinux lab06]$ ls
lab6-1.asm
[sayprachanh@archlinux lab06]$
```

Рис. 4.2: Создание файла

Я копирую файл `in_out.asm` из каталога загрузок, так как он будет использоваться в других программах (Рис. 4.3)


```
[sayprachanh@archlinux lab06]$ cp ~/Downloads/in_out.asm in_out.asm
[sayprachanh@archlinux lab06]$ ls
in_out.asm  lab6-1.asm
[sayprachanh@archlinux lab06]$
```

Рис. 4.3: Копирование файла

Я открываю созданный файл lab6-1.asm, затем вставляю программу для вывода значения регистра eax(Рис. 4.4)

```
sayprachanh@archlinux:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06
16 %include 'in_out.asm'
15
14 SECTION .bss
13 buf1:RESB 80
12
11 SECTION .text
10 GLOBAL _start
9 _start:
8 mov eax,'6'
7 mov ebx,'4'
6 add eax,ebx
5 mov [buf1],eax
4 mov eax,buf1
3 call sprintLF
2
1 call quit
~
```

Рис. 4.4: Редактирование файла

Я создаю новый исполняемый файл программы и запускаю его(Рис. 4.5) Программа выведет символ “j”, потому что символ “j” в системе ASCII имеет код 106, который в двоичной системе представляется как 01101010. Программа выводит символ в соответствии с кодировкой ASCII.

```
[sayprachanh@archlinux lab06]$ nasm -f elf lab6-1.asm
[sayprachanh@archlinux lab06]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[sayprachanh@archlinux lab06]$ ./lab6-1
j
[sayprachanh@archlinux lab06]$
```

Рис. 4.5: Запуск исполняемого файла

Я снова захожу в программный файл и меняю символы “6” и “4” на цифры 6 и 4(Рис. 4.6)

```
1  %include 'in_out.asm'
2
3  SECTION .bss
4  buf1:RESB 80
5
6  SECTION .text
7  GLOBAL _start
8  _start:
9  mov eax, 6
10 mov ebx, 4
11 add eax,ebx
12 mov [buf1],eax
13 mov eax,buf1
14 call sprintf
15
16 call quit
~
```

Рис. 4.6: Редактирование файла

Я создаю новый исполняемый файл программы и запускаю его (Рис. 4.7) В результате выводится символ с кодом 10 — это символ перевода строки (Line Feed). Он не отображается на экране как видимый символ, но при его выводе текст перемещается на новую строку.

```
[sayprachanh@archlinux lab06]$ nasm -f elf lab6-1.asm
[sayprachanh@archlinux lab06]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[sayprachanh@archlinux lab06]$ ./lab6-1

[sayprachanh@archlinux lab06]$
```

Рис. 4.7: Запуск исполняемого файла

С помощью команды touch, я создаю файл lab6-2.asm(Рис. 4.8)

```
[sayprachanh@archlinux lab06]$ touch lab6-2.asm
[sayprachanh@archlinux lab06]$
```

Рис. 4.8: Создание файла

Я вхожу в программу, которая выводит значение регистра eax(Рис. 4.9)

```
11  %include 'in_out.asm'
10
9   SECTION .text
8   GLOBAL _start
7   _start:
6
5   mov eax,'6'
4   mov ebx,'4'
3   add eax,ebx
2   call iprintLF
1
12  call quit
```

Рис. 4.9: Редактирование файла

Я создаю новый исполняемый файл программы и запускаю его.(Рис. 4.10) Теперь программа выводит число 106 в качестве результата. Потому что эта про-

грамма позволяет выводить число (десятичное в ASCII), а не символ.

```
[sayprachanh@archlinux lab06]$ nasm -f elf lab6-2.asm
[sayprachanh@archlinux lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[sayprachanh@archlinux lab06]$ ./lab6-2
106
[sayprachanh@archlinux lab06]$
```

Рис. 4.10: Запуск исполняемого файла

Я снова захожу в программный файл lab6-2.asm и меняю символы “6” и “4” на цифры 6 и 4(Рис. 4.11)

```
7 %include 'in_out.asm'
6
5 SECTION .text
4 GLOBAL _start
3 _start:
2
1 mov eax,6
8 mov ebx,4
1 add eax,ebx
2 call iprintLF
3
4 call quit
~
```

Рис. 4.11: Редактирование файла

Теперь программа добавляет не код, соответствующий символам в системе ASCII, а сами числа, поэтому вывод равен 10(Рис. 4.12)

```
[sayprachanh@archlinux lab06]$ nasm -f elf lab6-2.asm
[sayprachanh@archlinux lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[sayprachanh@archlinux lab06]$ ./lab6-2
10
[sayprachanh@archlinux lab06]$
```

Рис. 4.12: Запуск исполняемого файла

Я меняю программную функцию `iprintLF` на `iprint` в тексте программы (Рис. 4.13)

```
%include 'in_out.asm'

SECTION .text
GLOBAL _start
_start:

    mov eax,6
    mov ebx,4
    add eax,ebx
    call iprint

    call quit
```

Рис. 4.13: Редактирование файла

Я создаю и запускаю новый исполняемый файл (Рис. 4.14). Вывод не изменился, потому что символ новой строки не появился, когда программа использовала функцию `iprintLF`. В отличие от `iprintLF`, функция `iprint` не добавляет новый символ новой строки в вывод.

```
[sayprachanh@archlinux lab06]$ helix lab6-2.asm
[sayprachanh@archlinux lab06]$ nasm -f elf lab6-2.asm
[sayprachanh@archlinux lab06]$ ld -m elf_i386 -o lab6.2 lab6-2.o
[sayprachanh@archlinux lab06]$ ./lab6-2
10
[sayprachanh@archlinux lab06]$
```

Рис. 4.14: Запуск исполняемого файла

4.2 Выполнение арифметических операций в NASM

С помощью команды `touch`, я создаю файл `lab6-2.asm` (Рис. 4.15)

```
[sayprachanh@archlinux lab06]$ touch lab6-3.asm
[sayprachanh@archlinux lab06]$
```

Рис. 4.15: Создание файла

Затем я ввожу в созданный файл программу, которая вычисляет значение выражения $f(x) = (5 * 2 + 3)/3$ (Рис. 4.16)

```
35 %include 'in_out.asm' ; подключение внешнего файла
34
33 SECTION .data
32
31 div: DB 'Результат: ',0
30 rem: DB 'Остаток от деления: ',0
29
28 SECTION .text
27 GLOBAL _start
26 _start:
25
24
23 ;---- Вычисление выражения
22 mov eax,5 ; EAX=5
21 mov ebx,2 ; EBX=2
20 mul ebx ; EAX=EAX*EBX
19 add eax,3 ; EAX=EAX+3
18 xor edx,edx ; обнуляем EDX для корректной работы div
17 mov ebx,3 ; EBX=3
16 div ebx ; EAX=EAX/3, EDX=остаток от деления\
15
14 mov edi,eax ; запись результата вычисления в 'edi'
13
12 ;---- Вывод результата на экран
11 mov eax,div; вызов подпрограммы печати
10 call sprint; сообщения 'Результат: '
9 mov eax,edi; вызов подпрограммы печати значения
8 call iprintLF; из 'edi' в виде символов
7
6 mov eax,rem; вызов подпрограммы печати
5 call sprint ; сообщения 'Остаток от деления: '
4 mov eax,edx; вызов подпрограммы печати значения
3 call iprintLF; из 'edx' (остаток) в виде символов
2
1 call quit; вызов подпрограммы завершения
36
```

Рис. 4.16: Редактирование файла

Я создаю исполняемый файл и запускаю его (Рис. 4.17)

```
[sayprachanh@archlinux lab06]$ nasm -f elf lab6-3.asm
[sayprachanh@archlinux lab06]$ ld -m elf_i386 -o lab6-3 lab6-3.o
[sayprachanh@archlinux lab06]$ ./lab6-3
Результат: 4
Остаток от деления: 1
[sayprachanh@archlinux lab06]$
```

Рис. 4.17: Запуск исполняемого файла

Я изменяю программу, чтобы она вычисляла значение выражения $f(x) = (4 * 6 + 2)/5$ (Рис. 4.18)

```

19 %include 'in_out.asm' ; подключение внешнего файла
18
17 SECTION .data
16
15     div: DB 'Результат: ',0
14     rem: DB 'Остаток от деления: ',0
13
12     SECTION .text
11     GLOBAL _start
10     _start:
9
8
7     ;---- Вычисление выражения
6     mov eax,4 ; EAX=5
5     mov ebx,6; EBX=2
4     mul ebx ;EAX=EAX*EBX
3     add eax,2 ; EAX=EAX+3
2     xor edx,edx ; обнуляем EDX для корректной работы div
1     mov ebx,5 ; EBX=3
20    div ebx ; EAX=EAX/3, EDX=остаток от деления\
1
2     mov edi,eax ; запись результата вычисления в 'edi'
3
4     ;---- Вывод результата на экран
5     mov eax,div; вызов подпрограммы печати
6     call sprint; сообщения 'Результат: '
7     mov eax,edi; вызов подпрограммы печати значения
8     call iprintLF; из 'edi' в виде символов
9
10    mov eax,rem; вызов подпрограммы печати
11    call sprint ; сообщения 'Остаток от деления: '
12    mov eax,edx; вызов подпрограммы печати значения
13    call iprintLF; из 'edx' (остаток) в виде символов
14
15    call quit; вызов подпрограммы завершения
16
~

```

Рис. 4.18: Редактирование файла

Затем я создаю и запускаю исполняемый файл (Рис. 4.19). Чтобы проверить правильность работы программы, я самостоятельно вычисляю значение выражения, программа работает правильно


```
[sayprachanh@archlinux lab06]$ nasm -f elf lab6-3.asm
[sayprachanh@archlinux lab06]$ ld -m elf_i386 -o lab6-3 lab6-3.o
[sayprachanh@archlinux lab06]$ ./lab6-3
Результат: 5
Остаток от деления: 1
[sayprachanh@archlinux lab06]$
```

Рис. 4.19: Запуск исполняемого файла

С помощью команды `touch`, я создаю файл `variant.asm`(Рис. 4.20)

```
[sayprachanh@archlinux lab06]$ touch variant.asm
[sayprachanh@archlinux lab06]$
```

Рис. 4.20: Создание файла

Затем я ввожу в созданный файл программу, которая вычисляет вариант задания на основе номера студенческого билета(Рис. 4.21)

```

33 %include 'in_out.asm'
32 SECTION .data
31 msg: DB 'Введите № студенческого билета: ',0
30 rem: DB 'Ваш вариант: ',0
29
28 SECTION .bss
27 x:
26 RESB 80
25
24 SECTION .text
23 GLOBAL _start
22 _start:
21
20 mov eax, msg
19 call sprintLF
18
17 mov ecx, x
16 mov edx, 80
15 call sread
14
13 mov eax,x; вызов подпрограммы преобразования
12 call atoi; ASCII кода в число, `eax=x`
11
10 xor edx,edx
9 mov ebx,20
8 div ebx
7 inc edx
6
5 mov eax,rem
4 call sprint
3 mov eax,edx
2 call iprintLF
1
34 call quit

```

Рис. 4.21: Редактирование файла

Я создаю и запускаю исполняемый файл (Рис. 4.22). Я ввожу номер своего студенческого билета, и программа выдает, что мой вариант равен 13

```
[sayprachanh@archlinux lab06]$ nasm -f elf variant.asm
[sayprachanh@archlinux lab06]$ ld -m elf_i386 -o variant variant.o
[sayprachanh@archlinux lab06]$ ./variant
Введите № студенческого билета:
1032249112
Ваш вариант: 13
[sayprachanh@archlinux lab06]$
```

Рис. 4.22: Запуск исполняемого файла

4.2.1 ответы на вопросы

1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения 'Ваш вариант:'?

```
mov eax, rem
call sprint
```

2. Для чего используются следующие инструкции?

```
mov ecx, x
mov edx, 80
call sread
```

`mov ecx, x` — записывает в регистр ECX значение переменной `x`, которое может быть адресом строки или её длиной.

`mov edx, 80` — записывает в регистр EDX максимальную длину вводимой строки.

`call sread` — вызывает подпрограмму `sread`, которая обеспечивает ввод строки с клавиатуры.

3. Для чего используется инструкция "`call atoi`"?

`call atoi` используется для вызова подпрограммы из внешнего файла, которая преобразует строку (в формате ASCII) в целое число и записывает результат в регистр EAX.

4. Какие строки листинга 6.4 отвечают за вычисления варианта?

```
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,20 ; EBX = 20
div ebx ; EAX=EAX/3, EDX=остаток от деления
inc edx ; Увеличиваем EDX на 1 (edx = edx + 1)
```

5. В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”?

При выполнении инструкции “div ebx” остаток от деления записывается в регистр edx

6. Для чего используется инструкция “inc edx”?

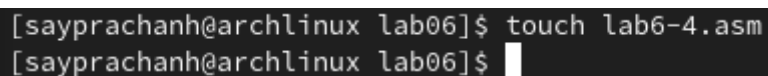
Инструкция inc edx используется для увеличения значения в регистре EDX на 1

7. Какие строки листинга 6.4 отвечают за вывод на экран результата вычислений?

```
mov eax, edx
call iprintLF
```

5 Выполнение заданий для самостоятельной работы

Используя touch команду, я создаю файл lab6-4.asm(Рис. 5.1)



```
[sayprachanh@archlinux lab06]$ touch lab6-4.asm
[sayprachanh@archlinux lab06]$
```

Рис. 5.1: Создание файла

Я открываю созданный файл для редактирования и, поскольку мой вариант равен 13, ввожу в него программу для вычисления значения выражения $(8x + 6) * 10$.(Рис. 5.2)

```

32 %include 'in_out.asm'
31 SECTION .data
30 msg: DB 'Введите значение x: ',0
29 rem: DB 'Результат: ',0
28 SECTION .bss
27 x: RESB 80
26 SECTION .text
25 GLOBAL _start
24 _start:
23 ;---- Вычисление выражения
22 mov eax, msg
21 call sprint
20
19 mov ecx, x
18 mov edx, 80
17 call sread
16 mov eax, x
15 call atoi
14 mov ebx, 8
13 mul ebx
12
11 add eax, 6
10
9 mov ebx, 10
8 mul ebx
7
6 mov edi, eax
5 ;---- Вывод результата
4 mov eax, rem
3 call sprint
2 mov eax, edi
1 call iprintLF
33 call quit
~

```

Рис. 5.2: Редактирование файла

Я создаю и запускаю исполняемый файл. (Рис. 5.3) Я ввожу значение 1, и она выдает 140, затем я снова запускаю исполняемый файл, чтобы проверить работу программы, я ввожу значение 4, и она выдает 380. Программа работает верно

```

[sayprachanh@archlinux lab06]$ nasm -f elf lab6-4.asm
[sayprachanh@archlinux lab06]$ ld -m elf_i386 -o lab6-4 lab6-4.o
[sayprachanh@archlinux lab06]$ ./lab6-4
Введите значение x: 1
Результат: 140
[sayprachanh@archlinux lab06]$ ./lab6-4
Введите значение x: 4
Результат: 380
[sayprachanh@archlinux lab06]$ █

```

Рис. 5.3: Запуск исполняемого файла

Программа для вычисления значения выражения $(8x + 6) \cdot 10$

```

%include 'in_out.asm' ; подключение внешнего файла
SECTION .data ; секция инициированных данных
msg: DB 'Введите значение x: ',0
rem: DB 'Результат: ',0
SECTION .bss
x: RESB 80
SECTION .text ; Код программы
GLOBAL _start ; Начало программы
_start:
;---- Вычисление выражения
mov eax, msg
call sprint

mov ecx, x
mov edx, 80
call sread

mov eax,x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`
mov ebx,8 ; EBX = 8

```

```
mul ebx ; EAX = EAX*EBX
```

```
add eax,6 ; EAX = EAX + 6
```

```
mov ebx,10 ; EBX = 10
```

```
mul ebx ; EAX = EAX*EBX
```

```
mov edi,eax ; запись результата вычисления в 'edi'
```

```
;---- Вывод результата
```

```
mov eax,rem ; вызов подпрограммы печати
```

```
call sprint ; сообщения 'Результат: '
```

```
mov eax,edi ; вызов подпрограммы печати значения
```

```
call iprintLF ; из 'edi' в виде символов
```

```
call quit ; вызов подпрограммы завершения
```


6 Выводы

При выполнении данной лабораторной работы, я освоил арифметические инструкции языка ассемблера NASM.

7 Ответы на вопросы для самопроверки

1. Какой синтаксис команды сложения чисел?

`add <регистр>, <операнд>`

Складывает два операнда и сохраняет результат в первый операнд. Пример:
`add eax, ebx` ; складывает значение `ebx` с `eax` и сохраняет результат в `eax`

2. Какая команда выполняет умножение без знака?

Команда `mul` умножает содержимое регистра `eax` на операнд. Результат сохраняется в два регистра: `edx` и `eax`. Пример:

`mul ebx` ; умножает `eax` на `ebx`, результат в `edx:eax`

3. Какой синтаксис команды деления чисел без знака?

Команда `div` делит число в регистрах `edx:eax` на операнд.

Делимое: хранится в регистрах `edx:eax`.

Частное: сохраняется в `eax`.

Остаток: сохраняется в `edx`.

Пример:

`div ebx` ; делит `edx:eax` на `ebx`, результат в `eax`, остаток в `edx`

4. Куда помещается результат при умножении двух байтовых операндов?

При умножении двух байтовых операндов результат сохраняется в регистрах AX (младшие 16 бит) и DX (старшие 8 бит).

`mul bl` ; умножает `al` на `bl`, результат в `ax` (младший 16 байт) и `dx` (старший 8 байт)

5. Перечислите арифметические команды с целочисленными операндами и дайте их назначение.

`add` — сложение,

`sub` — вычитание,

`mul` — умножение без знака (результат в AX или DX:AX),

`imul` — умножение с знаком (результат в AX или DX:AX),

`div` — деление без знака (частное в AL или AX, остаток в AH или DX),

`idiv` — деление с знаком (частное в AL или AX, остаток в AH или DX)

6. Где находится делимое при целочисленном делении операндов?

Делимое в регистре AX (для 16-битных чисел) или DX:AX (для 32-битных чисел).

7. Куда помещаются неполное частное и остаток при делении целочисленных операндов?

При использовании команды `DIV` частное помещается в AL (для 8-битных чисел), AX (для 16-битных чисел) или EAX (для 32-битных чисел).

Остаток помещается в AH (для 8-битных чисел), DX (для 16-битных чисел) или EDX (для 32-битных чисел).

8 Список литературы

Архитектура ЭВМ

Таблица ASCII