7. Лабораторная работа №7. Арифметические операции в NASM.

7.1. Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

7.2. Теоретическое введение

7.2.1. Адресация в NASM

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.

Например, определим переменную `intg DD 3` – это означает, что задается область памяти размером 4 байта, адрес которой обозначен меткой `intg`. В таком случае, команда

```
mov eax,[intg]
```

копирует из памяти по адресу `intg` данные в регистр `eax`. В свою очередь команда

```
mov [intg],eax
```

запишет в память по адресу `intg` данные из регистра `eax`.

Также рассмотрим команду

```
mov eax,intg
```

В этом случае в регистр `eax` запишется адрес `intg`. Допустим, для `intg` выделена память начиная с ячейки с адресом `0x600144`, тогда команда `mov eax,intg` аналогична команде `mov eax,0x600144` – т.е. эта команда запишет в регистр `eax` число `0x600144`.

7.2.2. Арифметические операции в NASM

7.2.2.1. Целочисленное сложение add.

Схема команды целочисленного сложения `add` (от англ. *addition* - добавление) выполняет сложение двух операндов и записывает результат по адресу первого операнда. Команда `add` работает как с числами со знаком, так и без знака и выглядит следующим образом:

```
add <операнд_1>, <операнд_2>
```

Допустимые сочетания операндов для команды `add` аналогичны сочетаниям операндов для команды `mov`.

Так, например, команда `add eax,ebx` прибавит значение из регистра `eax` к значению из регистра `ebx` и запишет результат в регистр `eax`.

Примеры:

```
add ax,5      ; AX = AX + 5
add dx,cx     ; DX = DX + CX
add dx,cl     ; Ошибка: разный размер операндов.
```

7.2.2.2. Целочисленное вычитание sub.

Команда целочисленного вычитания sub (от англ. *subtraction* – вычитание) работает аналогично команде add и выглядит следующим образом:

```
sub <операнд_1>, <операнд_2>
```

Так, например, команда `sub ebx, 5` уменьшает значение регистра ebx на 5 и записывает результат в регистр ebx.

7.2.2.3. Команды инкремента и декремента.

Довольно часто при написании программ встречается операция прибавления или вычитания единицы. Прибавление единицы называется инкрементом, а вычитание — декрементом. Для этих операций существуют специальные команды: inc (от англ. *increment*) и dec (от англ. *decrement*), которые увеличивают и уменьшают на 1 свой операнд.

Эти команды содержат один операнд и имеет следующий вид:

```
inc <операнд>
dec <операнд>
```

Операндом может быть регистр или ячейка памяти любого размера. Команды инкремента и декремента выгодны тем, что они занимают меньше места, чем соответствующие команды сложения и вычитания.

Так, например, команда `inc ebx` увеличивает значение регистра ebx на 1, а команда `inc ax` уменьшает значение регистра ax на 1.

7.2.2.4. Команда изменения знака операнда neg.

Еще одна команда, которую можно отнести к арифметическим командам это команда изменения знака neg:

```
neg <операнд>
```

Команда neg рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный. Операндом может быть регистр или ячейка памяти любого размера.

```
mov  ax,1    ; AX = 1
neg  ax      ; AX = -1
```

7.2.2.5. Команды умножения mul и imul.

Умножение и деление, в отличии от сложения и вычитания, для знаковых и беззнаковых чисел производиться по-разному, поэтому существуют различные команды.

Для *беззнакового* умножения используется команда mul (от англ. *multiply* – умножение):

```
mul <операнд>
```

Для *знакового* умножения используется команда imul:

```
imul <операнд>
```

Для команд умножения один из сомножителей указывается в команде и должен находиться в регистре или в памяти, но не может быть непосредственным операндом. Второй сомножитель в команде явно не указывается и должен находиться в регистре EAX,AX или AL, а результат помещается в регистры EDX:EAX, DX:AX или AX, в зависимости от размера операнда 7.1.

Таблица 7.1. Регистры используемые командами умножения в Nasm

Размер операнда	Неявный множитель	Результат умножения
1 байт	AL	AX
2 байта	AX	DX:AX
4 байта	EAX	EDX:EAX

Пример использования инструкции `mul`:

```
a dw 270

mov ax, 100    ; AX = 100
mul a          ; AX = AX*a,
mul bl         ; AX = AL*BL
mul ax         ; DX:AX = AX*AX
```

7.2.2.6. Команды деления `div` и `idiv`.

Для деления, как и для умножения, существует 2 команды `div` (от англ. *divide* - деление) и `idiv`:

```
div <делитель>     ; Беззнаковое деление
idiv <делитель>    ; Знаковое деление
```

В командах указывается только один операнд – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным операндом. Местоположение делимого и результата для команд деления зависит от размера делителя. Кроме того, так как в результате деления получается два числа – частное и остаток, то эти числа помещаются в определённые регистры 7.2.

Таблица 7.2. Регистры используемые командами деления в Nasm

Размер операнда (делителя)	Делимое	Частное	Остаток
1 байт	AX	AL	AH
2 байта	DX:AX	AX	DX
4 байта	EDX:EAX	EAX	EDX

Например, после выполнения инструкций

```
mov   ax,31
mov   dl,15
div   dl
```

результат 2 (31/15) будет записан в регистр `al`, а остаток 1 (остаток от деления 31/15) — в регистр `ah`.

Если делитель — это слово (16-бит), то делимое должно записываться в регистрах `dx:ax`. Так в результате выполнения инструкций

```
mov   ax,2   ; загрузить в регистровую
mov   dx,1   ; пару `dx:ax` значение 10002h
mov   bx,10h
div   bx
```

в регистр `ax` запишется частное 1000h (результат деления 10002h на 10h), а в регистр `dx` — 2 (остаток от деления).

7.2.3. Перевод символа числа в десятичную символьную запись

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). Поэтому, 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 необходимо проводить преобразование ASCII символов в числа и обратно.

Для выполнения лабораторных работ в файле `in_out.asm` реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Это:

- `iprint` – вывод на экран чисел в формате ASCII, перед вызовом `iprint` в регистр `eax` необходимо записать выводимое число (`mov eax,<int>`).
- `iprintLF` – работает аналогично `iprint`, но при выводе на экран после числа добавляет к символ перевода строки.
- `atoi` – функция преобразует ascii-код символа в целое число и записает результат в регистр `eax`, перед вызовом `atoi` в регистр `eax` необходимо записать число (`mov eax,<int>`).

7.3. Порядок выполнения лабораторной работы

7.3.1. Символьные и численные данные в NASM

1. Создайте каталог для программам лабораторной работы № 7, перейдите в него и создайте файл `lab7-1.asm`:

```
mkdir  ~/work/arch-pc/lab07
cd  ~/work/arch-pc/lab07
touch lab7-1.asm
```

2. Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения записанные в регистр `eax`.

Введите в файл `lab7-1.asm` текст программы из листинга 7.1. В данной программе в регистр `eax` записывается символ 6 (`mov  eax,'6'`), в регистр `ebx` символ 4 (`mov  ebx,'4'`). Далее к значению в регистре `eax` прибавляем значение регистра `ebx` (`add  eax,ebx`, результат сложения запишется в регистр `eax`). Далее выводим результат. Так как для работы функции `sprintLF` в регистр `eax` должен быть записан адрес, необходимо использовать дополнительную переменную. Для этого запишем значение регистра `eax` в переменную `buf1` (`mov [buf1],eax`), а затем запишем адрес переменной `buf1` в регистр `eax` (`mov eax,buf1`) и вызовем функцию `sprintLF`.

Листинг 7.1. Программа вывода значения регистра `eax`

```
%include 'in_out.asm'

SECTION  .bss
buf1:     RESB 80

 SECTION .text
 GLOBAL _start
```

```
_start:

mov  eax,'6'
mov  ebx,'4'
add  eax,ebx
mov [buf1],eax
mov eax,buf1
call sprintLF

call quit
```

Создайте исполняемый файл и запустите его.

```
nasm -f elf lab7-1.asm
ld -m elf_i386 -o lab7-1 lab7-1.o
./lab7-1
```

> ВАЖНО! Для корректной работы программы подключаемый файл `in_out.asm` должен лежать в том же каталоге, что и файл с текстом программы. Перед созданием исполняемого файла создайте копию файла `in_out.asm` в каталоге `~/work/arch-pc/lab07`.

В данном случае при выводе значения регистра `eax` мы ожидаем увидеть число `10`. Однако результатом будет символ `j`. Это происходит потому, что код символа `6` равен 00110110 в двоичном представлении (или 54 в десятичном представлении), а код символа `4` – 00110100 (52). Команда `add eax,ebx` запишет в регистр `eax` сумму кодов – 01101010 (106), что в свою очередь является кодом символа `j` (см. таблицу ASCII в приложении).

3. Далее изменим текст программы и вместо символов, запишем в регистры числа. Исправьте текст программы (Листинг 1) следующим образом: замените строки

```
mov  eax,'6'
mov  ebx,'4'
```

на строки

```
mov  eax,6
mov  ebx,4
```

Создайте исполняемый файл и запустите его.

Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10. Пользуясь таблицей ASCII определите какому символу соответствует код 10. Отображается ли этот символ при выводе на экран?

4. Как отмечалось выше, для работы с числами в файле in_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Преобразуем текст программы из Листинга 7.1 с использованием этих функций.

Создайте файл lab7-2.asm в каталоге ~/work/arch-pc/lab07 и введите в него текст программы из листинга 7.2.

```
touch ~/work/arch-pc/lab07/lab7-2.asm
```

Листинг 7.2. Программа вывода значения регистра eax

```
%include 'in_out.asm'

 SECTION .text
 GLOBAL _start
  _start:

 mov  eax,'6'
```

```
mov  ebx,'4'
add  eax,ebx
call iprintLF

call quit
```

Создайте исполняемый файл и запустите его.

```
nasm -f elf lab7-1.asm
ld -m elf_i386 -o lab7-1 lab7-1.o
./lab7-1
```

В результате работы программы мы получим число `106`. В данном случае, как и в первом, команда `add` складывает коды символов ‘6’ и ‘4’ (54+52=106). Однако, в отличии от программы из листинга 7.1, функция `iprintLF` позволяет вывести число, а не символ, кодом которого является это число.

5. Аналогично предыдущему примеру изменим символы на числа. Замените строки

```
mov  eax,'6'
mov  ebx,'4'
```

на строки

```
mov  eax,6
mov  ebx,4
```

Создайте исполняемый файл и запустите его. Какой результат будет получен при исполнении программы?

Замените функцию `iprintLF` на `iprint`. Создайте исполняемый файл и запустите его. Чем отличается вывод функций `iprintLF` и `iprint`?

7.3.2. Выполнение арифметических операций в NASM

6. В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения $f(x) = (5 * 2 + 3)/3$.

Создайте файл `lab7-3.asm` в каталоге `~/work/arch-pc/lab07`:

```
touch ~/work/arch-pc/lab07/lab7-3.asm
```

Внимательно изучите текст программы из листинга 7.3 и введите в `lab7-3.asm`.

Листинг 7.3. Программа вычисления выражения $f(x) = (5 * 2 + 3)/3$

```
;-------------------------------
; Программа вычисления выражения
;-------------------------------

%include    'in_out.asm'      ; подключение внешнего файла

  SECTION .data

  div:  DB 'Результат: ',0
  rem:  DB 'Остаток от деления: ',0

  SECTION .text
  GLOBAL _start
   _start:

   ; ---- Вычисление выражения
   mov eax,5          ; EAX=5
   mov ebx,2          ; EBX=2
   mul ebx            ; EAX=EAX*EBX
```

```
add eax,3               ; EAX=EAX+3
xor edx,edx         ; обнуляем EDX для корректной работы div
mov ebx,3           ; EBX=3
div ebx             ; EAX=EAX/3, EDX=остаток от деления

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

; ----  Вывод результата на экран

mov  eax,div        ; вызов подпрограммы печати
call sprint         ; сообщения 'Результат: '
mov  eax,edi        ; вызов подпрограммы печати значения
call iprintLF       ; из 'edi' в виде символов

mov  eax,rem        ; вызов подпрограммы печати
call sprint         ; сообщения 'Остаток от деления: '
mov  eax,edx        ; вызов подпрограммы печати значения
call iprintLF       ; из 'edx' (остаток) в виде символов

call quit           ; вызов подпрограммы завершения
```

Создайте исполняемый файл и запустите его. Результат работы программы должен быть следующим:

```
user@dk4n31:~$ ./lab7-3
Результат: 4
Остаток от деления: 1
user@dk4n31:~$
```

Измените текст программы для вычисления выражения $f(x) = (4 * 6 + 2)/5$. Создайте исполняемый файл и проверьте его работу.

7. В качестве другого примера рассмотрим программу вычисления варианта

задания по номеру студенческого билета, работающую по следующему алгоритму:

- вывести запрос на введение № студенческого билета
- вычислить номер варианта по формуле: $(S_n \mod 20) + 1$, где S_n – номер студенческого билета (В данном случае $a \mod b$ – это остаток от деления a на b).
- вывести на экран номер варианта.

В данном случае число, над которым необходимо проводить арифметические операции, вводится с клавиатуры. Как отмечалось выше ввод с клавиатуры осуществляется в символьном виде и для корректной работы арифметических операций в NASM символы необходимо преобразовать в числа. Для этого может быть использована функция `atoi` из файла `in_out.asm`.

Создайте файл `variant.asm` в каталоге `~/work/arch-pc/lab07`:

```
touch ~/work/arch-pc/lab07/variant.asm
```

Внимательно изучите текст программы из листинга 7.4 и введите в файл `variant.asm`.

Листинг 7.4. Программа вычисления вычисления варианта задания по номеру студенческого билета

```
;-----------------------------
; Программа вычисления варианта
;-----------------------------

%include    'in_out.asm'

  SECTION .data
  msg:  DB 'Введите № студенческого билета: ',0
  rem:  DB 'Ваш вариант: ',0
```

```
SECTION  .bss
x:    RESB 80

SECTION .text
GLOBAL _start
 _start:

 mov  eax, msg
 call sprintLF

 mov  ecx, x
 mov  edx, 80
 call sread

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

 xor edx,edx
 mov ebx,20
 div ebx
 inc edx

 mov  eax,rem
 call sprint
 mov  eax,edx
 call iprintLF

 call quit
```

Создайте исполняемый файл и запустите его. Проверьте результат работы программы вычислив номер варианта аналитически.

Включите в отчет по выполнению лабораторной работы ответы на следующие

вопросы:

1. Какие строки листинга 7.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’?
2. Для чего используется следующие инструкции? `nasm  mov ecx, x  mov edx, 80  call sread`
3. Для чего используется инструкция “`call atoi`”?
4. Какие строки листинга 7.4 отвечают за вычисления варианта?
5. В какой регистр записывается остаток от деления при выполнении инструкции “`div ebx`”?
6. Для чего используется инструкция “`inc edx`”?
7. Какие строки листинга 7.4 отвечают за вывод на экран результата вычислений?

7.4. Задание для самостоятельной работы

1. Написать программу вычисления выражения $y = f(x)$. Программа должна выводить выражение для вычисления, выводить запрос на ввод значения x, вычислять заданное выражение в зависимости от введенного x, выводить результат вычислений. Вид функции $f(x)$ выбрать из таблицы 7.3 вариантов заданий в соответствии с номером полученным при выполнении лабораторной работы. Создайте исполняемый файл и проверьте его работу для значений x_1 и x_2 из 7.3.

7.5. Варианты заданий

Таблица 7.3. Выражения для $f(x)$ для задания №1

Номер варианта	Выражение для $f(x)$	x_1	x_2
1	$(10 + 2x)/3$	1	10
2	$(12x + 3)5$	1	6
3	$(2 + x)^2$	2	8
4	$\frac{4}{3}(x - 1) + 5$	4	10
5	$(9x - 8)/8$	8	64
6	$x^3/2 + 1$	2	5
7	$5(x - 1)^2$	3	5
8	$(11 + x) \cdot 2 - 6$	1	9
9	$10 + (31x - 5)$	3	1
10	$5(x + 18) - 28$	2	3
11	$10(x + 1) - 10$	1	7
12	$(8x - 6)/2$	1	5
13	$(8x + 6) \cdot 10$	1	4
14	$(\frac{x}{2} + 8) \cdot 3$	1	4
15	$(5 + x)^2 - 3$	5	1
16	$(10x - 5)^2$	3	1
17	$18(x + 1)/6$	3	1
18	$3(x + 10) - 20$	1	5
19	$(\frac{1}{3}x + 5) \cdot 7$	3	9
20	$x^3 \cdot \frac{1}{3} + 21$	1	3

При выполнении задания преобразовывать (упрощать) выражения для $f(x)$ **нельзя**. При выполнении деления в качестве результата можно использовать только целую часть от деления и не учитывать остаток (т.е. $5 : 2 = 2$).

7.6. Содержание отчёта

Отчёт должен включать:

- Титульный лист с указанием номера лабораторной работы и ФИО студента.
- Формулировка цели работы.
- Описание результатов выполнения лабораторной работы:
  - описание выполняемого задания;
  - скриншоты (снимки экрана), фиксирующие выполнение заданий лабораторной работы;
  - комментарии и выводы по результатам выполнения заданий.
- Описание результатов выполнения заданий для самостоятельной работы:
  - описание выполняемого задания;
  - скриншоты (снимки экрана), фиксирующие выполнение заданий;
  - комментарии и выводы по результатам выполнения заданий;
  - листинги написанных программ (текст программ).
- Выводы, согласованные с целью работы.

Отчёт по выполнению лабораторной работы оформляется в формате `Markdown`. В качестве отчёта необходимо предоставить отчёты в 3 форматах: `pdf`, `docx` и `md`. А также файлы с исходными текстами написанных при выполнении лабораторной работы программ (файлы `*.asm`). Файлы необходимо загрузить на странице курса в ТУИС в задание к соответствующей лабораторной работе и загрузить на Github.

7.7. Вопросы для самопроверки

1. Какой синтаксис команды сложения чисел?
2. Какая команда выполняет умножение без знака?
3. Какой синтаксис команды деление чисел без знака?
4. Куда помещается результат при умножении двухбайтовых операндов?
5. Перечислите арифметические команды с целочисленными операндами и дайте их назначение.
6. Где находится делимое при целочисленном делении операндов?
7. Куда помещаются неполное частное и остаток при делении целочисленных операндов?

7.8. Приложение. ASCII таблица

Таблица 7.4. Таблица символов ASCII

DEC	OCT	HEX	BIN	Symbol
0	0	0x00	0	NUL, `\0`
1	1	0x01	1	SOH
2	2	0x02	10	STX
3	3	0x03	11	ETX
4	4	0x04	100	EOT
5	5	0x05	101	ENQ
6	6	0x06	110	ACK
7	7	0x07	111	BEL
8	10	0x08	1000	BS

DEC	OCT	HEX	BIN	Symbol
9	11	0x09	1001	HT, \t
10	12	0x0A	1010	LF, \n
11	13	0x0B	1011	VT
12	14	0x0C	1100	FF
13	15	0x0D	1101	CR, \r
14	16	0x0E	1110	SO
15	17	0x0F	1111	SI
16	20	0x10	10000	DLE
17	21	0x11	10001	DC1
18	22	0x12	10010	DC2
19	23	0x13	10011	DC3
20	24	0x14	10100	DC4
21	25	0x15	10101	NAK
22	26	0x16	10110	SYN
23	27	0x17	10111	ETB
24	30	0x18	11000	CAN
25	31	0x19	11001	EM
26	32	0x1A	11010	SUB
27	33	0x1B	11011	ESC
28	34	0x1C	11100	FS
29	35	0x1D	11101	GS
30	36	0x1E	11110	RS

DEC	OCT	HEX	BIN	Symbol
31	37	0x1F	11111	US
32	40	0x20	100000	
33	41	0x21	100001	!
34	42	0x22	100010	"
35	43	0x23	100011	#
36	44	0x24	100100	$
37	45	0x25	100101	%
38	46	0x26	100110	&
39	47	0x27	100111	'
40	50	0x28	101000	(
41	51	0x29	101001	)
42	52	0x2A	101010	*
43	53	0x2B	101011	+
44	54	0x2C	101100	,
45	55	0x2D	101101	-
46	56	0x2E	101110	.
47	57	0x2F	101111	/
48	60	0x30	110000	0
49	61	0x31	110001	1
50	62	0x32	110010	2
51	63	0x33	110011	3
52	64	0x34	110100	4

DEC	OCT	HEX	BIN	Symbol
53	65	0x35	110101	5
54	66	0x36	110110	6
55	67	0x37	110111	7
56	70	0x38	111000	8
57	71	0x39	111001	9
58	72	0x3A	111010	:
59	73	0x3B	111011	;
60	74	0x3C	111100	<
61	75	0x3D	111101	=
62	76	0x3E	111110	>
63	77	0x3F	111111	?
64	100	0x40	1000000	@
65	101	0x41	1000001	A
66	102	0x42	1000010	B
67	103	0x43	1000011	C
68	104	0x44	1000100	D
69	105	0x45	1000101	E
70	106	0x46	1000110	F
71	107	0x47	1000111	G
72	110	0x48	1001000	H
73	111	0x49	1001001	I
74	112	0x4A	1001010	J

DEC	OCT	HEX	BIN	Symbol
75	113	0x4B	1001011	K
76	114	0x4C	1001100	L
77	115	0x4D	1001101	M
78	116	0x4E	1001110	N
79	117	0x4F	1001111	O
80	120	0x50	1010000	P
81	121	0x51	1010001	Q
82	122	0x52	1010010	R
83	123	0x53	1010011	S
84	124	0x54	1010100	T
85	125	0x55	1010101	U
86	126	0x56	1010110	V
87	127	0x57	1010111	W
88	130	0x58	1011000	X
89	131	0x59	1011001	Y
90	132	0x5A	1011010	Z
91	133	0x5B	1011011	[
92	134	0x5C	1011100	\
93	135	0x5D	1011101	]
94	136	0x5E	1011110	^
95	137	0x5F	1011111	_
96	140	0x60	1100000	‘

DEC	OCT	HEX	BIN	Symbol
97	141	0x61	1100001	a
98	142	0x62	1100010	b
99	143	0x63	1100011	c
100	144	0x64	1100100	d
101	145	0x65	1100101	e
102	146	0x66	1100110	f
103	147	0x67	1100111	g
104	150	0x68	1101000	h
105	151	0x69	1101001	i
106	152	0x6A	1101010	j
107	153	0x6B	1101011	k
108	154	0x6C	1101100	l
109	155	0x6D	1101101	m
110	156	0x6E	1101110	n
111	157	0x6F	1101111	o
112	160	0x70	1110000	p
113	161	0x71	1110001	q
114	162	0x72	1110010	r
115	163	0x73	1110011	s
116	164	0x74	1110100	t
117	165	0x75	1110101	u
118	166	0x76	1110110	v

DEC	OCT	HEX	BIN	Symbol
119	167	0x77	1110111	w
120	170	0x78	1111000	x
121	171	0x79	1111001	y
122	172	0x7A	1111010	z
123	173	0x7B	1111011	{
124	174	0x7C	1111100	|
125	175	0x7D	1111101	}
126	176	0x7E	1111110	~
127	177	0x7F	1111111	

См. побробнее https://snipp.ru/handbk/table-ascii