Машинно-зависимые языки программирования

Савельев Игорь Леонидович

План

- Арифметические операции
  - Перемещение данных
  - Сложение
  - Вычитание
  - Умножение
  - Деление

https://www.felixcloutier.com/x86/index.html

Видеопамять IBM PC

text/ grph	text resol	pixel box	pixel resolution	colors	disply pages	scrn addr	system
00h = T	40x25	8x8	320x200	16gray	8	**B800**	CGA,PCjr,Tandy
= T	40x25	8x14	320x350	16gray	8	B800	EGA
= T	40x25	8x16	320x400	16	8	B800	MCGA
= T	40x25	9x16	360x400	16	8	B800	VGA
01h = T	40x25	8x8	320x200	16	8	B800	CGA,PCjr,Tandy
= T	40x25	8x14	320x350	16	8	B800	EGA
= T	40x25	8x16	320x400	16	8	B800	MCGA
= T	40x25	9x16	360x400	16	8	B800	VGA
06h = G	80x25	8x8	640x200	2	.	B800	CGA,PCjr,EGA,MCGA,VGA
= G	80x25	.	.	mono	.	**B000**	HERCULES.COM on HGC
07h = T	80x25	9x14	720x350	mono	var	B000	MDA,Hercules,EGA
= T	80x25	9x16	720x400	mono	.	B000	VGA
0Dh = G	40x25	8x8	320x200	16	8	**A000**	EGA,VGA
0Eh = G	80x25	8x8	640x200	16	4	A000	EGA,VGA
0Fh = G	80x25	8x14	640x350	mono	2	A000	EGA,VGA

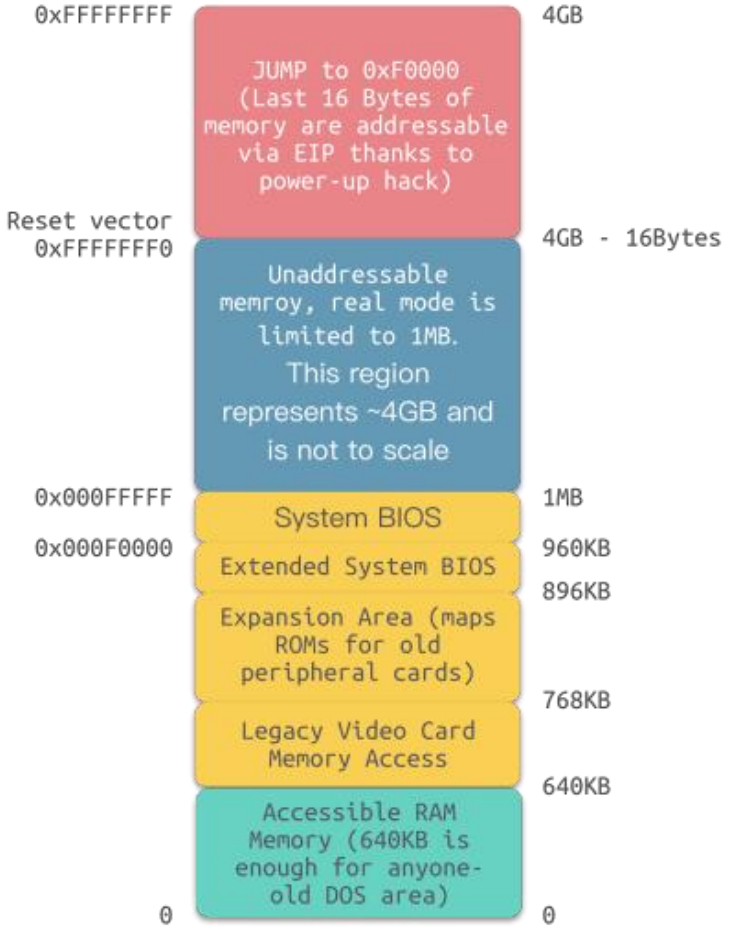
Старт программы на IBM PC

При включении

0xFFFFFFF0

Далее управление передается на

0xFFFF0000

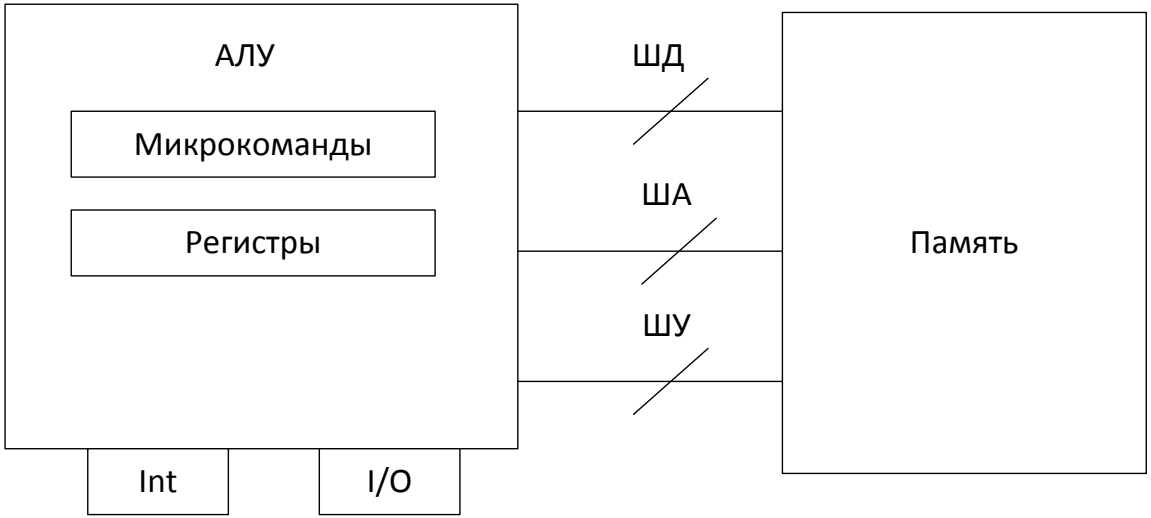


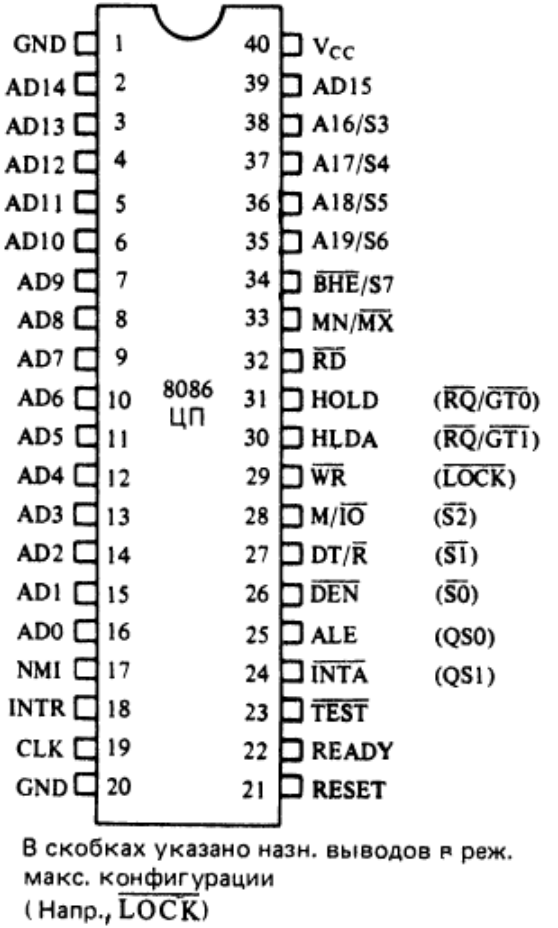
Порты ввода/вывода

```
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
```

Порты ввода/вывода





Порты ввода/вывода

IN reg, port

IN reg, DX

OUT port, reg

OUT DX, reg

Номер порта – [0, 255] или [0, 65535]

Размер данных (reg) – 8, 16, 32 бита

CPL <= IOPL

12 13	IOPL	I/O Privilege Level	Уровень приоритета ввода-вывода	Системный	80286

eflags	0x202	[IF]
cs	0x33	51

0x202 = 0000.0010.0000.0010

0x33 = 0000.0000.0011.0011

Операнды

Непосредственный (imm) – задается константой

mov ax, 3

mov 56, ax ; ошибка

Регистровый (reg) – задается именем регистра

mov ax, 3

mov bx, cx

Память (mem) – задается адресом в памяти, доступ по указателю

mov ax, [var]

mov [var], bx

Адресация в памяти

[**reg + reg*scale + const**]

reg – база, базовый адрес
req* - индекс
const – смещение
scale – 1, 2, 4, 8

Общий вид исполнительного адреса

Адресация в памяти

```
[750]                   ; displacement only
[rbp]                   ; base register only
[rcx + rsi*4]           ; base + index * scale
[rbp + rdx]             ; scale is 1
[rbx - 8]               ; displacement is -8
[rax + rdi*8 + 500]     ; all four components
[rbx + counter]         ; uses the address of the variable
                        ;'counter' as the displacement
```

Директивы

```
MAGIC_NUM   equ   22
```

```
%define MAGIC_NUM   22
```

```
%define thricexplusy(x,y) (3 * x + y)
```

*Прим: структуры «**Следование**», «Ветвление», «Цикл»*

Инструкции следуют в памяти и выполняются последовательно

Инструкции (операции)

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
void good() {printf("good\n");};
void bad() {printf("bad\n");};
int main (void)
{
    int a = 0;
    scanf("%d", &a);
    if (a == 0)
       bad();
    else
       good();

    return 0;
}
```

Инструкции (операции)

```
    if (a == 0)
    119f:       8b 45 fc                mov    -0x4(%rbp),%eax
    11a2:       85 c0                   test   %eax,%eax
    11a4:       75 0c                   jne    11b2 <main+0x3d>
       bad();
    11a6:       b8 00 00 00 00          mov    $0x0,%eax
    11ab:       e8 af ff ff ff          call   115f <bad>
    11b0:       eb 0a                   jmp    11bc <main+0x47>
    else
       good();
    11b2:       b8 00 00 00 00          mov    $0x0,%eax
    11b7:       e8 8d ff ff ff          call   1149 <good>

    return 0;
    11bc:       b8 00 00 00 00          mov
}
```

objdump –S ctest.exe

hexdump –C ctest.exe

```
00001190  00 00 48 89 c7 b8 00 00  00 00 e8 a1 fe ff ff 8b
000011a0  45 fc 85 c0 75 0c b8 00  00 00 00 e8 af ff ff ff
000011b0  eb 0a b8 00 00 00 00 e8  8d ff ff ff b8 00 00 00
000011c0  00 c9 c3 00 48 83 ec 08  48 83 c4 08 c3 00 00 00
000011d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
*
00002000  01 00 02 00 67 6f 6f 64  00 62 61 64 00 25 64 00
00002010  01 1b 03 3b 3c 00 00 00  06 00 00 00 10 f0 ff ff
00002020  88 00 00 00 40 f0 ff ff  b0 00 00 00 50 f0 ff ff
```

NOP

Размер – 1 байт
0x90

Выполняется – 1 такт

Зачем это?!

Инструкции (операции)

```
OUT  35,  ax
NOP            ; ожидание 3 такта
NOP
NOP
IN ax, 35
```

Инструкции (операции)

```
    if (a == 0)
    119f:       8b 45 fc                mov    -0x4(%rbp),%eax
    11a2:       85 c0                   test   %eax,%eax
    11a4:       75 0c                   jne    11b2 <main+0x3d>
       bad();
    11a6:       b8 00 00 00 00          mov    $0x0,%eax
    11ab:       e8 af ff ff ff          call   115f <bad>
    11b0:       eb 0a                   jmp    11bc <main+0x47>
    else
       good();
    11b2:       b8 00 00 00 00          mov    $0x0,%eax
    11b7:       e8 8d ff ff ff          call   1149 <good>

    return 0;
    11bc:       b8 00 00 00 00          mov    $0x0,%eax
}
```

Инструкции (операции)

```
    if (a == 0)
    119f:       8b 45 fc                mov    -0x4(%rbp),%eax
    11a2:       90 90                   test   %eax,%eax
    11a4:       90 90                   jne    11b2 <main+0x3d>
       bad();
    11a6:       90 90 90 90 90          mov    $0x0,%eax
    11ab:       90 90 90 90 90          call   115f <bad>
    11b0:       90 90                   jmp    11bc <main+0x47>
    else
       good();
    11b2:       b8 00 00 00 00          mov    $0x0,%eax
    11b7:       e8 8d ff ff ff          call   1149 <good>

    return 0;
    11bc:       b8 00 00 00 00          mov    $0x0,%eax
}
```

2. Арифметические операции. Общие сведения

Таблица П.1.1

Группа команд	№ п.п.	Мнемокод	n_T	Байты формата команды: нечетный	Байты формата команды: четный	Описание команды
Пересылки данных	1	MOV r , r	2	1 0 0 0 1 0 d w	mod reg r/m	Рг←Рг
	2	MOV r , mem	8+*E*			Рг←П
	3	MOV mem , r	9+*E*			П←Рг
	4	MOV mem , data	10+*E*	1 1 0 0 0 1 1 w data L	mod 0 0 0 r/m data H(w=1)	П←Д
	5	MOV r , data	4	1 0 1 1 w reg data H(w=1)	data L	Рг←Д
	6	MOV a , mem	10	1 0 1 0 0 0 d w addr H	addr L	А←П
	7	MOV mem , a	10			П←А
	8	MOV seg , r	2	1 0 0 0 1 1 d 0	mod 0 seg r/m	Сегментный Рг←Рг
	9	MOV seg , mem	8+*E*			Сегментный Рг←П
	10	MOV r , seg	2			Рг←сегментный Рг
	11	MOV mem , seg	9+*E*			П←сегментный Рг
	12	PUSH r	10	1 1 1 1 1 1 1 1	mod 1 1 0 r/m	Стек←Рг
	13	PUSH mem	16+*E*			Стек←П
	14	PUSH r	10	0 1 0 1 0 reg		Стек←Рг
	15	PUSH seg	10	0 0 0 seg 1 1 0		Стек←сегментный Рг
	16	POP r	8	1 0 0 0 1 1 1 1	mod 0 0 0 r/m	Рг←стек
	17	POP mem	17+*E*			П←стек
	18	POP r	8	0 1 0 1 1 reg		Рг←стек
	19	POP seg	8	0 0 0 seg 1 1 1		Сегментный Рг←стек
	20	XCHG r , mem	17+*E*	1 0 0 0 0 1 1 w	mod reg r/m	Рг←П
	21	XCHG r , r	4			Рг←Рг
	22	XCHG AX , r	3	1 0 0 1 0 reg		А←Рг
	23	IN port	10	1 1 1 0 0 1 0 w	port	А←порт
	24	IN	8	1 1 1 0 1 1 0 w		А←(*DX*)
	25	OUT port	10	1 1 1 0 0 1 1 w	port	Порт←А
	26	OUT	8	1 1 1 0 1 1 1 w		(*DX*)←А

Таблица П.1.2

Группы команд	Команды	Флаги состояния: OF	CF	AF	SF	ZF	PF
Сложение и вычитание	ADD ADC SUB SBC	+	+	+	+	+	+
	CMP NEG CMPS SCAS	+	+	+	+	+	+
	INC DEC	+	—	+	+	+	+
Умножение и деление	MUL IMUL	?	?	?	?	?	?
	DIV IDIV	?	?	?	?	?	?
Десятичная коррекция	DAA DAS	?	+	+	+	+	+
	AAA AAS	?	?	?	?	?	?
	AAM AAD	?	?	?	+	+	+
Логические операции	AND OR XOR TEST	0	0	?	+	+	+
Сдвиги:							
одиночный	SHL SHR	+	+	?	+	+	+
многоразрядный	SHL SHR	?	+	?	+	+	+
	SAR	0	+	?	+	+	+
одиночный	ROL ROR RCL RCR	+	+	—	—	—	—
многоразрядный	ROL ROR RCL RCR	?	+	—	—	—	—
Восстановление флагов	POPF IRET	+	+	+	+	+	+
	SAHF	—	+	+	+	+	+
Управление флагом переноса	STC	—	1	—	—	—	—
	CLC	—	0	—	—	—	—
	CMC	—	*	—	—	—	—

Группы команд	Команды	Флаги управления: DF	IF	TF
Восстановление флагов	POPF IRET	*	+	+
Прерывание	INT INTO	—	0	0
Управление флагами	STD	1	—	—
	CLD	0	—	—
Управление флагами	STI	—	1	—
	CLI	—	0	—

Примечание. + — влияет на флаг, 0 — сбрасывает в «0», 1 — устанавливает в «1», ? — не определено, * — инвертирует, — — не влияет.

2. Арифметические операции. Перемещение данных

- MOV
- POP/PUSH,…
- XCNG
- LEA,…

2. Арифметические операции. Перемещение данных

MOV r , r	2	1 0 0 0 1 0 d w	mod reg r/m	Рг←Рг
MOV r , mem	8+*E*			Рг←П
MOV mem , r	9+*E*			П←Рг
MOV mem , data	10+*E*	1 1 0 0 0 1 1 w data L	mod 0 0 0 r/m data H(w=1)	П←Д
MOV r , data	4	1 0 1 1 w reg data H(w=1)	data L	Рг←Д
MOV a , mem	10	1 0 1 0 0 0 d w addr H	addr L	А←П
MOV mem , a	10			П←А

MOV seg , r	2	1 0 0 0 1 1 d 0	mod 0 seg r/m	Сегментный Рг←Рг
MOV seg , mem	8+*E*			Сегментный Рг←П
MOV r , seg	2			Рг←сегментный Рг
MOV mem , seg	9+*E*			П←сегментный Рг

2. Арифметические операции. Перемещение данных

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
88 /*r*	MOV *r/m8,r8*	MR	Valid	Valid	Move *r8* to *r/m8.*
REX + 88 /*r*	MOV *r/m8***,r8****	MR	Valid	N.E.	Move *r8* to *r/m8.*
89 /*r*	MOV *r/m16,r16*	MR	Valid	Valid	Move *r16* to *r/m16.*
89 /*r*	MOV *r/m32,r32*	MR	Valid	Valid	Move *r32* to *r/m32.*
REX.W + 89 /*r*	MOV *r/m64,r64*	MR	Valid	N.E.	Move *r64* to *r/m64.*
8A /*r*	MOV *r8,r/m8*	RM	Valid	Valid	Move *r/m8* to *r8.*
REX + 8A /*r*	MOV *r8***,r/m8****	RM	Valid	N.E.	Move *r/m8* to *r8.*
8B /*r*	MOV *r16,r/m16*	RM	Valid	Valid	Move *r/m16* to *r16.*
8B /*r*	MOV *r32,r/m32*	RM	Valid	Valid	Move *r/m32* to *r32.*
REX.W + 8B /*r*	MOV *r64,r/m64*	RM	Valid	N.E.	Move *r/m64* to *r64.*

```
section .data
        ch1 db 0x11;        uint8_t ch1 = 0x11          void * ch1
        ch2 db 0x22;        uint8_t ch2 = 0x22          void * ch2
        sh1 dw 0xAABB;      uint16_t sh1 = 0xAABB       void * sh1
        myint  dd 0;                                    void * myint
section .text
global CMAIN
CMAIN:
        xor eax, eax
        mov al, [ch1]     ; uint8_t al =     *(uint8_t*)ch1;   al = 0x11
        mov ax, [ch1]     ; uint16_t ax =    *(uint16_t*)ch1;  ax = 0x2211
        mov eax, [ch1]    ; uint32_t eax = *(uint32_t*)ch1;  eax = 0xAABB2211
        mov [myint], eax     ; *myint = 0xAABB2211
        mov eax, ch1      ; uint32_t eax = ch1;  eax = 0x600042 – адрес!!!

        mov ax, 3
```

```
section .data
    myint dd 0x11223344
section .text
global CMAIN
CMAIN:
    xor eax, eax
    mov ax, [myint]             ; ax == 0x3344
    mov bx, [myint + 2]         ; bx == 0x1122
    mov cl, [myint]             ; cl == 0x44
    mov ch, [myint + 1]         ; ch == 0x33
    mov dl, [myint + 2]         ; dl == 0x22
    mov dh, [myint + 3]         ; dh == 0x11
    ret
```

AX < -- > BX

Байты:такты

```
MOV var, AX
MOV AX, BX
MOV BX, var
```

```
3: 10
2: 2
3: 8+E
```

8: 20+E

```
MOV CX, AX
MOV AX, BX
MOV BX, CX
```

```
2: 2
2: 2
2: 2
```

6: 6

2. Арифметические операции. Перемещение данных

PUSH r	10	1 1 1 1 1 1 1 1	mod 1 1 0 r/m	Стек←Pr
PUSH mem	16+*E*			Стек←П
PUSH r	10	0 1 0 1 0 reg		Стек←Pr
PUSH seg	10	0 0 0 seg 1 1 0		Стек←сегментный Pr
POP r	8	1 0 0 0 1 1 1 1	mod 0 0 0 r/m	Pr←стек
POP mem	17+*E*			П←стек
POP r	8	0 1 0 1 1 reg		Pr←стек
POP seg	8	0 0 0 seg 1 1 1		Сегментный Pr←стек

PUSHF	10	1 0 0 1 1 1 0 0		Стек←*F*
POPF	8	1 0 0 1 1 1 0 1		*F*←стек

```
section .data
        a dq 1
        b dq 2
section .text
global CMAIN
CMAIN:
        push qword [a]
        pop qword [b]
        ret
```

AX < -- > BX

Байты:такты

```
PUSH AX
MOV AX, BX
POP BX
```

1: 10
2: 2
1: 8

4: 20

AX < -- > BX

Байты:такты

PUSH AX MOV AX, BX POP BX	1: 10 2: 2 1: 8	4: 20

MOV var, AX
MOV AX, BX
MOV BX, var
	3: 10
	2: 2
	3: 8+E
		8: 20+E

MOV CX, AX
MOV AX, BX
MOV BX, CX
	2: 2
	2: 2
	2: 2
		6: 6

2. Арифметические операции. Перемещение данных

XCHG r , mem	17+*E*	1 0 0 0 0 1 1 w	mod reg r/m	Pr←П
XCHG r , r	4			Pr←Pr
XCHG AX , r	3	1 0 0 1 0 reg		A←Pr

MOV r , r	2	1 0 0 0 1 0 d w	mod reg r/m	Pr←Pr

```
MOV  AL, 1
MOV  AH, 2
XCHG AL, AH  ;Теперь AL=2, AH=1
XCHG AL, AH  ;Теперь AL=1, AH=2
```

2. Арифметические операции. Перемещение данных

LEA r, mem	2+*E*	10001101	mod reg r/m	Pr←*EA*
LDS r, mem	16+*E*	11000101	mod reg r/m	Pr и *DS*←П
LES r, mem	16+*E*	11000100	mod reg r/m	Pr и *ES*←П

LAHF	4	10011111		*AH*←*FL*
SAHF	4	10011110		*FL*←*AH*

PUSHF	10	10011100		Стек←*F*
POPF	8	10011101		*F*←стек

2. Арифметические операции. Перемещение данных

```
char  var1 = 4;
short var2 = va1; ????
```

```
CBW       ; ax <- al
CWD       ; dx:ax <- ax
CDQ       ; edx:eax <- eax

CWDE      ; eax <- ax
CDQE      ; rax <- eax
```

```
MOVZX     ; для беззнаковых, заполнение 0

MOVSX     ; для знаковых, учет знака
```

2. Арифметические операции. Преобразование операнда

Для ЗНАКОВЫХ данных существуют две команды распространения знака.

CBW (Convert Byte toWord — преобразовать байт, находящийся в регистре AL, в слово — регистр AX) и

CWD (Convert Word to Double word — преобразовать слово, находящееся в регистре AX, в двойное слово — регистры < DX:AX >).

Синтаксис: CBW
CWD

Операнды им НЕ нужны.

CBW

Получаем старшую часть (AH)	**Известная младшая часть (AL)**	
Биты 15 - 8	**Знак-бит 7**	**Биты 6 - 0**
1111 1111	1	Информационная часть числа
0000 0000	0	Информационная часть числа
Результат AX		

CWD

старшую часть (DX)	**Известная младшая часть (AX)**	
Биты 31 - 16	**Знак-бит 15**	**Биты 14 - 0**
1111 1111 1111 1111	1	Информационная часть числа
0000 0000 0000 0000	0	Информационная часть числа
Результат DX:AX		

2. Арифметические операции. Преобразование операнда

Применение:

Команда CBW используется для приведения операндов к нужной размерности с учетом знака. Такая необходимость может, в частности, возникнуть при программировании арифметических операций.

Команда CWD используется для расширения значения знакового бита в регистре ax на биты регистра dx. Данную операцию, в частности, можно использовать для подготовки к операции деления, для которой размер делимого должен быть в два раза больше размера делителя, либо для приведения операндов к одной размерности в командах умножения, сложения, вычитания.

2. Арифметические операции. Преобразование операнда

MOVSX – позволяет преобразовать значение в памяти или в регистре и присвоить в другой регистр увеличенного размера с сохранением знака

```
movsx    RAX,  AL

movsx  EDX, [var] ; ошибка – не определен размер
movsx  EDX, byte [var] ;
```

MOVZX – позволяет преобразовать значение в памяти или в регистре и присвоить в другой регистр увеличенного размера без сохранения знака

```
movzx    RAX,  AL

movzx  EDX, [var] ; ошибка – не определен размер
movzx  EDX, byte [var] ;
```

1. Арифметические операции. Перемещение данных

```
section .data
   b db 255
   c db 255
section .text
global CMAIN
CMAIN:
    movzx  bx, [c]          ; bx = c
    movzx  ax, [b]          ; ax = b

   mov al, [c]              ; тоже самое
   cbw
   mov bx, ax
   mov al, [b]
   cbw

   ret
```

5. Арифметические операции. Перемещение данных

```
section .data
   b db  55
   c db -10
section .text
global CMAIN
CMAIN:
    movsx  bx, [c]          ; bx = c
    movzx  ax, [b]          ; ax = b

   mov al, [c]
   cbw
   mov bx, ax
   mov al, [b]
   cbw

   ret
```

2. Арифметические операции. Сложение

Команды сложения ADD, ADC, INC.

Командам **БЕЗРАЗЛИЧНО** какие числа складываются (**знаковые или нет**).

Если в результате сложения результат **НЕ** поместился в отведенное место, устанавливается флаг переноса **CF=1**. Команда **ADC** как раз и реагирует на этот флаг. Вырабатываются еще **4** флага: **PF**, **SF**, **ZF**, **OF**.

2. Арифметические операции. Сложение

Команды сложения ADD, ADC, INC.

Флаг	Пояснение
CF=1	Результат сложения НЕ поместился в операнде-приемнике
PF=1	Результат сложения имеет четное число бит со значением 1
SF=1	Копируется СТАРШИЙ (ЗНАКОВЫЙ) бит результата сложения
ZF=1	Результат сложения равен НУЛЮ
OF=1	Если при сложении двух чисел ОДНОГО знака результат сложения получился БОЛЬШЕ допустимого значения. В этом случае приемник МЕНЯЕТ ЗНАК.

2. Арифметические операции. Сложение

Команда **ADD** (ADDition)

Назначение: сложение двух операндов источник и приемник размерностью байт, слово или двойное слово.

Синтаксис: ADD Приемник, Источник

Алгоритм работы:

- сложить операнды источник и приемник;
- записать результат сложения в приемник;
- установить флаги.

Логика работы:<Приемник> = < Приемник> + <Источник>

Применение: Команда **add** используется для сложения двух целочисленных операндов. Результат сложения помещается по адресу первого операнда. Если результат сложения выходит за границы операнда приемник (возникает переполнение), то учесть эту ситуацию следует путем анализа флага **cf** и последующего возможного применения команды **adc**.

2. Арифметические операции. Сложение

Команда ADC (Addition with Carry)

Назначение: сложение двух операндов с учетом переноса из младшего разряда.

Синтаксис: ADC Приемник, Источник

Алгоритм работы:

- сложить два операнда;
- поместить результат в первый операнд;
- в зависимости от результата установить флаги.

Логика работы: <Приемник> = <Приемник> + <Источник> + < CF >

Применение: Команда **adc** используется при сложении длинных двоичных чисел. Ее можно использовать как самостоятельно, так и совместно с командой **add**. При совместном использовании команды **adc** с командой **add** сложение младших байтов/слов/двойных слов осуществляется командой **add**, а уже старшие байты/слова/двойные слова складываются командой **adc**, учитывающей переносы из младших разрядов в старшие. Таким образом, команда **adc** значительно расширяет диапазон значений складываемых чисел.

5. Арифметические операции. Сложение

a = b + c
a = 10 + 22 = 32 (0x20)

a = b + c
al == al + bl

```
section .data
   b db 10
   c db 22
   a db 0
section .text
global CMAIN
CMAIN:
    mov  bl, [c]          ; bl = c
    mov  al, [b]          ; al = b
    add al, bl            ; al = al + bl =b + c
    mov [a], al           ; a = b + c
ret
```

5. Арифметические операции. Сложение

a = b + c
a = 255 + 255 = 510 (0x01FE)

a = b + c
ax == ax + bx

```
section .data
   b db 255
   c db 255
   a dw 0
section .text
global CMAIN
CMAIN:
    movzx  bx, [c]          ; bx = c
    movzx  ax, [b]          ; ax = b
    add ax, bx                          ; ax = ax + bx =b + c
    mov [a], ax             ; a = b + c
ret
```

2. Арифметические операции. Сложение

```
                ;x1=a1+b1
section .DATA
extern x1  ; dw
extern a1  ; dw
extern b1  ;dw

section .text

addaL :
                MOV AX, [a1]          ; ax <== мл. часть a1
                MOV BX, [a1+2]        ; bx <== ст. часть a1
                MOV CX, [b1]          ; cx <== мл. часть b1
                MOV DX, [b1 + 2]      ; dx <== ст. часть b1
                ADD AX, CX                    ; < ax >:=< ax >+< cx > мл. часть
                ADC BX, DX                    ; < bx >:=< bx >+< dx >+< CF >      ст. часть
                MOV [x1], AX          ; мл. часть x1 <== < ax >
                MOV [x1+2], BX        ; ст. часть x1 <== < bx >
RET
```

2. Арифметические операции. Сложение

Команда **INC** (INCrement operand by 1)

Назначение: увеличение значения операнда в памяти или регистре на 1.

Синтаксис: INC Операнд

Алгоритм работы: команда увеличивает операнд на единицу.

Логика работы: < Операнд > = < Операнд > + 1

Применение: Команда используется для увеличения значения байта, слова, двойного слова в памяти или регистре на единицу. При этом команда не воздействует на флаг **cf**.

INC AX ; увеличить значение в ax на 1

2. Арифметические операции. Сложение

ADD r1 , r2	3	0 0 0 0 0 0 d w	mod reg r/m	Pr←Pr+Pr
ADD r , mem	9+*E*			Pr←Pr+П
ADD mem , r	16+*E*			П←Pr+Pr
ADD r , data	4	1 0 0 0 0 0 s w	mod 0 0 0 r/m	Pr←Pr+Д
		data L	data H(sw=01)	
ADD mem , data	17+*E*			П←П+Д
ADD a , data	4	0 0 0 0 0 1 0 w	data L	A←A+Д
		data H(w=1)		
ADC r1 , r2	3	0 0 0 1 0 0 d w	mod reg r/m	Pr←Pr+Pr+*CF*
ADC r , mem	9+*E*			Pr←Pr+П+*CF*
ADC mem , r	16+*E*			П←П+Pr+*CF*
ADC r , data	4	1 0 0 0 0 0 sw	mod 0 1 0 r/m	Pr←Pr+Д+*CF*
ADC mem , data	17+*E*	data L	data H(sw=01)	П←П+Д+*CF*
ADC a , data	4	0 0 0 1 0 1 0 w	data L	A←A+Д+*CF*
		data H(w=1)		
INC r	2	1 1 1 1 1 1 1 w	mod 0 0 0 r/m	Pr←Pr+1
INC mem	15+*E*			П←П+1
INC r	2	0 1 0 0 0 reg		Pr←Pr(8)+1

2. Арифметические операции. Вычитание

Команды вычитания **SUB, SBB, DEC** ОБРАТНЫ соответствующим командам сложения ADD, ADC и INC.
Они имеют те же самые операнды.

SUB (SUBtract — Вычитание).

SBB (SuBtract with Borrow CF — Вычитание с заемом флага переноса CF).

DEC (DECrement operand by 1 — Уменьшение значения операнда на 1).

5. Арифметические операции. Вычитание

a = b - c

a = -100 - 100 = -200 (0xFF38)

a = b - c

ax == ax - bx

```
section .data
  b db -100
  c db 100
  a dw 0
section .text
global CMAIN
CMAIN:
   mov al, [c]
   cbw                  ; ax = c
   mov bx, ax           ; bx = c
   mov al, [b]
   cbw                  ; ax = b
   sub ax, bx           ; ax = ax - bx =b - c
   mov [a], ax          ; a = b - c
ret
```

2. Арифметические операции. Вычитание

```
                ; x1=a1-b1
section .DATA
extern x1  ; dw
extern a1  ; dw
extern b1  ;dw
section .text
suba:
                MOV AX, [a1]          ; ax <== мл. часть a1
                MOV BX, [a1+2]        ; bx <== ст. часть a1
                MOV CX, [b1]          ; cx <== мл. часть b1
                MOV DX, [b1 + 2]      ; dx <== ст. часть b1
                SUB AX, CX
                SBB BX, DX
                MOV [x1], AX          ; мл. часть x1 <== < ax >
                MOV [x1+2], BX        ; ст. часть x1 <== < bx >
ret
```

2. Арифметические операции. Вычитание

Команда **NEG** (NEGate operand)

Назначение: изменение знака (получение двоичного дополнения) источника.

Синтаксис: NEG Операнд

Алгоритм работы:

- выполнить вычитание (0 – источник) и поместить результат на место источника;
- если источник = 0, то его значение не меняется.

Логика работы: < Операнд > = - < Операнд >

Применение: Команда используется для формирования двоичного дополнения операнда в памяти или регистре. Операция двоичного дополнения предполагает инвертирование всех разрядов операнда с последующим сложением операнда с двоичной единицей. Если операнд отрицательный, то операция **NEG** над ним означает получение его модуля.

```
MOV AL,2
NEG AL      ; al=0feh — число -2 в дополнительном коде
```

SUB r1 , r2 SUB r , mem	3 9+*E*	0 0 1 0 1 0 d w	mod reg r/m	Pr←Pr—Pr Pr←Pr—П

SUB mem , r	16+*E*			П←П—Pr
SUB r , data	4	1 0 0 0 0 0 s w data L	mod 1 0 1 r/m data (sw = 01)	Pr←Pr—Д
SUB mem , data	17+*E*			П←П—Д
SUB a , data	4	0 0 1 0 1 1 0 w data H (w = 1)	data L	A←A—Д
SBB r1 , r2	3	0 0 0 1 1 0 d w	mod reg r/m	Pr←Pr—Pr—*CF*
SBB r , mem	9+*E*			Pr←Pr—П—*CF*
SBB mem , r	16+*E*			П←П—Pr—*CF*
SBB r , data	4	1 0 0 0 0 0 s w data L	mod 0 1 1 r/m data H (sw = 01)	Pr←Pr—Д—*CF*
SBB mem , data	17+*E*			П←П—Д—*CF*
SBB a , data	4	0 0 0 1 1 1 0 w data H (w = 1)	data L	A←A—Д
DEC r	2	1 1 1 1 1 1 1 w	mod 0 0 1 r/m	Pr←Pr—1
DEC mem	155+*E*			П←П—1
DEC r	2	0 1 0 0 1 reg		Pr←Pr (8) —1
NEG r	3	1 1 1 1 0 1 1 w	mod 0 1 1 r/m	Pr←0—Pr
NEG mem	16+*E*			П←0—П

MOV AX, 0

MOV r , data	4	1 0 1 1 w reg data H(w = 1)	data L	Pr←Д

SUB AX, AX

SUB r1 , r2 SUB r , mem	3 9+*E*	0 0 1 0 1 0 d w	mod reg r/m	Pr←Pr—Pr Pr←Pr—П

2. Арифметические операции. Преобразование операнда

Преобразование байта в слово и слова в двойное слово

Данное преобразование для знаковых и беззнаковых данных осуществляется по разному.

БЕЗЗНАКОВЫЕ числа занимают всю ячейку памяти, понятие знак для них НЕ существует - они считаются ПОЛОЖИТЕЛЬНЫМИ. Поэтому при преобразовании БЕЗЗНАКОВЫХ чисел в СТАРШУЮ часть результата надо занести НОЛЬ. Это можно сделать уже известными нам командами: MOV AH,0 или MOV DX,0. Однако это НЕ эффективно, используем родную для компьютера команду вычитания:
SUB AH,AH или SUB DX,DX.

2. Арифметические операции. Вычитание

CMP r1 , r2	3	0 0 1 1 1 0 d w	mod reg r/m	Рг—Рг
CMP r , mem	9+*E*			Рг—П
CMP mem , r	16+*E*			П—Рг
CMP r , data	4	1 0 0 0 0 0 s w data L	mod 1 1 1 r/m data H(w=1)	Рг—Д
CMP mem , data	17+*E*			П—Д
CMP a , data	4	0 0 1 1 1 1 0 w data H(w=1)	data L	А—Д

Группы команд	Команды	Флаги состояния					
		OF	CF	AF	SF	ZF	PF
Сложение и вычитание	ADD ADC SUB SBC	+	+	+	+	+	+
	CMP NEG CMPS SCAS	+	+	+	+	+	+
	INC DEC	+	—	+	+	+	+

2. Арифметические операции. Умножение

MUL (MULtiply) - беззнаковое умножение

Назначение: операция умножения двух целых чисел без учета знака.

Синтаксис: MUL сомножитель_1

Алгоритм работы: Команда выполняет умножение двух целочисленных сомножителей. Один из сомножителей указан в качестве операнда. Расположение второго сомножителя указано НЕЯВНО и зависит от размера первого сомножителя.

- если операнд, указанный в команде — байт, то второй сомножитель должен располагаться в **AL**;
- если операнд, указанный в команде — слово, то второй сомножитель должен располагаться в **AX;**
- если операнд, указанный в команде — двойное слово, то второй сомножитель должен располагаться в **EAX**.

2. Арифметические операции. Умножение

От размера первого сомножителя также зависит размер результата. Результат умножения помещается также в фиксированное место, определяемое размером сомножителей:

- при умножении байтов результат помещается в AX;
- при умножении слов результат помещается в пару DX:AX;
- при умножении двойных слов результат помещается в пару EDX:EAX.

Логика работы: <Произведение> = <Сомножитель_1> * <Сомножитель_2>

Длина Произведения всегда в ДВА раза больше, чем у Множителя. Причем старшая часть Произведения находится либо в регистре AH, либо в DX.

Эти команды тоже вырабатывают флаги. Устанавливаются флаги CF=1 и OF=1, если результат слишком велик для отведенных ему регистров назначения.

2. Арифметические операции. Умножение

IMUL (Integer MULtiply) - умножение с учетом знака

Назначение: операция умножения двух целочисленных двоичных значений со знаком.
В отличии от MUL, команда может иметь 3 различных вида.
Синтаксис: IMUL сомножитель_1
IMUL сомножитель_1, сомножитель_2
IMUL результат, сомножитель_1, сомножитель_2

Алгоритм работы: Алгоритм работы команды зависит от используемой формы команды. Форма команды с одним операндом требует явного указания местоположения только одного сомножителя, который может быть расположен в ячейке памяти или регистре. Местоположение второго сомножителя фиксировано и зависит от размера первого сомножителя:

- если операнд, указанный в команде, — байт, то второй сомножитель располагается в AL;
- если операнд, указанный в команде, — слово, то второй сомножитель располагается в AX;
- если операнд, указанный в команде, — двойное слово, то второй сомножитель располагается в EAX.

2. Арифметические операции. Умножение

Результат умножения для команды с одним операндом также помещается в строго определенное место, определяемое размером сомножителей:

при умножении байтов результат помещается в AX;
при умножении слов результат помещается в пару DX:AX;
при умножении двойных слов результат помещается в пару EDX:EAX.

Команды с двумя и тремя операндами однозначно определяют расположение результата и сомножителей следующим образом:

- в команде с двумя операндами первый операнд определяет местоположение первого сомножителя. На его место впоследствии будет записан результат. Второй операнд определяет местоположение второго сомножителя;
- в команде с тремя операндами первый операнд определяет местоположение результата, второй операнд — местоположение первого сомножителя, третий операнд может быть непосредственно заданным значением размером в байт, слово или двойное слово.

Логика работы: <Произведение> = <Сомножитель_1> * <Сомножитель_2>

2. Арифметические операции. Умножение

Длина Произведения всегда в ДВА раза больше, чем у Множителя. Причем старшая часть Произведения находится либо в регистре **AH**, либо в **DX**.

Эти команды тоже вырабатывают флаги. Устанавливаются флаги **CF=1** и **OF=1**, если результат слишком велик для отведенных ему регистров назначения.

MUL src	71+E 124+E	1 1 1 1 0 1 1 w	mod 1 0 0 r/m	AX←AL источник (при w=0) DX, AX←AX источник (при w=1)
IMUL src	90+E 144+E	1 1 1 1 0 1 1 w	mod 1 0 0 r/m	AX←AL источник (при w=0) DX, AX←AX источник (при w=1) со знаком

5. Арифметические операции. Умножение

a = b * c

ax == al * bl

```
section .data
   b db -100
   c db  100
   a dw 0
section .text
global CMAIN
CMAIN:
    xor eax, eax
    xor ebx, ebx
    mov  bl, [c]
    mov  al, [b]
    imul bl                 ; ax = al * bl
    mov [a], ax             ;
    ; mov [a], al
    ; mov [a + 1], ah
ret
```

5. Арифметические операции. Умножение

a = b * c

dx:ax == ax * bx

```
section .data
   b dw -100
   c dw  100
   a dd 0
section .text
global CMAIN
CMAIN:
    xor eax, eax
    xor ebx, ebx
    mov  bx, [c]
    mov  ax, [b]
    imul bx                 ; dx:ax = ax * bx
    mov [a], ax
    mov [a + 2], dx
ret
```

5. Арифметические операции. Умножение

a = b * c

cx == cx * bx

```
section .data
  b dw 1000
   c dw 1000
   a dw 0
section .text
global CMAIN
CMAIN:
   xor ecx, ecx
   xor ebx, ebx
   mov bx, [c]
   mov cx, [b]
   imul cx, bx ;          cx = cx * bx, старшая часть игнорируется
   mov [a], cx
ret
```

5. Арифметические операции. Умножение

a = b * const

dx == cx * X

```
section .data
  b dw 1000
  a dw 0
section .text
global CMAIN
CMAIN:
   xor ecx, ecx
   xor edx, edx
   mov cx, [b]
   imul dx, cx, 4;          dx = cx * 4, старшая часть игнорируется
   mov [a], dx
ret
```

Умножение больших чисел

Чтобы умножить большие числа, придется вспомнить правила умножения десятичных чисел в столбик: множимое умножают на каждую цифру множителя, сдвигают влево на соответствующее число разрядов и затем складывают полученные результаты. В нашем случае роль цифр будут играть байты, слова или двойные слова, а сложение должно выполняться по правилам сложения чисел повышенной точности. Алгоритм умножения оказывается заметно сложнее, поэтому умножим для примера только 64-битные числа

1. Арифметические операции. Умножение двух больших чисел

```
1.   MOV EAX, DWORD [X]
2.   MOV EBX,EAX          ; сохраняем для шага 6
3.   MUL DWORD [Y]        ; перемножить младшие двойные слова
4.   MOV DWORD [Z], EAX   ; сохранить младшее слово произведения
5.   MOV ECX,EDX          ; сохранить старшее двойное слово
6.   MOV EAX,EBX          ; младшее слово "X" в eax
7.   MUL DWORD [Y+4]       ; умножить младшее слово на старшее
8.   ADD EAX, ECX
9.   ADC EDX,0            ; добавить перенос
10.  MOV EBX,EAX          ; сохранить частичное произведение
11.  MOV ECX,EDX
12.  MOV EAX,DWORD [X+4]
13.  MUL DWORD [Y]        ; умножить старшее слово на младшее
14.  ADD EAX,EBX          ; сложить с частичным произведением
15.  MOV DWORD [Z+4],EAX
16.  ADC ECX,EDX
17.  MOV EAX,DWORD [X+4]
18.  MUL DWORD [Y+4]      ; умножить старшие слова
19.  ADD EAX,ECX          ; сложить с частичным произведением
20.  ADC EDX,0            ; и добавить перенос
21.  MOV DWORD [Z+8],EAX
22.  MOV DWORD [Z+12],EDX
23.  ret
```

```
section .data
   X DQ 0x0000000100000002
   Y DQ 0x0000000300000004
   Z DQ 0, 0
```

```
   1 2
   3 4
   ---
   0 8
 0 6
 0 6     (AX+ CX)
 0 6     (+ CF)
 0 4
 0 A     (AX + BX)
 0 A     (+ CF)
0 3
0 3      (AX + CX)
0 3      ( + CF)
---
0 3 A 8
```

```
[L:1] AX = 2
[L:2] BX = 2
[L:3-5] CX = DX = 0, AX = 8 (2*4)
[L:7]DX = 0, AX = 6 (2*3)
[L:8](0 6 + 0 0)
[L:9]CX = DX = 0, BX = AX  = 6
[L:13] DX = 0, AX = 4 (1 * 4)
[L:14-15]
[L:16] CX = 0
[L:18] DX = 1, AX = 3 (1*3)
[L:19]
[L:20]
[L:21-22]
```

0 – результат умножения, 6 – результат сложения

Жирный шрифт (**8, A, 0, 3**) – сохранение результата

[L:20] – номер строки кода

2. Арифметические операции. Деление

Команды деления **DIV и IDIV**

Назначение:
DIV выполнение операции деления двух двоичных беззнаковых значений.
IDIV операция деления двух двоичных значений со знаком.
Синтаксис: DIV делитель
IDIV делитель

Алгоритм работы: Для команды необходимо задание двух операндов — делимого и делителя. Делимое задается неявно и размер его зависит от размера делителя, который указывается в команде:

- если делитель размером в байт, то делимое должно быть расположено в регистре ax. После операции частное помещается в AL, а остаток — в AH;
- если делитель размером в слово, то делимое должно быть расположено в паре регистров DX:AX, причем младшая часть делимого находится в AX. После операции частное помещается в ax, а остаток — в DX;
- если делитель размером в двойное слово, то делимое должно быть расположено в паре регистров EDX:EAX, причем младшая часть делимого находится в EAX. После операции частное помещается в EAX, а остаток — в EDX.

Логика работы: < Частное: Остаток> = <Делимоё> / <Делитель>

2. Арифметические операции. Деление

Команда **IDIV** реагирует на ЗНАК обрабатываемых чисел.

Результат состоит из Частного и Остатка, которым при обычных целочисленных вычислениях пренебрегают.

Длина Делимого всегда в ДВА раза больше, чем у Делителя. Причем старшая часть Делимого находится либо в регистре AH, либо в DX.

Эти команды тоже вырабатывают флаги. Но устанавливаются флаги **CF=1** и **OF=1**, если частное НЕ помещается в регистры AL или AX.

Здесь, в отличие от команд умножения, может генерироваться ПРЕРЫВАНИЕ "Деление на ноль”.

Применение: Команда выполняет целочисленное деление операндов с выдачей результата деления в виде частного и остатка от деления. При выполнении операции деления возможно возникновение исключительной ситуации: 0 — ошибка деления. Эта ситуация возникает в одном из двух случаев: делитель равен 0 или частное слишком велико для его размещения в регистре **eax/ax/al.**

1. Арифметические операции. Деление

a = b / c

dx:ax / bx => ax*dx/bx, dx - остаток

```
section .data
  b dw 100
  c dw  48
  a dw 0
section .text
global CMAIN
CMAIN:
   mov  bx, [c]
   mov  ax, [b]
   cwd
   div bx                ; dx:ax = ax / bx
                         ; ax = 2, dx = 4
   mov [a], ax           ; a = 2
ret
```

0. Арифметические операции. Деление

```
section .data

divident dd 0xaabbccdd

divisor dw 0x1122

quotent dd 0

modulo dw 0
```

```
xor rax, rax
MOV AX, [divident + 2] ; ax = 0xaabb
XOR DX, DX             ;  dx = 0 – старшая часть

DIV word [divisor]      ;  ax – частное, dx – остаток
                        ; в следующем делении – старшая часть
MOV [quotent + 2], AX ; сохраняем ст.часть частного
MOV AX, [divident]     ; ax = 0xccdd
DIV word [divisor]      ;  ax – частное, dx – остаток
MOV [quotent], AX      ; сохраняем мл.часть частного
MOV [modulo], DX       ; сохраняем остаток
```

0. Арифметические операции. Деление

```
section .data

divident  dq 0x1122334455667788
divisor    dw 0xaabb
quotent  dq 0
modulo  dw 0
```

```
xor rax, rax
MOV AX, [divident + 6]
XOR DX, DX
DIV word [divisor]
MOV [quotent + 6], AX
MOV AX, [divident + 4]
DIV word [divisor]
MOV [quotent + 4], AX
MOV AX,[divident + 2]
DIV word [divisor]
MOV [quotent + 2], AX
MOV AX, [divident]
DIV word [divisor]
MOV [quotent], AX
MOV [modulo], DX
```

Спасибо