

Машинно-зависимые языки программирования

Савельев Игорь Леонидович

- Ввод/вывод, DOS и Linux
 - ASCII
 - Файловые операции

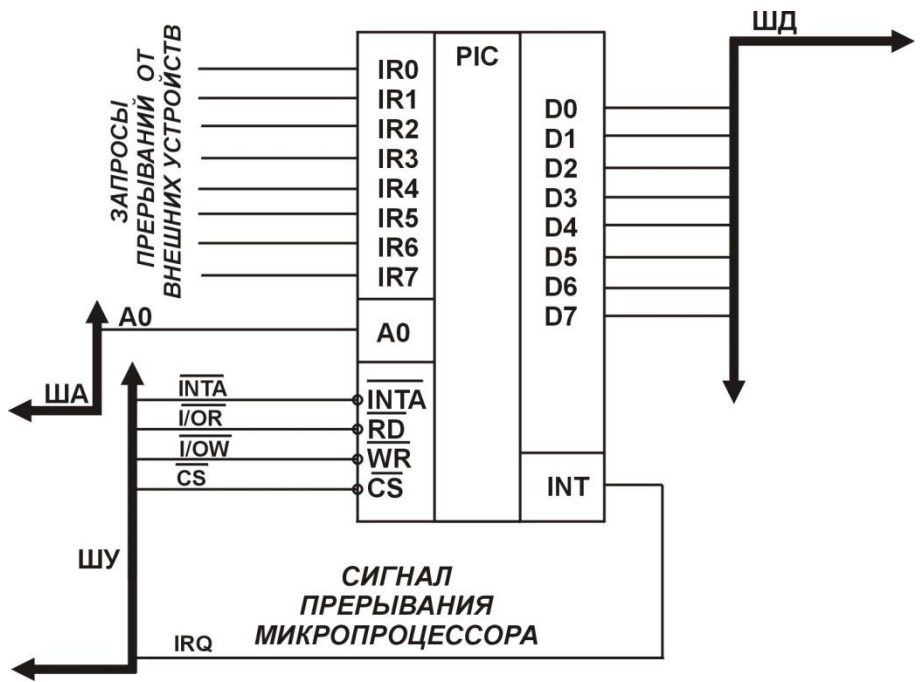
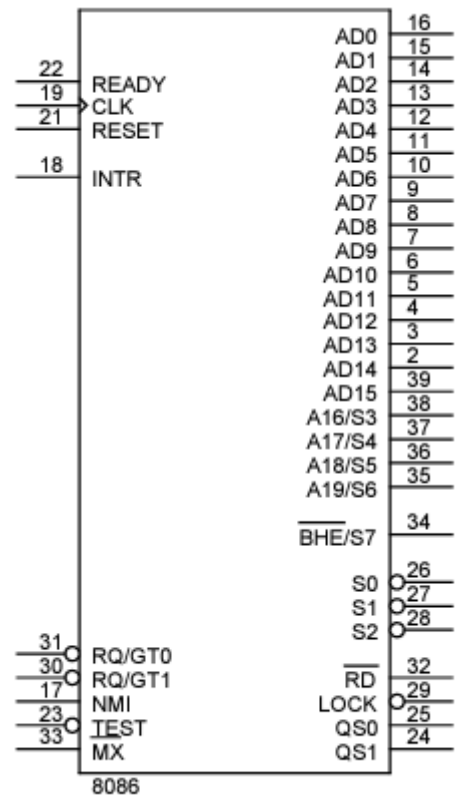
4. Ввод/вывод. Прерывание. Программное прерывание

- Polling (данные по опросу, инициатор получатель)
- Извещение (данные по инициативе отправителя)
 - Подписка (получать не все извещения)
- Прерывание (сигнал о наличии данных или о событии)

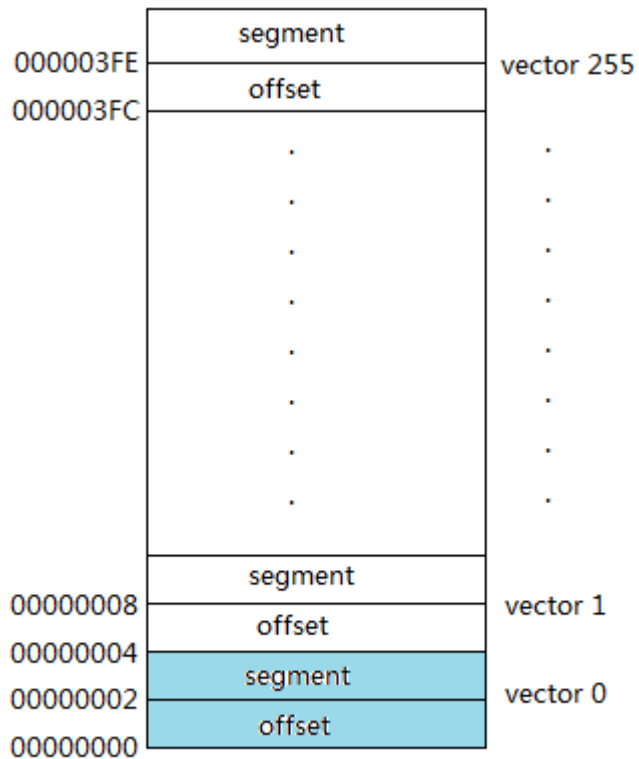
В зависимости от источника, прерывания делятся на

- **аппаратные** - возникают как реакция микропроцессора на физический сигнал от некоторого устройства (клавиатура, системные часы, клавиатура, жесткий диск и т.д.), по времени возникновения эти прерывания асинхронны, т.е. происходят в случайные моменты времени;
- **программные** - вызываются искусственно с помощью соответствующей команды из программы (int), предназначены для выполнения некоторых действий операционной системы, являются синхронными;
- **исключения** - являются реакцией микропроцессора на нестандартную ситуацию, возникшую внутри микропроцессора во время выполнения некоторой команды программы (деление на ноль, прерывание по флагу TF (трассировка)).

4. Ввод/вывод. Прерывание. Программное прерывание



4. Ввод/вывод. Прерывание. Программное прерывание



Адрес вида: 0000:0000 (CS:IP)

$$256 * 4 = 1K$$

- Аппаратный сигнал
- Программный вызов
- Исключение

Interrupt Vector Table

4. Ввод/вывод. Прерывание. Программное прерывание

Номер прерывания	Описание
0x00	Деление на 0
0x01	Пошаговый режим (отладка)
0x02	Немаскируемое прерывание
0x03	Точка останова (breakpoint)
0x04	Переполнение
0x05	Печать экрана (Print Screen)
0x06	Недопустимая команда (для 286 и выше)
0x07	Вызов отсутствующего сопроцессора
0x08	Таймер (IRQ0)
0x09	Клавиатура (IRQ1)

4. Ввод/вывод. Прерывание. Программное прерывание

Номер прерывания	Описание
0x14	Обслуживание COM портов
0x15	АТ функции (задержка, работа с джойстиком и др.)
0x16	Клавиатурный ввод/вывод (чтение клавиш и др.)
0x17	Обслуживание LPT портов
0x18	ROM-Basic
0x19	Начальная загрузка
0x1A	Системное время
0x1B	Обработчик Ctrl+Break
0x1C	Процедура, вызываемая обработчиком INT 08h (заглушка)

4. Ввод/вывод. Прерывание. Программное прерывание

Номер прерывания	Описание
0x20	Завершение программы (обычно, для COM файлов)
0x21	Обращение к функциям DOS
0x22	Адрес обработчика завершения задачи
0x23	Адрес обработчика Ctrl+Break
0x24	Адрес обработчика критической ошибки
0x25	Чтение логических секторов
0x26	Запись логических секторов
0x27	Создание резидентной программы
0x28	Ожидание нажатия клавиши

Ralf Brown's Interrupt List (RBIL, англ. Список прерываний Ральфа Брауна) — справочное руководство, обширный список аппаратных и программных интерфейсов (прежде всего, сервисов, доступных через прерывания и сопутствующих структур данных) на архитектуре x86.

«Список» вёлся Ральфом Брауном (в наст. время старший исследователь-системотехник в Университете Карнеги-Меллон) и собран из различных источников с 1989 по 2000 годы.

«Список» широко использовался в качестве справочного пособия программистами на IBM-совместимых компьютерах во времена MS-DOS и не теряет актуальности для системных программистов.

Последняя редакция 61 от 16 июля 2000 года состоит из 5 Мб текста, что соответствует 2-3 тыс. печатных страниц.

4. Ввод/вывод. Прерывание. Программное прерывание

Вывод на экран средствами DOS осуществляет **09** функция **INT 21H** DOS. Номер функции указывается в регистре **AH**. Адрес выводимой строки в **DS:DX**. В процессе выполнения операции конец сообщения определяется по ограничителю (\$).

PRMP DB 'Строка','\$'

.

MOV AH, 09 ; Запрос вывода на экран

LEA DX, PRMP ; Загрузка адреса сообщ.

INT 21H ; Вызов DOS

Знак ограничителя '\$' можно кодировать непосредственно после символьной строки (как показано в примере), внутри строки: 'Имя покупателя?\$', или в следующем операторе DB '\$'. Используя данную операцию, нельзя вывести на экран символ доллара "\$". Кроме того, если знак доллара будет отсутствовать в конце строки, то на экран будут выводиться все последующие символы, пока знак "\$" не встретится в памяти.

Команда LEA загружает адрес области PRMP в регистр DX для передачи в DOS адреса выводимой информации. Адрес поля PRMP, загружаемый в DX по команде LEA, является относительным, поэтому для вычисления абсолютного адреса данных DOS складывает значения регистров DS и DX (DS:DX).

В архитектуре x86-64 ОС Linux существует несколько различных способов системных вызовов:

- `int 80h`
- `sysenter/sysexit`
- `syscall/sysret`
- `vsyscall`
- `vDSO`

4. Ввод/вывод. Прерывание. Программное прерывание

Изначально, в архитектуре x86, Linux использовал программное прерывание *128* для совершения системного вызова.

Для указания номера системного вызова, пользователь задаёт в *eax* номер системного вызова, а его параметры располагает по порядку в регистрах *ebx*, *ecx*, *edx*, *esi*, *edi*, *ebp*.

Далее вызывается инструкция *int 80h*, которая программно вызывает прерывание. Процессором вызывается обработчик прерывания, установленный ядром Linux ещё во время инициализации ядра.

В x86-64 вызов прерывания используется только во время эмуляции режима x32 для обратной совместимости.

4. Ввод/вывод. Прерывание. Программное прерывание

```
section .text  
global _start
```

```
_start:
```

```
    mov     edx, len  
    mov     ecx, msg  
    mov     ebx, 1      ; file descriptor (stdout)  
    mov     eax, 4      ; system call number (sys_write)  
    int     0x80        ; call kernel
```

```
    mov     eax, 1      ; system call number (sys_exit)  
    int     0x80        ; call kernel
```

```
section .data
```

```
msg     db 'Hello, world!',0xa  
len     equ $ - msg
```

4. Ввод/вывод. Прерывание. Программное прерывание

`sysenter/sysexit`

Спустя некоторое время, ещё когда не было x86-64, в Intel поняли, что можно ускорить системные вызовы, если создать специальную инструкцию системного вызова, тем самым минуя некоторые издержки прерывания. Так появилась пара инструкций **sysenter/sysexit**. Ускорение достигается за счёт того, что на аппаратном уровне при выполнении инструкции **sysenter** опускается множество проверок на валидность дескрипторов, а так же проверок, зависящих от уровня привилегий. Также инструкция опирается на то, что вызывающая её программа использует плоскую модель памяти. В архитектуре Intel, инструкция валидна как для режима совместимости, так и для расширенного режима, но у AMD данная инструкция в расширенном режиме приводит к исключению неизвестного опкода. Поэтому в настоящее время пара **sysenter/sysexit** используется только в режиме совместимости.

4. Ввод/вывод. Прерывание. Программное прерывание

```
section .text  
global _start
```

```
_start:
```

```
    mov     edx, len           ;message length  
    mov     ecx, msg          ;message to write  
    mov     ebx, 1            ;file descriptor (stdout)  
    mov     eax, 4            ;system call number (sys_write)  
    sysenter
```

```
....
```

```
section .data  
msg     db 'Hello, world!',0xa  
len     equ $ - msg
```

4. Ввод/вывод. Прерывание. Программное прерывание

Так как именно AMD разработали x86-64 архитектуру, которая и называется AMD64, то они решили создать свой собственный системный вызов.

Инструкция разрабатывалась AMD, как аналог ***sysenter/sysexit*** для архитектуры IA-32. В AMD позаботились о том, чтобы инструкция была реализована как в расширенном режиме, так и в режиме совместимости, но в Intel решили не поддерживать данную инструкцию в режиме совместимости. Несмотря на всё это, Linux имеет 2 обработчика для каждого из режимов: для x32 и x64

4. Ввод/вывод. Прерывание. Программное прерывание

```
section .text
global _start
```

```
_start:
```

```
    mov     rdx, len           ;message length
    mov     rsi, msg           ;message to write
    mov     rdi, 1             ;file descriptor (stdout)
    mov     rax, 1             ;system call number (sys_write)
    syscall
```

```
    mov     rax, 60            ;system call number (sys_exit)
    syscall
```

```
section .data
```

```
msg     db 'Hello, world!',0xa
len     equ $ - msg
```

4. Ввод/вывод. Прерывание. Программное прерывание

Реализация	время (нс)
int 80h	498
sysenter	338
syscall	278
vsyscall emulate	692
vsyscall native	278
vDSO	37
vDSO-32	51

4. Ввод/вывод. Прерывание. Программное прерывание

%eax	Имя системного вызова	%ebx	%ecx	%edx
2	sys_fork	struct pt_regs	—	—
3	sys_read	unsigned int	char *	size_t
4	sys_write	unsigned int	const char *	size_t
5	sys_open	const char *	int	int
6	sys_close	unsigned int	—	—
8	sys_creat	const char *	int	—
19	sys_lseek	unsigned int	off_t	unsigned int

4. Ввод/вывод. Прерывание. Программное прерывание

```
ssize_t sys_read(unsigned int fd, char * buf, size_t count)
```

; Считываем данные из консоли

mov eax, 3 ; sys_read()

mov ebx, 0 ; stdin

mov ecx, buf ; куда читаем

mov edx, [count] ; сколько читаем

int 0x80

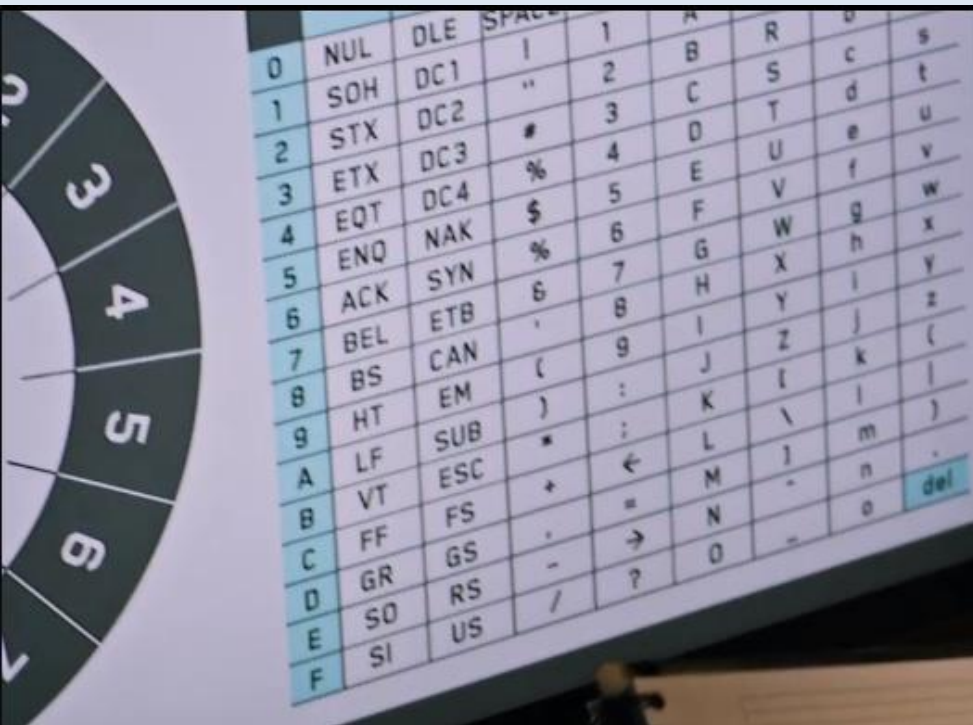
section .data

buf db 27 dup(0)

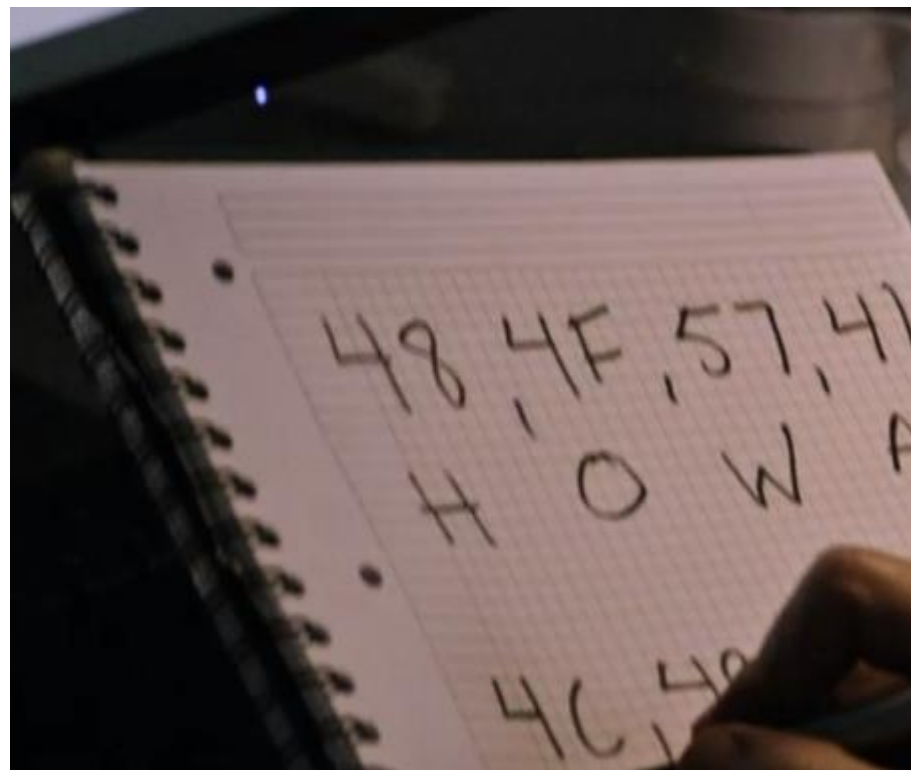
count db 26

4. Ввод/вывод. Прерывание. Программное прерывание

```
SECTION .data
msg db "Hello, world!",0
fmt db "%s",0Ah
SECTION .text
GLOBAL _start           ; точка входа в программу
EXTERN printf          ; внешняя функция библиотеки Си
_start:
push msg                ; второй параметр - указатель на строку
push fmt                ; первый параметр - указатель на формат
call printf            ; вызов функции
add esp, 4*2            ; очистка стека от параметров
mov eax, 1              ; системный вызов № 1 — sys_exit
xor ebx, ebx            ; выход с кодом 0
int 80h                 ; вызов ядра
```



0	NUL	DLE	SPACE	1	A	R	D	S
1	SOH	DC1	"	2	B	S	c	t
2	STX	DC2	#	3	C	T	d	u
3	ETX	DC3	%	4	D	U	e	v
4	EQT	DC4	\$	5	E	V	f	w
5	ENQ	NAK	%	6	F	W	g	x
6	ACK	SYN	6	7	G	X	h	y
7	BEL	ETB	'	8	H	Y	i	z
8	BS	CAN	(9	I	Z	j	(
9	HT	EM)	:	J	[k	
A	LF	SUB	*	:	K	\	l	
B	VT	ESC	+	<	L]	m)
C	FF	FS	.	=	M	-	n	.
D	GR	GS	-	→	N	_	o	del
E	SO	RS	/	?	O	-		
F	SI	US						



48, 4F, 57, 41
H O W A
4C, 48

1. Ввод/вывод. ASCII

ASCII (англ. *American standard code for information interchange*) — название таблицы (кодировки, набора), в которой некоторым распространённым печатным и непечатным символам сопоставлены числовые коды. Таблица была разработана и стандартизирована в США, в 1963 году.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

1. Ввод/вывод. ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

```
char VarA = 'A'; // == 0x41
```

```
char Var_a = 'a'; // == 0x61
```

```
VarA == Var_a - 0x20; // - 32
```

```
char dig0 = '0'; // == 0x30
```









```
char dig1 = '1'; // == 0x31
```

```
char dig9 = '9'; // == 0x39
```

```
dig9 == dig0 + 0x09 == '0' + 9
```

```
9 == dig9 - dig0 == dig9 - '0'
```


1. Ввод/вывод. ASCII – dos866

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
8.	А 410	Б 411	В 412	Г 413	Д 414	Е 415	Ж 416	З 417	И 418	Й 419	К 41A	Л 41B	М 41C	Н 41D	О 41E	П 41F
9.	Р 420	С 421	Т 422	У 423	Ф 424	Х 425	Ц 426	Ч 427	Ш 428	Щ 429	Ъ 42A	Ы 42B	Ь 42C	Э 42D	Ю 42E	Я 42F
A.	а 430	б 431	в 432	г 433	д 434	е 435	ж 436	з 437	и 438	й 439	к 43A	л 43B	м 43C	н 43D	о 43E	п 43F
B.	 2591	 2592	 2593	 2502	├ 2524	┤ 2561	├┤ 2562	┐ 2556	┑ 2555	├┤ 2563	├├ 2551	┐┐ 2557	┐┐ 255D	┐┐ 255C	┐┐ 255B	┐ 2510
C.	┐ 2514	┐ 2534	┐ 252C	┐ 251C	— 2500	┐ 253C	┐ 255E	┐┐ 255F	┐ 255A	┐ 2554	┐ 2569	┐ 2566	┐ 2560	= 2550	┐ 256C	┐ 2567
D.	┐ 2568	┐ 2564	┐ 2565	┐ 2559	┐ 2558	┐ 2552	┐ 2553	┐ 256B	┐ 256A	┐ 2518	┐ 250C	 2588	 2584	 258C	 2590	 2580
E.	р 440	с 441	т 442	у 443	ф 444	х 445	ц 446	ч 447	ш 448	щ 449	ъ 44A	ы 44B	ь 44C	э 44D	ю 44E	я 44F
F.	Ё 401	ё 451	Є 404	є 454	Ї 407	ї 457	Ў 40E	ў 45E	° B0	· 2219	· B7	√ 221A	№ 2116	☒ A4	■ 25A0	 A0

1. Ввод/вывод. ASCII – win1251

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
8.	Ъ 402	Ѓ 403	‚ 201A	ѓ 453	„ 201E	… 2026	† 2020	‡ 2021	€ 20AC	‰ 2030	Љ 409	‹ 2039	Њ 40A	Ќ 40C	Ћ 40B	Ц 40F
9.	ђ 452	‘ 2018	’ 2019	“ 201C	” 201D	• 2022	— 2013	— 2014		™ 2122	љ 459	› 203A	њ 45A	ќ 45C	ћ 45B	џ 45F
A.		Ў 40E	ў 45E	Ј 408	Ѡ A4	Ѓ 490	Ќ A6	§ A7	Ё 401	© A9	Є 404	« AB	¬ AC		® AE	Ї 407
B.	° B0	± B1	І 406	і 456	г 491	μ B5	¶ B6	· B7	ё 451	№ 2116	є 454	» BB	ј 458	Ѕ 405	ѕ 455	ї 457
C.	А 410	Б 411	В 412	Г 413	Д 414	Е 415	Ж 416	З 417	И 418	Й 419	К 41A	Л 41B	М 41C	Н 41D	О 41E	П 41F
D.	Р 420	С 421	Т 422	У 423	Ф 424	Х 425	Ц 426	Ч 427	Ш 428	Щ 429	Ъ 42A	Ы 42B	Ь 42C	Э 42D	Ю 42E	Я 42F
E.	а 430	б 431	в 432	г 433	д 434	е 435	ж 436	з 437	и 438	й 439	к 43A	л 43B	м 43C	н 43D	о 43E	п 43F
F.	р 440	с 441	т 442	у 443	ф 444	х 445	ц 446	ч 447	ш 448	щ 449	ъ 44A	ы 44B	ь 44C	э 44D	ю 44E	я 44F

ASCII и KOI8-R

Код Обмена Информацией, 8 бит
rfc1489

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F		
0-	⌘	⌘	⌘	♥	♦	♣	♠	•	◼	◻	◼	♂	♀	♂	♂	⌘	0-	
1-	▶	◀	⬆	!!	¶	§	—	‡	↑	↓	→	←	⌞	⌞	▲	▼	1-	
2-		!	"	#	\$	%	&	'	()	*	+	,	-	.	/	2-	
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	3-	
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	4-	
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_	5-	
6-	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	6-	
7-	p	q	r	s	t	u	v	w	x	y	z	{		}	~	␣	7-	
8-	—		Г	г	Л	л	Т	т	␣	␣	␣	␣	␣	␣	␣	␣	8-	
9-	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	9-
A-	=		Г	ё	П	Г	Г	П	П	Л	Ц	Ц	Г	Ц	Ц	Г	A-	
B-			Г	Ё			Г	П	П	␣	␣	␣	␣	␣	␣	␣	B-	
C-	Ю	А	Б	Ц	Д	Е	Ф	Г	Х	И	Й	К	Л	М	Н	О	C-	
D-	П	Я	Р	С	Т	У	Ж	В	Ь	Ы	Э	Ш	Э	Ш	Ч	Ъ	D-	
E-	Ю	А	Б	Ц	Д	Е	Ф	Г	Х	И	Й	К	Л	М	Н	О	E-	
F-	П	Я	Р	С	Т	У	Ж	В	Ь	Ы	Э	Ш	Э	Ш	Ч	Ъ	F-	
	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F		

```
admin@host:~$ locale
LANG=ru_RU.UTF-8
LC_CTYPE="ru_RU.UTF-8"
LC_NUMERIC="ru_RU.UTF-8"
LC_TIME="ru_RU.UTF-8"
LC_COLLATE="ru_RU.UTF-8"
LC_MONETARY="ru_RU.UTF-8"
LC_MESSAGES="ru_RU.UTF-8"
LC_PAPER="ru_RU.UTF-8"
LC_NAME="ru_RU.UTF-8"
LC_ADDRESS="ru_RU.UTF-8"
LC_TELEPHONE="ru_RU.UTF-8"
LC_MEASUREMENT="ru_RU.UTF-8"
LC_IDENTIFICATION="ru_RU.UTF-8"
LC_ALL=ru_RU.UTF-8
```

1. Ввод/вывод. ASCII

Юнико́д^[1] (чаще всего) или **Унико́д** (англ. *Unicode*) — стандарт кодирования символов, включающий в себя знаки почти всех письменных языков мира. В настоящее время стандарт является преобладающим в Интернете.

Стандарт предложен в 1991 году некоммерческой организацией «Консорциум Юникода» (англ. *Unicode Consortium, Unicode Inc.*). Применение этого стандарта позволяет закодировать очень большое число символов из разных систем письменности: в документах, закодированных по стандарту Юникод, могут соседствовать китайские иероглифы, математические символы, буквы греческого алфавита, латиницы и кириллицы, символы музыкальной нотной нотации, при этом становится ненужным переключение кодовых страниц.

Стандарт состоит из двух основных частей: универсального набора символов (англ. *Universal character set, UCS*) и семейства кодировок (англ. *Unicode transformation format, UTF*). Универсальный набор символов перечисляет допустимые по стандарту Юникод символы и присваивает каждому символу код в виде неотрицательного целого числа, записываемого обычно в шестнадцатеричной форме с префиксом **U+**, например, **U+040F**. Семейство кодировок определяет способы преобразования кодов символов для передачи в потоке или в файле.

1. Ввод/вывод. ASCII

Диапазон номеров символов	Требуемое количество октетов
00000000-0000007F	1
00000080-000007FF	2
00000800-0000FFFF	3
00010000-0010FFFF	4

Символ		Двоичный код символа	UTF-8 в двоичном виде	UTF-8 в шестнадцатеричном виде
₪	U+0024	0100100	00100100	24
¢	U+00A2	10100010	11000010 10100010	C2 A2
€	U+20AC	100000 10101100	11100010 10000010 10101100	E2 82 AC
⌘	U+10348	1 00000011 01001000	11110000 10010000 10001101 10001000	F0 90 8D 88

A = U+0410 = 0100.0001.0000 = 010000.010000 = 11010000.10010000 = D0 90

1. Ввод/вывод. DOS

Функция DOS 09h: Записать строку в STDOUT с проверкой на **Ctrl-Break**

Вход: AH = 09h

DS:DX = адрес строки, заканчивающейся символом \$ (24h)

Выход: Никакого, согласно документации, но на самом деле:

AL = 24h (код последнего символа)

```
PRMP DB 'Строка','$'
```

```
.
```

```
MOV AH, 09          ; Запрос вывода на экран
```

```
LEA DX, PRMP        ; Загрузка адреса сообщ.
```

```
INT 21H             ; Вызов DOS
```

Функция DOS 06h: Записать символ в STDOUT без проверки на **Ctrl-Break**

Вход: AH = 06h

DL = ASCII-код символа (кроме 0FFh)

Выход: Никакого, согласно документации, но на самом деле:

AL = код записанного символа (копия DL)

Функция DOS 02h: Записать символ в STDOUT с проверкой на **Ctrl-Break**

Вход: AH = 02h

DL = ASCII-код символа

Выход: Никакого, согласно документации, но на самом деле:

AL = код последнего записанного символа (равен DL, кроме случая, когда DL = 09h (табуляция), тогда в AL возвращается 20h).

1. Ввод/вывод. DOS

NAMEPAR LABEL BYTE ; Список параметров:

MAXLEN DB 20 ; Максимальная длина

ACTLEN DB ? ; Реальная длина

NAMEFLD DB 20 DUP (' ') ; Введенные символы

MOV AH, 0AH ; Запрос функции ввода

LEA DX, NAMEPAR ; Адреса списка

параметров

INT 21H ; Вызвать DOS

1. Ввод/вывод. DOS

Функция DOS 01h: Считать символ из STDIN с эхом, ожиданием и проверкой на **Ctrl-Break**

Вход: AH = 01h

Выход: AL = ASCII-код символа или 0. Если AL = 0, второй вызов этой функции возвратит в AL расширенный ASCII-код символа.

Функция DOS 08h: Считать символ из STDIN без эха, с ожиданием и проверкой на **Ctrl-Break**

Вход: AH = 08h

Выход: AL = код символа

Функция DOS 07h: Считать символ из STDIN без эха, с ожиданием и без проверки на **Ctrl-Break**

Вход: AH = 07h

Выход: AL = код символа

Функция DOS 06h: Считать символ из STDIN без эха, без ожидания и без проверки на **Ctrl-Break**

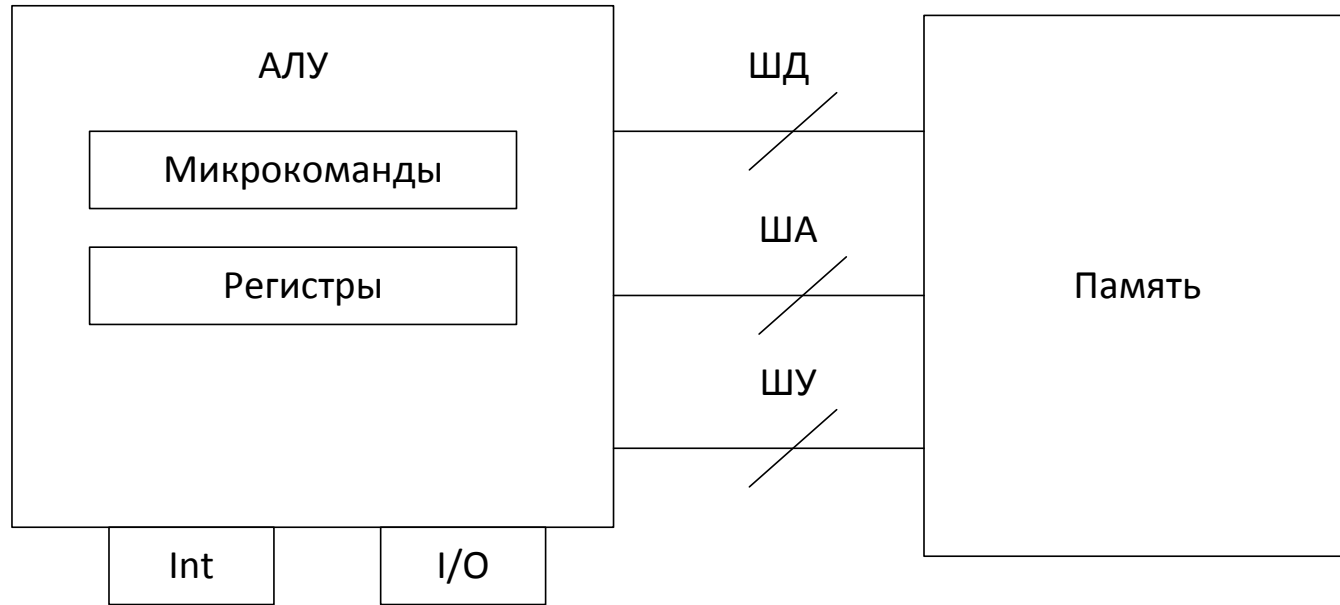
Вход: AH = 06h

DL = 0FFh

Выход: ZF = 1, если не была нажата клавиша, и AL = 00

ZF = 0, если клавиша была нажата. В этом случае AL = код символа

1. Ввод/вывод. DOS / BIOS



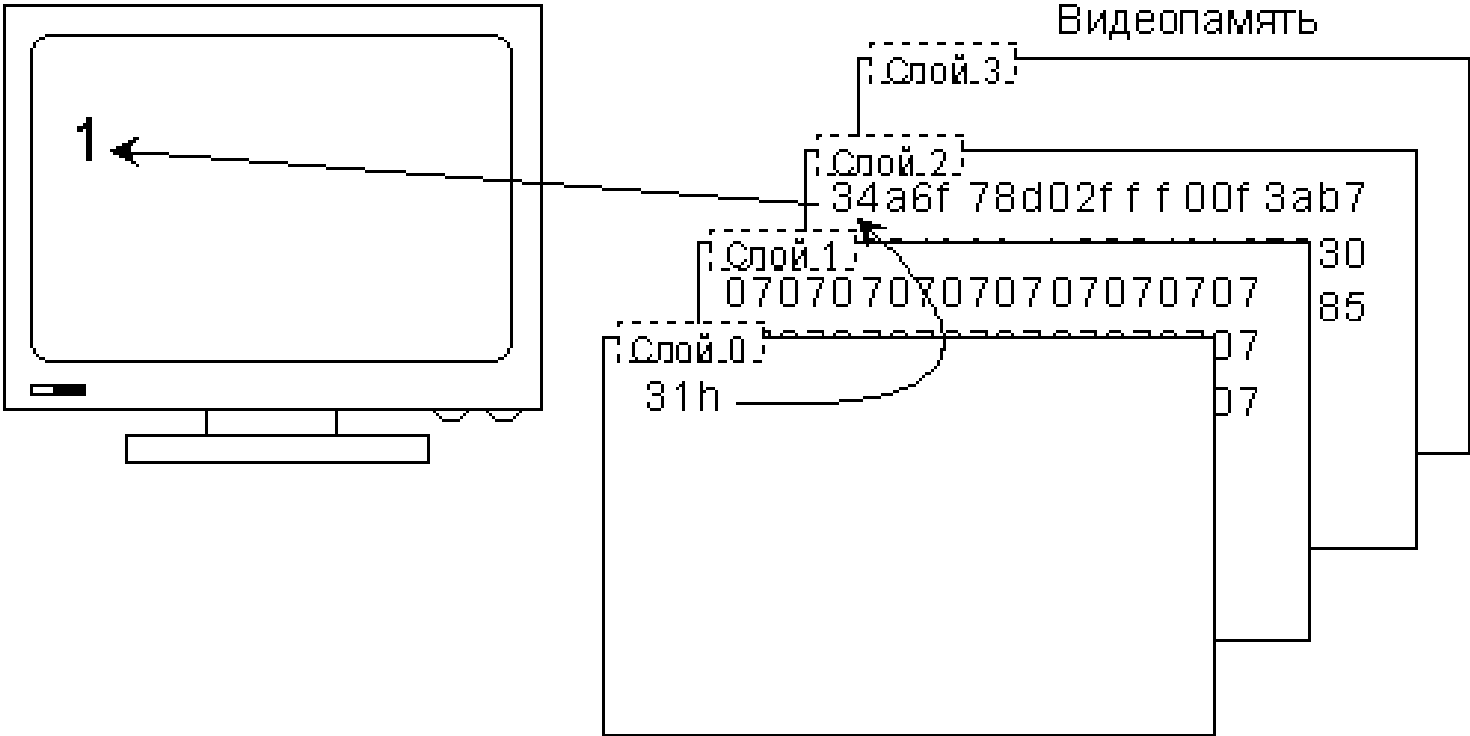
1. Ввод/вывод. DOS / BIOS

```
user@host: ~$ lspci -v
```

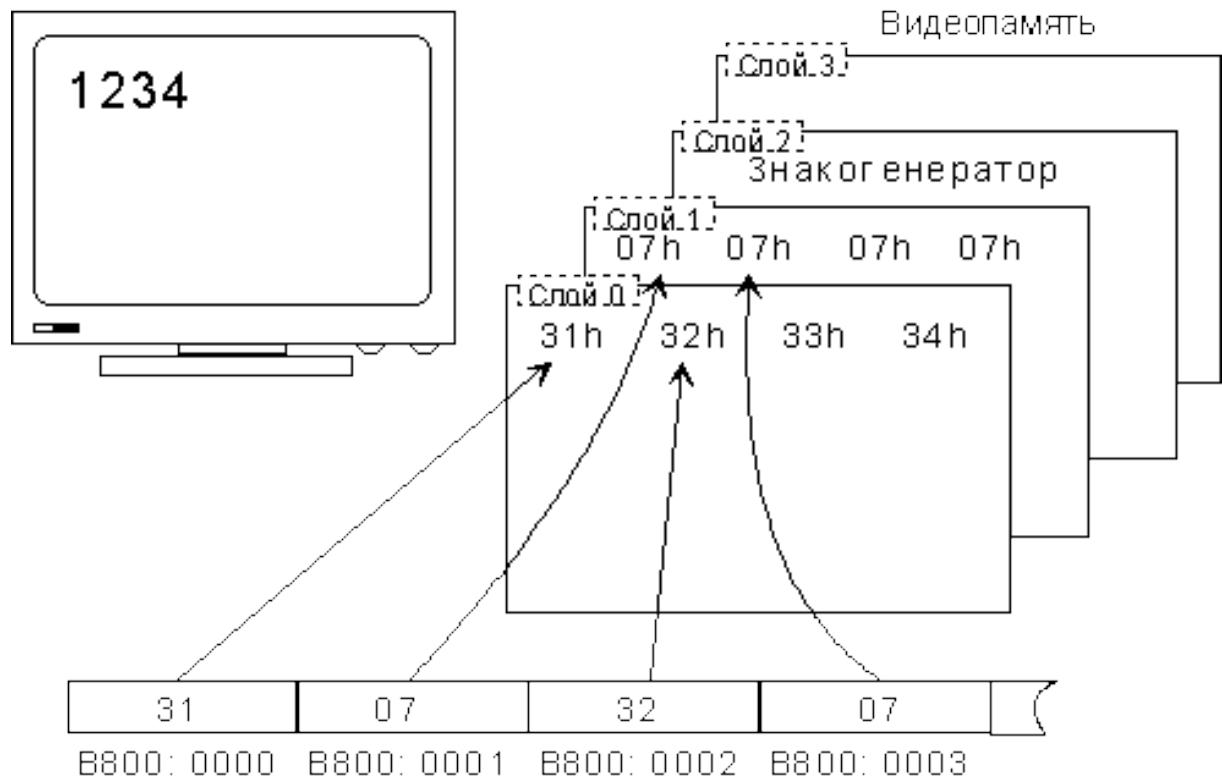
```
01:03.0 SATA controller: Device 1fff:801a (prog-if 01 [AHCI 1.0])
  Flags: bus master, medium devsel, latency 16, IRQ 20, NUMA node 0
  I/O ports at 1060 [size=8]
  I/O ports at 1070 [size=16]
  I/O ports at 1080 [size=8]
  I/O ports at 1090 [size=16]
  I/O ports at 10a0 [size=16]
  Memory at 8320a000 (32-bit, non-prefetchable) [size=8K]
  Kernel driver in use: ahci

01:03.1 SATA controller: Device 1fff:801a (prog-if 01 [AHCI 1.0])
  Flags: bus master, medium devsel, latency 16, IRQ 21, NUMA node 0
  I/O ports at 10b0 [size=8]
  I/O ports at 10c0 [size=16]
  I/O ports at 10d0 [size=8]
  I/O ports at 10e0 [size=16]
  I/O ports at 10f0 [size=16]
  Memory at 8320c000 (32-bit, non-prefetchable) [size=8K]
  Kernel driver in use: ahci
```

0xB800:0000 – 0xB800:FFFF



1. Ввод/вывод. DOS / BIOS



1. Ввод/вывод. DOS / BIOS

Адрес	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
B800:0000	91	07	E2	07	E0	07	AE	07-AA	07	A0	07	20	07	AD	07		С.т.р.о.к.а. .н.
B800:0010	AE	07	AC	07	A5	07	E0	07-20	07	30	07	20	07	20	07		о.м.е.р. .0. . .
B800:0020	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:0030	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:0040	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:0050	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:0060	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:0070	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:0080	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:0090	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:00A0	91	07	E2	07	E0	07	AE	07-AA	07	A0	07	20	07	AD	07		С.т.р.о.к.а. .н.
B800:00B0	AE	07	AC	07	A5	07	E0	07-20	07	31	07	20	07	20	07		о.м.е.р. .1. . .
B800:00C0	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:00D0	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:00E0	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:00F0	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:0100	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:0110	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:0120	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	
B800:0130	20	07	20	07	20	07	20	07-20	07	20	07	20	07	20	07	

1. Ввод/вывод. DOS / BIOS

INT 10h, AH = 02h: Установить положение курсора

Вход: AH = 02h
BH = номер страницы
DH = строка
DL = столбец

INT 10h, AH = 03h: Считать положение и размер курсора

Вход: AH = 03h
BH = номер страницы
Выход: DH, DL = строка и столбец текущей позиции курсора
CH, CL = первая и последняя строки курсора

1. Ввод/вывод. DOS / BIOS

INT 10h, AH = 08h. Считать символ и атрибут символа в текущей позиции курсора

Вход: AH = 08h

BH = номер страницы

Выход: AH = атрибут символа

AL = ASCII-код символа

INT 10h, AH = 09h. Вывести символ с заданным атрибутом на экран

Вход: AH = 09h

BH = номер страницы

AL = ASCII-код символа

BL = атрибут символа

CX = число повторений символа

Атрибут символа

- ☐ бит 7: символ мигает (по умолчанию) или фон яркого цвета (если его действие было переопределено видеофункцией 10h);
- ☐ биты 6-4: цвет фона;
- ☐ бит 3: символ яркого цвета (по умолчанию) или фон мигает (если его действие было переопределено видеофункцией 11h);
- ☐ биты 2-0: цвет символа.

1. Ввод/вывод. DOS / BIOS

INT 10h, AH = 13h: Вывести строку символов с заданными атрибутами

Вход: AH - 13h

AL = режим вывода:

бит 0: переместить курсор в конец строки после вывода

бит 1: строка содержит не только символы, но и атрибуты, так что
каждый символ описывается двумя байтами: ASCII-код
и атрибут

биты 2-7: зарезервированы

CX = длина строки (только число символов)

BL = атрибут, если строка содержит только символы

DH, DL = строка и столбец, начиная с которых будет выводиться строка

ES:BP = адрес начала строки в памяти

1. Ввод/вывод. DOS.Файлы

Функция DOS 3Ch: Создать файл

Вход: AH = 3Ch

CX = атрибут файла

бит 7: файл можно открывать разным процессам в Novell Netware

бит 6: не используется

бит 5: архивный бит (1, если файл не сохранялся)

бит 4: директория (должен быть 0 для функции 3Ch)

бит 3: метка тома (игнорируется функцией 3Ch)

бит 2: системный файл

бит 1: скрытый файл

бит 0: файл только для чтения

DS:DX = адрес ASCIZ-строки с полным именем файла (ASCIZ-строка
ASCII-символов, оканчивающаяся нулем)

Выход: CF = 0 и AX = идентификатор файла, если не произошла ошибка

CF = 1 и AX = 03h, если путь не найден

CF = 1 и AX = 04h, если слишком много открытых файлов

CF = 1 и AX = 05h, если доступ запрещен

Если файл уже существует, функция 3Ch все равно открывает его, присваивая ему нулевую длину. Чтобы этого не произошло, следует пользоваться функцией 5Bh.

0: STDIN - стандартное устройство ввода (обычно клавиатура);

1: STDOUT - стандартное устройство вывода (обычно экран);

2: STDERR - устройство вывода сообщений об ошибках (всегда экран);

3: AUX - последовательный порт (обычно COM1);

4: PRN - параллельный порт (обычно LPT1);

1. Ввод/вывод. DOS.Файлы

Функция DOS 3Dh: Открыть существующий файл

Вход: • AH = 3Dh

AL = режим доступа

бит 0: открыть для чтения

бит 1: открыть для записи

биты 2-3: зарезервированы (0)

биты 6-4: режим доступа для других процессов:

000: режим совместимости (остальные процессы также должны открывать этот файл в режиме совместимости)

001: все операции запрещены

010: запись запрещена

011: чтение запрещено

100: запрещений нет

бит 7: файл не наследуется порождаемыми процессами

DS:DX = адрес ASCII-строки с полным именем файла

CL = маска атрибутов файлов

Выход: CF = 0 и AX = идентификатор файла, если не произошла ошибка

CF = 1 и AX = код ошибки (02h - файл не найден, 03h - путь не найден, 04h - слишком много открытых файлов, 05h - доступ запрещен, 0Ch — неправильный режим доступа)

Функция DOS 5Bh: Создать и открыть новый файл

Вход: AH = 5Bh

CX = атрибут файла

DS:DX = адрес ASCII-строки с полным именем файла

Выход: CF = 0 и AX = идентификатор файла, открытого для чтения/записи
в режиме совместимости, если не произошла ошибка

CF = 1 и AX = код ошибки (03h - путь не найден, 04h - слишком много
открытых файлов, 05h - доступ запрещен, 50h - файл
уже существует)

1. Ввод/вывод. DOS.Файлы

Функция DOS 3Fh: Чтение из файла или устройства

Вход: AH = 3Fh

BX = идентификатор

CX = число байтов

DS:DX = адрес буфера для приема данных

Выход: CF = 0 и AX = число считанных байтов, если не было ошибки

CF = 1 и AX = 05h, если доступ запрещен, 06h, если неправильный идентификатор

Функция DOS 40h: Запись в файл или устройство

Вход: AH = 40h

BX = идентификатор

CX = число байтов

DS:DX = адрес буфера с данными

Выход: CF = 0 и AX = число записанных байтов, если не произошла ошибка

CF = 1 и AX = 05h, если доступ запрещен; 06h, если неправильный идентификатор

Функция DOS 42h: Переместить указатель чтения/записи

Вход: AH - 42h

BX = идентификатор

CX:DX = расстояние, на которое надо переместить указатель (со знаком)

AL = перемещение:

0 — от начала файла

1 - от текущей позиции

2 — от конца файла

Выход: CF = 0 и CX:DX = новое значение указателя (в байтах от начала файла),
если не произошла ошибка

CF = 1 и AX = 06h, если неправильный идентификатор

1. Ввод/вывод. DOS.Файлы

.model	tiny	
	.code	
	ORG 100h	;начало COM-файла
start:	MOV AH, 3Ch	;Создаем новый файл
	MOV CX, 0	;Атрибуты файла
	MOV DX, offset path	;Путь к файлу
	INT 21h	;Вызов функции 3Ch
	PUSH AX	;Помещаем в стек идентификатор файла
	MOV AX, 3D02h	;Открываем файл для записи
	MOV DX, offset path	;Путь к файлу
	INT 21h	;Вызов функции 3Dh
	MOV AH, 40h	;Записываем в файл
	POP BX	;Идентификатор файла
	MOV DX, offset path	;Адрес буфера с данными
	MOV CX, 29	;Будем записывать 29 байтов
	INT 21h	;Вызов функции 40h
	MOV AX, 3D00h	;Открываем файл для чтения
	MOV DX, offset path	;Путь к файлу
	INT 21h	;Вызов функции 3Dh
	PUSH AX	;Помещаем в стек идентификатор файла

1. Ввод/вывод. DOS.Файлы

```
MOV AH, 3Fh ;Читаем из файла
POP BX ;Идентификатор файла
MOV DX, offset buf ;Адрес буфера для данных
INT 21h
MOV AH, 3Eh ;Закрываем файл
INT 21h

;-----
; Этот участок кода добавляет в конец буфера с прочитанными данными
; символ $, для того, чтобы эту строку можно было вывести на экран
MOV DI, offset buf ;Адрес буфера с прочитанными данными
MOV BX, 29 ;BX = количеству символов в строке
MOV BYTE PTR [DI+BX], '$'

;-----
MOV AH, 9 ;Выводим строку, считанную из файла
MOV DX, offset buf ;на экран
INT 21h
RET

path DB 'C:\MyProg\TEST\F_DOS\file.txt', 0
buf DB ?
END start
```

```
user@host: ~$ man syscall
```

NAME

syscall - indirect system call

SYNOPSIS

```
#include <sys/syscall.h>    /* Definition of SYS_* constants */
```

```
#include <unistd.h>
```

```
long syscall(long number, ...);
```

1. Ввод/вывод. Linux

Arch/ABI	Instruction	System call #	Ret val	Ret val2	Error	Notes
i386	int \$0x80	eax	eax	edx	-	
x86-64	syscall	rax	rax	rdx	-	

Arch/ABI	arg1	arg2	arg3	arg4	arg5	arg6	arg7	Notes
i386	ebx	ecx	edx	esi	edi	ebp	-	
x86-64	rdi	rsi	rdx	r10	r8	r9	-	

1. Ввод/вывод. Linux

%eax	Имя системного вызова	%ebx	%ecx	%edx
2	sys_fork	struct pt_regs	—	—
3	sys_read	unsigned int	char *	size_t
4	sys_write	unsigned int	const char *	size_t
5	sys_open	const char *	int	int
6	sys_close	unsigned int	—	—
8	sys_creat	const char *	int	—
19	sys_lseek	unsigned int	off_t	unsigned int

<https://filippo.io/linux-syscall-table/>

Файловый дескриптор

Файловый дескриптор (или «дескриптор файла») — это 16-битное целое число, присваиваемое файлу в качестве идентификатора. Когда создается новый файл или открывается существующий, дескриптор файла используется для доступа к нему.

Файловый дескриптор стандартных файловых потоков:

`stdin` — 0;

`stdout` — 1;

`stderr` — 2.

Файловый указатель

Файловый указатель указывает местоположение для последующей операции чтения/записи в файл. Каждый файл рассматривается как последовательность байтов и ассоциируется с файловым указателем, который задает смещение в байтах относительно начала файла. Когда файл открыт, значением файлового указателя является 0.

Создание и открытие файла

Для создания и открытия файла выполняются следующие шаги:

- поместите системный вызов `sys_creat()` — номер 8 в регистр EAX;
- поместите имя файла в регистр EBX;
- поместите права доступа к файлу в регистр ECX.

Системный вызов возвращает файловый дескриптор созданного файла в регистр EAX. В случае ошибки, код ошибки также будет находиться в регистре EAX.

S_IRWXU (0x0700) – владелец имеет права R W X

S_IRUSR (0x0400) | S_IWUSR (0x0200) | S_IXUSR (0x0100)

S_IRWXG (0x0070) – группа имеет права R W X

S_IRWXO (0x0007) – другие имеют права R W X

```
int creat(const char *pathname, mode_t mode);
```

Открытие существующего файла

Для открытия существующего файла выполняются следующие шаги:

- поместите системный вызов `sys_open()` — номер 5 в регистр EAX;
- поместите имя файла в регистр EBX;
- поместите режим доступа к файлу в регистр ECX;
- поместите права доступа к файлу в регистр EDX.

Системный вызов возвращает файловый дескриптор открытого файла в регистр EAX.

В случае ошибки, код ошибки также будет находиться в регистре EAX.

Среди **режимов доступа к файлам** чаще всего используются:

- **O_RDONLY** только чтение (0);
- **O_WRONLY** только запись (1);
- **O_RDWR** чтение-запись (2).

```
int open(const char *pathname, int flags);  
int open(const char *pathname, int flags, mode_t mode);
```


Открытие существующего файла

Для открытия существующего файла выполняются следующие шаги:

- поместите системный вызов `sys_open()` — номер 5 в регистр EAX;
- поместите имя файла в регистр EBX;
- поместите режим доступа к файлу в регистр ECX;
- поместите права доступа к файлу в регистр EDX.

Системный вызов возвращает файловый дескриптор открытого файла в регистр EAX.

В случае ошибки, код ошибки также будет находиться в регистре EAX.

Среди **режимов доступа к файлам** чаще всего используются:

- **O_RDONLY** только чтение (0);
- **O_WRONLY** только запись (1);
- **O_RDWR** чтение-запись (2).

```
int open(const char *pathname, int flags);  
int open(const char *pathname, int flags, mode_t mode);
```

mode всегда должен быть указан при использовании **O_CREAT**; во всех остальных случаях этот параметр игнорируется. **creat** эквивалентен **open** с *flags*, которые равны **O_CREAT | O_WRONLY | O_TRUNC**.

Чтение файла

Для чтения данных из файла выполняются следующие шаги:

- поместите системный вызов `sys_read()` — номер 3 в регистр EAX;
- поместите файловый дескриптор в регистр EBX;
- поместите указатель на входной буфер в регистр ECX;
- поместите размер буфера, т.е. количество байтов для чтения, в регистр EDX.

Системный вызов возвращает количество прочитанных байтов в регистр EAX.

В случае ошибки, код ошибки также будет находиться в регистре EAX.

```
ssize_t read(int fd, void *buf, size_t count);
```

Запись в файл

Для записи в файл выполняются следующие шаги:

- поместите системный вызов `sys_write()` — номер 4 в регистр EAX;
- поместите файловый дескриптор в регистр EBX;
- поместите указатель на выходной буфер в регистр ECX;
- поместите размер буфера, т.е. количество байтов для записи, в регистр EDX.

Системный вызов возвращает фактическое количество записанных байтов в регистр EAX. В случае ошибки, код ошибки также будет находиться в регистре EAX.

```
ssize_t write(int fd, const void *buf, size_t count);
```

Заккрытие файла

Для закрытия файла выполняются следующие шаги:

- поместите системный вызов `sys_close()` — номер 6 в регистр EAX;
- поместите файловый дескриптор в регистр EBX.

В случае ошибки, системный вызов возвращает код ошибки в регистр EAX.

```
int close(int fd) ;
```

Перемещение по файлу

Для перемещения по файлу выполняются следующие шаги:

- поместите системный вызов `sys_lseek()` — номер 19 в регистр EAX;
- поместите файловый дескриптор в регистр EBX;
- поместите значение смещения в регистр ECX;
- поместите исходную позицию для смещения в регистр EDX.

Исходная позиция может быть:

- **SEEK_SET** началом файла — значение 0;
- **SEEK_CUR** текущей позицией — значение 1;
- **SEEK_END** концом файла — значение 2.

В случае ошибки, системный вызов возвращает код ошибки в регистр EAX.

```
off_t lseek(int fildes, off_t offset, int whence);
```

1. Ввод/вывод. Linux

section .data

file_name db 'myfile.txt'

msg db 'Welcome to Ravesli!'

len equ \$-msg

msg_done db 'Written to file', 0xa

len_done equ \$-msg_done

section .bss

fd_out resb 1

fd_in resb 1

info resb 26

1. Ввод/вывод. Linux

section .text

```
global _start ;
```

```
_start: ;
```

; Создаем файл

```
mov eax, 8 ;номер системного вызова (sys_creat)
```

```
mov ebx, file_name
```

```
mov ecx, 0777 ; читать, писать и выполнять могут все
```

```
int 0x80 ; вызов ядра
```

```
mov [fd_out], eax ; сохраняем дескриптор
```

; Записываем данные в файл

```
mov edx, len ; количество байтов
```

```
mov ecx, msg ; сообщение для записи в файл
```

```
mov ebx, [fd_out] ; файловый дескриптор
```

```
mov eax, 4 ; номер системного вызова (sys_write)
```

```
int 0x80 ; вызов ядра
```

1. Ввод/вывод. Linux

; Закрываем файл

```
mov eax, 6          ; sys_close()
```

```
mov ebx, [fd_out]
```

```
int 0x80            ; вызов ядра
```

; Выводим на экран сообщение, указывающее на конец записи в файл

```
mov eax, 4          ; sys_write()
```

```
mov ebx, 1          ; stdout
```

```
mov ecx, msg_done
```

```
mov edx, len_done
```

```
int 0x80
```


1. Ввод/вывод. Linux

; Открываем файл для чтения

```
mov eax, 5          ;sys_open()
mov ebx, file_name
mov ecx, 0          ; доступ "Только для чтения"
mov edx, 0777      ; читать, писать и выполнять могут все
int 0x80
mov [fd_in], eax
```

; указатель в начале файла!

; Считываем данные из файла

```
mov eax, 3          ;sys_read()
mov ebx, [fd_in]
mov ecx, info
mov edx, 26
int 0x80
```

1. Ввод/вывод. Linux

; Закрываем файл

```
mov eax, 6          ; sys_close()
```

```
mov ebx, [fd_in]
```

```
int 0x80
```

; Выводим на экран данные из буфера info

```
mov eax, 4          ; sys_write()
```

```
mov ebx, 1          ; stdout
```

```
mov ecx, info
```

```
mov edx, 26
```

```
int 0x80
```

```
mov eax, 1          ; номер системного вызова (sys_exit)
```

```
int 0x80            ; вызов ядра
```

Written to file

Welcome to Ravesli!

Спасибо