АРХИТЕКТУРА микропроцессоров и программирование на языке ассемблера

Рекомендуемая литература

Основные учебники:

- Юров В. И. Assembler: учебник для вузов. СПб: Питер, 2000г. – 624 с.
- Пильщиков В.Н. Программирование на языке Ассемблера IBM PC. - М.: ДИАЛОГ-МИФИ, 1996.- 288с.: ил.

Методические указания:

- Программирование на языке ассемблера: методические указания к лабораторному практикуму. В 2 ч. / Сост. - Бейлекчи Д.В., Калинкина Н.Е. – Муром: Изд. - ИПЦ МИ ВлГУ, 2007. - Ч1. 60 с.: , Ч2. 63 с .

Электронные книги: 1,2,3+

- Голубь Н.Г. Искусство программирования на ассемблере: лекции и упражнения. – СПб.: ДиаСофтЮП, 2002.– 656с.
- Ирвин К. Язык ассемблера для процессоров Intel. – М.: Вильямс, 2005. – 912 с.
- Зубков С.В. Assembler для DOS, Windows и Unix. М.: ДМК Пресс, 2000. – 608 с.

Справочник

- Браун Р. Кайл Д. Справочник по прерываниям для IBM PC. в 2х томах. М.: Мир, 1994. Т1. 1994. - 558с. Т2. 1994. - 480с., 1 экз.

Архитектура ЭВМ

Понятие архитектуры ЭВМ включает в себя:

– структурную схему ЭВМ;

– средства и способы доступа к элементам структурной схемы;

– организацию и разрядность интерфейсов ЭВМ;

– набор и доступность регистров;

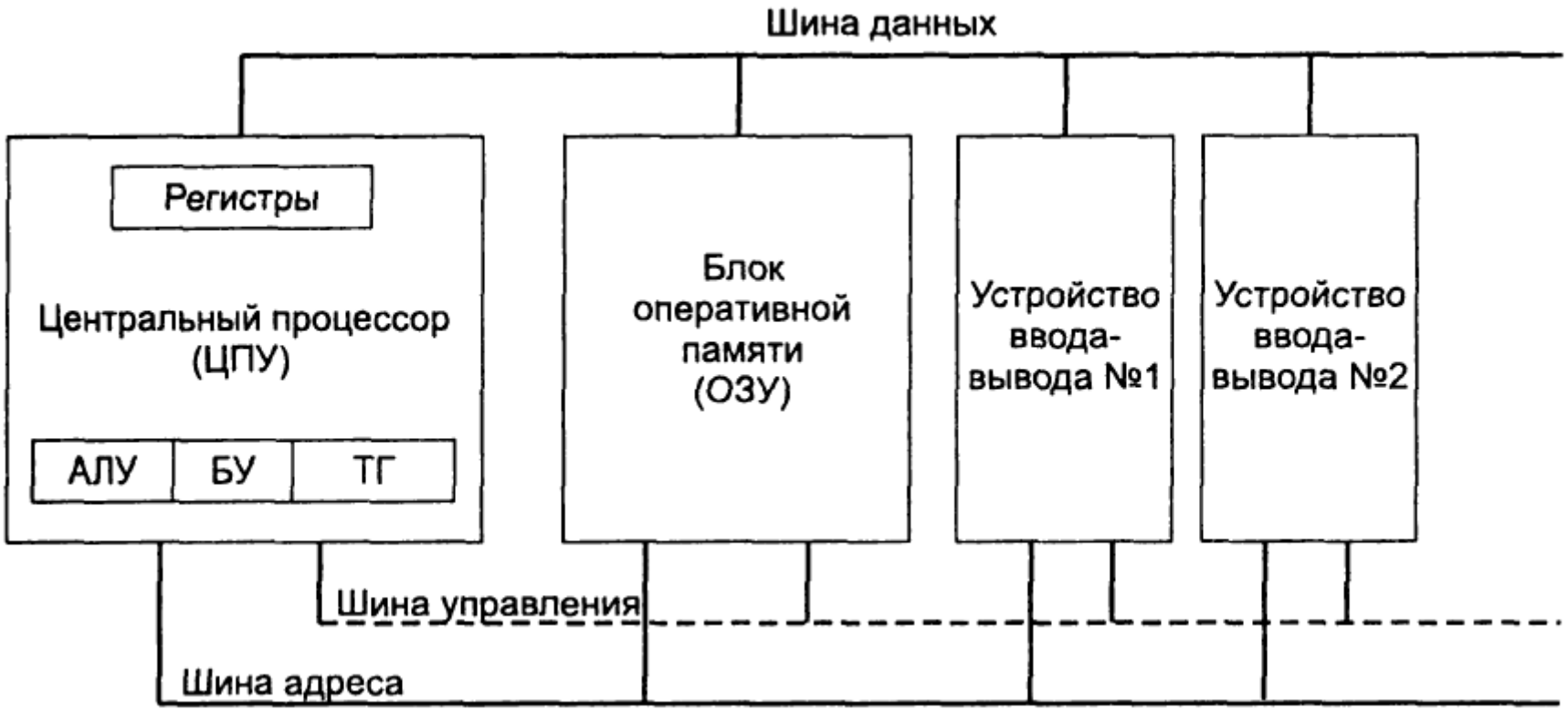
– организацию и способы адресации памяти;

– способы представления и форматы данных ЭВМ;

– набор машинных команд ЭВМ;

– обработку нештатных ситуаций (прерываний).

Структурная схема типичной МПС



Все вычисления и логические операции выполняются блоком центрального процессора (ЦПУ). В нём предусмотрены: небольшое число внутренних ячеек памяти, называемых *регистрами*, высокочастотный *тактовый генератор* (ТГ), *блок управления* (БУ) и *арифметико-логическое устройство* (АЛУ).

Подробнее: Ирвин К. Язык ассемблера для процессоров Intel.

Программная модель микропроцессора (МП)

Регистры общего назначения (16-ти разрядные):

Регистры данных:

AX

AH	AL

BX

BH	BL

CX

CH	CL

DX

DH	DL

AX – accumulator – аккумулятор

BX – base – регистр базы

CX – counter – регистр счетчик

DX – data – регистр данных

Регистры указателей и индексов:

SP – Stack Pointer – указатель стека

IP – Instruction Pointer – указатель команд

BP – Base Pointer – указатель базы

SI – Source Index – индекс источника

DI – Destination Index – индекс приёмника

Сегментные регистры:

CS – Code Segment – сегментный регистр кода

SS – Stack Segment – сегментный регистр стека

DS – Data Segment – сегментный регистр данных

ES – Extended Segment – сегментный регистр данных

Регистр флагов

Регистр флагов.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	×	×	×	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

OF – флаг переполнения (OVERFLOW) = 1, если возникает арифметическое переполнение;

SF – флаг знака (SIZE) = 1, когда старший бит результата равен 1;

ZF – флаг нуля (ZERO) = 1, если результат равен нулю;

AF – флаг вспомогательного переноса (AUXILIARY CARRY) перенос или между полубайтами;

PF – флаг чётности (PARITY) = 1, если младшие 8 бит результата содержат чётное число бит =1;

CF – флаг переноса (CARRY) = 1, если имеет место перенос или заём из старшего бита результата.

DF – флаг направления (DIRECTION) устанавливается в 1 для автоматического декремента и в 0 для автоинкрементна индексных регистров после выполнения операции над строками;

IF – флаг прерывания (INTERRUPT). Если IF = 1, то прерывания разрешены, иначе распознаются лишь немаскируемые прерывания;

TF – флаг трассировки (TRACE).

Представление данных в памяти ЭВМ

		5A	6B	22	7F		
A-2	A-1	A	A+1	A+2	A+3	A+4	

A-2	
A-1	
A	5A
A+1	6B
A+2	22
A+3	7F
A+4	

1 байт = $5A_{16}$

2 байта = $6B5A_{16}$

4 байта = $7F226B5A_{16}$

Сегментная адресация памяти

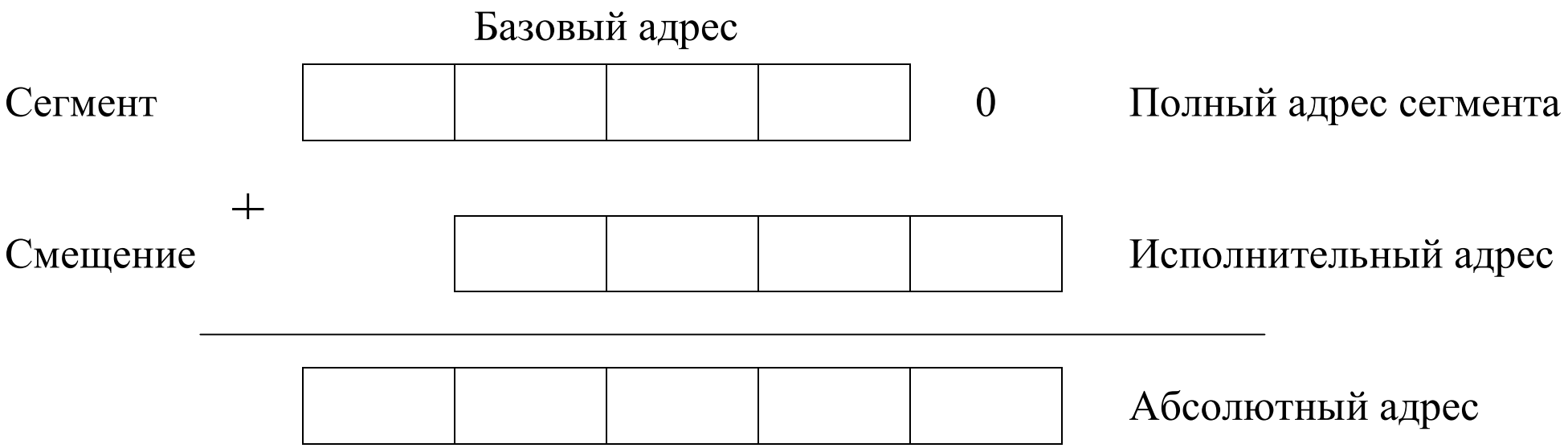


Рис.1 Формирование физического адреса

Пример: вычислим физический адрес, соответствующий логическому 38DE: 4A3F.

```
  38DE0
+  4A3F
-------
  3D81F
```

Режимы адресации

Неявная	XLAT
Регистровая	MOV AX,BX
Непосредственная	MOV AX, 100

Косвенная адресация	Формат	Сегмент по умолчанию	Примеры
Прямая	метка, смещение (ofs)	DS DS	MOV [LAB], AX MOV [1234h], AX
Индексная	[DI+ofs] [SI+ofs]	DS DS	MOV [DI+12h], AX MOV [SI+LAB], AX
Базовая	[BX+ofs] [BP+ofs]	DS SS	MOV [BX+1], AX MOV [BP+LAB], AX

Базово-Индексная	[BX+SI+ofs]
	[BX+DI+ofs]
	[BP+SI+ofs]
	[BP+DI+ofs]
		DS
		DS
		SS
		SS
			MOV [BX+SI-12h], AX
			MOV [BX+DI], AX
			MOV [BP+SI], AX
			MOV [BP+DI+LAB], AX

Начальные сведения о языке

Допустимыми символами при именовании объектов программы являются:
Латинские буквы: a..z, A..Z. Большие и маленькие – неразличимы.
Цифры: 0..9
Знаки: ? @ $ _
Разделители: [ ] ( ) . , < > { } = / + - * % ! " " ' ' \ # ^

Идентификаторы – последовательность латинских букв, цифр и знаков ? @ $ _, начинающаяся с буквы, @ ? . _

Идентифика торы делятся на служебные слова (регистры, мнемокоды) и имена. Нельзя использовать служебные слова в качестве имён.

Комментарии – компилятор игнорирует весь текст за ; до конца строки, поэтому в комментарии можно использовать любые символы.

Многострочный комментарий:

```
Comment * это уже комментарий
это всё ещё комментарий
а это * последняя строка комментария
```

Константы

Целочисленные константы:

Состоят из необязательного знака, одной или более цифр и необязательного суффикса – указателя системы счисления.

```
162, -24, -48d, +33D          ; десятичные числа
101b, -11100B                 ; двоичные числа
162q, -74Q, -44o, 13O         ; восьмеричные числа
0A2h, 65H                     ; шестнадцатеричные числа
```

Символьные константы:

Один символ, заключенный в одинарные или двойные кавычки или его код (целое от 0 до 255). Символ автоматически будет заменен на соответствующий ASCII-код: 2Ah, '*', "*".

Строковые константы:

Последовательность символов, заключенных в одинарные или двойные кавычки.
"Привет!" или 'Привет!'

Выражения (можно пропустить)

Выражения вычисляются компилятором на момент подготовки исполняемого кода.

Результат вычисления выражения – 16-ти разрядная **константа**. Если результат не константа, то будет выдана ошибка компиляции.

```
Zn EQU 175h
```

Арифметические операции:	+, -, *, / (частное от деления нацело), MOD (остаток), SHL, SHR	mov ax, 23*6 mov ax, 65 SHR 2
Логические:	AND, OR, XOR, NOT.	mov ax, 3 AND 5
Отношения:	EQ, NE, LT, GT, LE, GE.	mov ax, Zn LT 3

Возвращающие значения:

$	счётчик текущей ячейки	jmp $+7
SEG	сегментную часть адреса	mov ax, SEG Val
OFFSET	смещение адреса	mov ax, Offset Val
LENGTH	длину (в байтах или словах) переменной, объявленной с использованием DUP	
TYPE	длину элемента переменной	
SIZE	= LENGTH * TYPE	
HIGH	старший байт 16-битового выражения	mov al, HIGH Zn
LOW	младший байт 16-битового выражения	mov al, LOW Zn

Присваивания атрибута

PTR	изменяет атрибут типа (BYTE, WORD…) или дистанции (NEAR, FAR) на новое значение	Val dw 1234h mov al, byte ptr [Val]
SHORT	изменяет атрибут NEAR на SHORT.	jmp short metka

Директивы определения данных

Директивы	DB,	DW,	DD,	DP,	DF,	DQ,	DT	служат для выделения
	1	2	4	4	6	8	10	байт соответственно.

Примеры:

```
T1    DB    16                    ; выделить 1 байт и заполнить его значением 16
T2    DB    ?                     ; 1 байт, значение не определено
T3    DW    4*3                   ; 1 слово (2 байта), значение = 12
S1    DB    'abCDf'               ; 5 байт: 'a',’b ’,’C ’,’D ’,’f ’
S2    DD    'ab'                  ; 4 байта памяти и заполнить  00006162h
T4    DB    7 DUP(?)              ; 7 байт, значение не определено
T5    DW    5 DUP(5 DUP(10))      ; 5*5 слов памяти, каждое = 10
T6    DB    3 DUP(1,2,3,4,5)      ; 15 байт памяти = 1,2,3,4,5,1,2,3,4,5…
T7    DQ    18446744073709516     ; 8 байт и заполнить
T8    DT    12089258196146291747  ; 10 байт и заполнить
```

Система команд микропроцессора 8086.

Используемые обозначения:

dst	– операнд-приемник
src	– операнд-источник
r, r8, r16	– операнд регистре 8 или 16-ти разрядном.
m, m8, m16, m32	– операнд в памяти размером байт, слово или двойное слово
sreg	– операнд в сегментном регистре
imm	– непосредственный операнд

Команды передач данных

MOV dst, src ; dst ⇐ src

MOV r,im	MOV r/m,r	MOV sreg,r/m
MOV m,im	MOV r,r/m	MOV r/m,sreg

Примеры использования команды (укажите режимы адресации?):

```
MOV  DI,20          ; записать в регистр DI константу 20₁₀
MOV  AX, BX         ; записать содержимое регистра BX в AX
MOV  AX, [BX]       ; записать слово с адреса DS:BX в AX
MOV  DI,  Index     ; скопировать содержимое двухбайтной
MOV  DI, [Index]    ; переменной Index  в регистр DI
MOV  Star, DS
MOV  Days,365
```

Обмен регистра/памяти с регистром

XCHG dst1, dst2 ; dst1 ⇔ dst2

XCHG r,r/m XCHG r/m,r

Примеры:

```
XCHG   BL,BH      ; обмен содержимого регистров BL и BH
XCHG   DH, Char   ; обмен содержимого регистра DH с ячейкой памяти Char
```

Команды работы со стеком

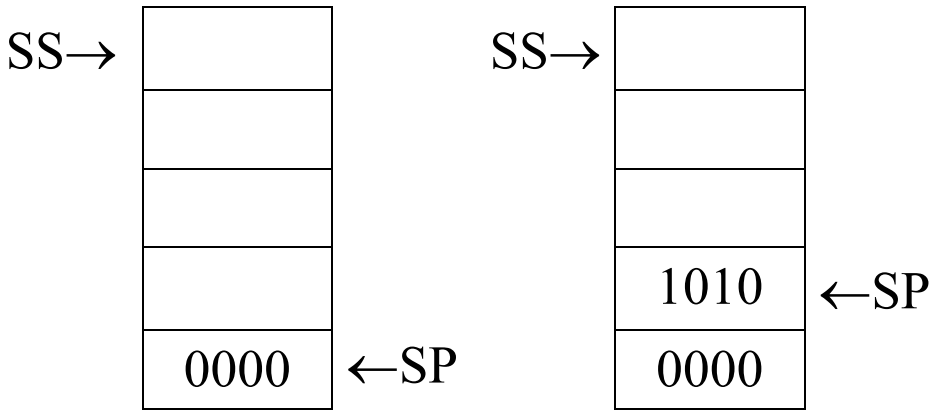
POP dst ; Извлечь слово из стека (dst ⇐ SS:[SP]; SP+=2)

PUSH src ; Записать слово в стек (SP-=2; SS:[SP] ⇐ src)

КОП r/m/sreg PUSH imm

Примеры: (Пусть AX=1010h)

```
PUSH   AX
POP    BX
```



POPF ; Извлечь флаги из стека

PUSHF ; Записать флаги в стек

Загрузка эффективного адреса

LEA – Load Effective Address.

LEA r,m ; r ⇐ рез-тат вычисления исполнительного адреса

Пример:
```
LEA BX, TABLE+4          ; записать в BX смещение TABLE+4
MOV  BX, OFFSET Table+4  ; то же, но другим способом
```

Загрузка полного указателя

LDS r, m ; загрузить полный адрес ячейки памяти m в DS:Reg

LES r, m ; загрузить полный адрес ячейки памяти m в ES:Reg

Пример:
```
LES  DI, Table    ; загрузка в ES:DI полного адреса Table
LDS  BX,[BP+4]    ; загрузка в DS:BX  указателя с адреса SS:BP+4
```

Табличное преобразование

Команда **XLAT** (transLATe – преобразовать) заменяет содержимое регистра AL байтом с номером AL из таблицы перекодировки. Адрес таблицы перекодировки в DS:BX.

XLAT **; AL ⇐ DS:[BX+ беззнаковый AL]**

Пример:
```
MOV     BX,OFFSET Talbe
MOV     AL,2
XLAT                          ; AL ⇐ 'c'   (счёт с нуля!)
…
Table DB    'abcde'
```

Команды ввода-вывода

Команды IN ввода и OUT вывода предназначены для обращения к портам. Команда IN применяется для получения, а команда OUT для выдачи одного байта или слова в порт.

IN AL/AX, n ; Ввод байта/слова из порта

OUT n,AL/AX ; Вывод байта/слова в порт.

IN al/ax,imm8/DX OUT imm8/DX, al/ax

Примеры:

```
IN   AL,61h     ; Читаем байт из порта 61h
OUT  DX,AL      ; Пишем байт из AL в порт
```

Команды передач данных

1.	**MOV**	dst, src
2.	**XCHG**	dst1, dst2
3.	**XLAT**	
4.	**POP**	dst
5.	**PUSH**	src
6.	**POPF**	
7.	**PUSHF**	
8.	**LAHF**	
9.	**SAHF**	
10.	**LEA**	**r,m**
11.	**LDS**	**r, m**
12.	**LES**	**r, m**
13.	**IN A**	**L/AX, n**
14.	**OUT**	**n,AL/AX**

Арифметические команды

Все арифметические команды влияют на флаги **OF, SF, ZF, AF, PF и CF** в зависимости от результата операции (устанавливают или сбрасывают).

Команды сложения и вычитания

Мнемоника	oszapc	Описание
INC dst	*****–	dst ⇐ dst +1
DEC dst	*****–	dst ⇐ dst - 1
NEG dst	******	dst ⇐ 0 - dst (изменить знак операнда)

КОП r/m

Мнемоника	oszapc	Описание
ADD dst,src	*–****	dst ⇐ dst + src
ADC dst,src	*–****	dst ⇐ dst + src + CF add with carry
SUB dst,src	*–****	dst ⇐ dst – src
SBB dst,src	*–****	dst ⇐ dst - src - CF subtract with borrow
CMP dst,src	*–****	флаги ⇐ dst - src

КОП r/m,im КОП r/m,r КОП r,r/m

Команды сложения и вычитания

Примеры:

```
    .   .                .   .
+   2   7   6        –   4   2   7   5
    4   9   1   6            9   1   6
-------------------  -------------------
    5   1   9   2        3   3   5   9
```

```
INC  BX                 DEC  DI                 NEG  AX
INC  byte ptr [BX]      DEC  word ptr [DI+5]    NEG  [Count]
ADD  AL,12h             SUB  AL,12h             CMP  AL,12h
ADD  Count,1            SUB  Count,1            CMP  Count,1
ADC  BX,14              SBB  BX,14              CMP  BX,14
ADD  AX,BX              SUB  AX,BX              CMP  AX,BX
ADC  Count, DI          SBB  Count, DI          CMP  Count, DI
```

Команды умножения MUL и IMUL и деления DIV и IDIV.

Мнемоника	oszapc	Описание
DIV src8 IDIV src8	??????	AL ⇐ (AX div src8); AH ⇐ (AX mod src8)
DIV src16 IDIV src16	??????	AX ⇐ (DX:AX div src16); DX ⇐ (DX:AX mod src16)
MUL src8 IMUL src8	*????*	AX ⇐ (AL *src8)
MUL src16	*????*	DX:AX ⇐ (AX * src16)

Примеры:

```
MUL   BX        ; DX:AX  ←  AX·BX
MUL   DL        ; AX     ←  AL·DL
DIV   CX        ; AX     ←  DX:AX div CX
                ; DX     ←  DX:AX mod CX
DIV   BL        ; AL     ←  AX div BL
                ; AH     ←  AX mod BL
```

Логические команды

Мнемоника	oszapc	Описание
AND dst,src	0**?*0	принудительно обнуляет биты dst, заданные нулем в src, не меняя остальные
TEST dst,src	0**?*0	
ORdst,src	0**?*0	принудительно присваивает 1 битам dst, установленным в 1 в src, не меняя остальные.
XOR dst,src	0**?*0	выясняет, в каких битах src и dst отличаются. Бит результата равен 1, если соответствующие биты обоих операндов различны, иначе бит результата = 0.

КОП r/m,im КОП r,r/m КОП m,r

NOT dst	------	dst := не dst; инвертировать все биты dst

NOT r/m

Примеры:

```
AND  AL,11110000b      ; обнулить 4 младших бита AL
TEST AL,11110000b
OR   AL, 10001000b     ; Установить 3 и 7 биты AL
XOR  AL,11110000b
XOR  AL,AL             ; Обнулить AL
```

Команды сдвига

SHL / SHR	shift left / right	Логический сдвиг влево / вправо
SAL / SAR	shift arithmetic	Арифметический сдвиг влево / вправо
ROL / ROR	rotate left / right	Циклический сдвиг влево / вправо
RCL / RCR	rotate through carry	Циклический сдвиг через перенос

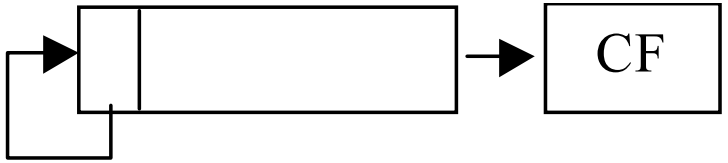
Действие команд сдвига иллюстрируют следующие рисунки.

SHL

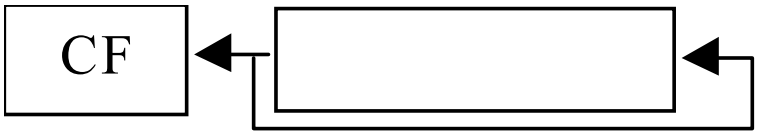
SHR

SAL

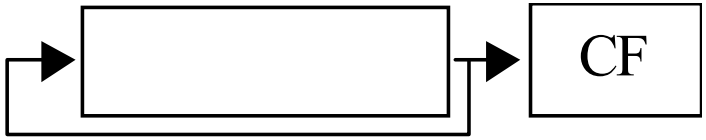
SAR



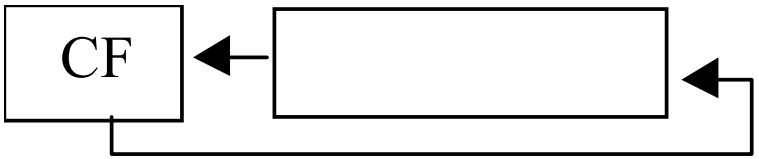
ROL



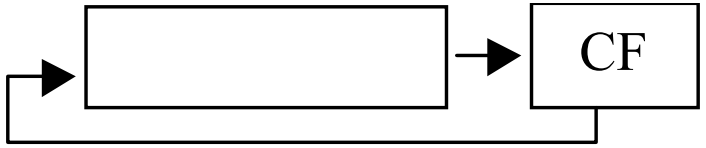
ROR



RCL



RCR



Команды сдвига

КОП dst,1 ; сдвиг dst на 1 разряд

КОП dst,CL ; сдвиг dst на CL разрядов (CL≥0).

КОП r/m,1 КОП r/m,CL

Примеры:

```
Пусть до выполнения команд     CH = 10111001
SHL  CH,1          тогда после   CH =  01110010
ROL  CH,CL         Если CL=3,то  CH =    11001101
```

Арифметические команды

INC	dst	DEC	dst	MUL	src
ADD	dst,src	NEG	dst	IMUL	src
ADC	dst,src	SUB	dst,src	DIV	src
		SBB	dst,src	IDIV	src
		CMP	dst,src		

Логические команды

AND	dst,src
TEST	dst,src
OR	dst,src
XOR	dst,src
NOT	dst

Команды сдвига

SHL	dst,src	SHR	dst,src
SAL	dst,src	SAR	dst,src
ROL	dst,src	ROL	dst,src
RCL	dst,src	RCL	dst,src

Команды безусловной передачи управления

Безусловный переход

```
        jmp L                       jmp near ptr metka    ; прямой near переход
        …                           jmp word ptr [BX]     ; косвенный near переход
L:      mov ax,7                    jmp dword ptr [BX]    ; косвенный far  переход
```

Переход на подпрограмму и возврат из неё.

```
CALL   [дистанция]  имя_подпрограммы      call my_proc
RET [n]                                   mov ax,20
```

Вспомним, что в CS:IP находится адрес команды, следующей за исполняемой.

Если выполняется команда CALL, то в CS:IP адрес той команды, которую надо будет выполнить после завершения подпрограммы.

Команды безусловной передачи управления

Действия команды CALL

1. сохранить адрес возврата (2 или 4 байта) на вершине стека (содержимое IP или CS:IP).
2. передать управление на указанный в команде адрес.

Действия команды RET

1. извлечь с вершины стека адрес возврата (2 или 4 байта) в IP или CS:IP.
2. далее автоматически происходит выполнение команды, следовавшей за вызвавшей подпрограмму командой CALL.

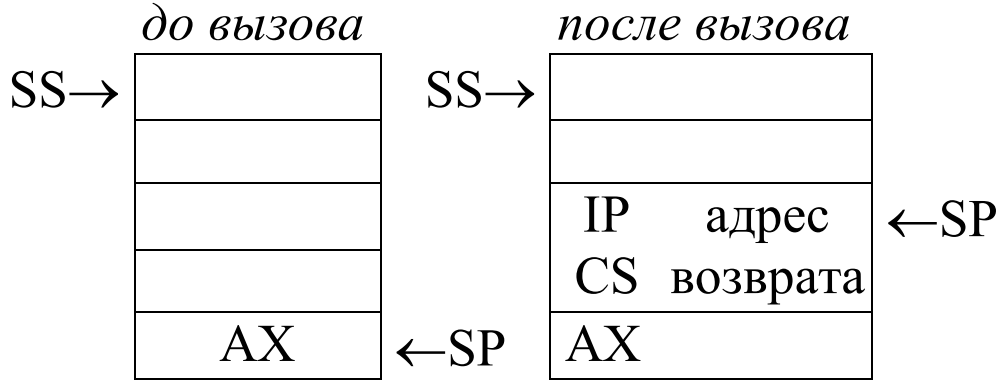
Два байта будут сохранены (извлечены) если применяется значение дистанции NEAR или приведение типа к WORD. Иначе, при использовании значений FAR (DWORD), сохраняются (извлекаются) 4 байта.

Основная программа

```
PUSH   AX
CALL   my_pr
…
CALL   my_pr
…
```

Подпрограмма

```
my_pr  PROC  FAR
       …
       RET
my_pr  ENDP
```



Команды безусловной передачи управления

Переход на прерывание и возврат из него

INT номер

IRET

Подробнее рассмотрим обработку программных прерываний.

1. Сохранить флаги в стеке.

2. Сохранить FAR адрес возврата в стеке (CS:IP).

3. Выполнить косвенный переход по адресу (0:номер*4) на обработчик прерывания.

Команда IRET извлекает из стека IP, CS, флаги и возобновляет прерванную программу.

Команды условной передачи управления

Условие перехода	Команда для чисел без знака
dst > src	JA/JNBE short_label
dst ≥ src	JAE/JNB short_label
dst < src	JB/JC short_label
dst ≤ src	JBE/JNA short_label
Условие перехода	Команда для чисел со знаком
dst > src	JG/JNGE short_label
dst ≥ src	JGE/JNL short_label
dst < src	JL/JNGE short_label
dst ≤ src	JLE/JNG short_label

Пример:

```
        Cmp ax,bx
        ja  m1
        xchg ax,bx
    m1:
```

Команды условной передачи управления

Команда		Описание
JE/JZ	label	if равно
JNE/JNZ	label	if не равно
JC	label	if Carry
JNC	label	if не Carry
JO	label	if переполнение
JNO	label	if не переполнение
JP/JPE	label	if чётная сумма битов =1
JNP/JPO	label	if нечётная сумма битов
JS	label	if знаковыйбит =1
JNS	label	if знаковый бит =0
JCXZ	label	переход если CX=0

Команды условной передачи управления
Организация циклов

LOOP	label	DEC CX, переход на метку, если CX>0
LOOPE/ LOOPZ	label	DEC CX, переход на метку, если CX>0 и ZF=1
LOOPNE/ LOOPNZ	label	DEC CX, переход если CX<>0 и ZF=0

Команды LOOPE/ LOOPZ и LOOPNE/ LOOPNZ расширяют действие команды LOOP тем, что дополнительно анализируют флаг ZF, что даёт возможность организовать досрочный выход из цикла.

Примеры:

```
MAS DW 10 DUP (?)
    …
    LEA BX,MAS
    MOV CX,LENGTH MAS
    XOR AX,AX
M1: MOV [BX],AX
    INC BX
    INC BX
    LOOP M1
```

```
MAS Db 10 DUP (?)
    …
    LEA BX,MAS-1
    MOV CX,10
    XOR AL,AL
M1: INC BX
    CMP [BX],AL
    LOOPZ M1
    JZ    no
    …
NO:
```

```
MAS Db 10 DUP (?)
    …
    LEA BX,MAS-1
    MOV CX,10
    XOR AL,AL
M1: INC BX
    CMP [BX],AL
    LOOPNZ M1
    JNZ    no
    …
NO:
```

Инструкции обработки цепочек

Цепочка (строка) – последовательность контекстно-связанных байт (слов), находящихся в соседних ячейках памяти.

По умолчанию строка-приёмник адресуется через ES:DI, а строка-источник – через DS:SI.

MOVS (MOVSB / MOVSW) – копировать строку байт (слов).

LODS (LODSB / LODSW) – загрузить (прочитать) байт (слово) из строки в AL (AX).

STOS (STOSB / STOSW) – сохранить (записать) байт (слово) из AL (AX) в строку

CMPS (CMPSB / CMPSW) – сравнить строки байт (слов).

SCAS (SCASB / SCASW) – найти байт (слово) в строке

Команды обработки цепочек автоматически модифицируют регистры SI и DI для адресации следующего элемента строки. Например, команда MOVS увеличивает или уменьшает значение индексных регистров SI и DI после каждого цикла своего исполнения на 1 (MOVSB) или на 2 (MOVSW).

Флаг направления DF определяет, будут значения регистров SI и DI увеличены или уменьшены по завершении выполнения команды манипулирования строками. Если флаг DF равен 0, то значения регистров SI и DI увеличиваются, иначе они уменьшаются. Состоянием флага DF можно управлять с помощью команды **CLD** (clear direction flag – сбросить флаг направления), которая полагает его равным нулю, и **STD** (установить флаг направления), которая присваивает ему значение 1.

Префиксы повторения.

Можно сделать так, чтобы одна команда обработки строк работала с группой последовательных элементов памяти. Для этого перед ней надо указать префикс повторения. Это не команда, а однобайтный модификатор, который приведет к аппаратному повторению команды обработки строк, что сокращает время на обработку длинных строк по сравнению с программно-организованными циклами. Префикс повторения указывается перед цепочечной командой в поле метки. Число повторений извлекается из регистра CX. На каждом шаге значение CX уменьшается на 1.

REP	повторять строковую операцию пока CX≠0
REPE/REPZ	операция повторяется пока (CX≠0 и ZF=1)
REPNE/REPNZ	операция повторяется пока (CX≠0 и ZF=0).

Команда пересылки строки MOVS (MOVSB, MOVSW).

MOVS dest, src ; Move string – переслать строку байт или слов

MOVSB ; ES:[DI]:=DS:[SI]; DI=DI±1; SI=SI±1

MOVSW ; ES:[DI]:=DS:[SI]; DI=DI±2; SI=SI±2

MOVSB – копировать байт из DS:[SI] в ES:[DI];

MOVSW – копировать слово из DS:[SI] в ES:[DI].

Каждая групповая пересылка с помощью команды MOVS осуществляется с помощью пяти шагов:

- обнулить (CLD) или установить (STD) DF;
- загрузить адрес строки-источника в DS:SI;
- загрузить адрес строки-приёмника в ES:DI;
- загрузить счётчик элементов в CX;
- выполнить команду MOVS с префиксом REP.

```
S1 db 10 dup (?)
S2 db 10 dup (?)
 …
   cld
   les si,s1
   lds di,s2
   mov cx,10
rep movsb
```

Команда загрузки строки LODS (LODSB, LODSW)

LODS src ; Load string – загрузить строку

LODSB ; AL: = DS:[SI]; SI±=1;

LODSW ; AX: = DS:[SI]; SI±=2;

Команда сохранения строки STOS (STOSB, STOSW).

STOS dest ;Store string – сохранить строку

STOSB ;ES:[DI]:=AL; DI±=1;

STOSW ;ES:[DI]:=AX; DI±=2;

Пример:

```
        push ds
        pop es
        cld
        mov cx,n
        lea si,s1
        lea di,s2
m1:     lodsb
        and al,5fh
        stosb
        loop m1
```

Команда сканирования строки SCAS (SCASB, SCASW).

SCAS dest ; Scaning string- сканировать строку (поиск в строке)

SCASB ; флаги:=(результат CMP AL, ES:[DI]); DI±=1

SCASW ; флаги:=(результат CMP AX, ES:[DI]); DI±=2

Сравнивает содержимое регистра AL (AX) с байтом (словом) по адресу ES:DI, после чего регистр DI устанавливается на следующий элемент. Устанавливает флаги, аналогично команде CMP.

Команда сравнения строк CMPS (CMPSB, CMPSW).

CMPS dest, src ; Compare sring – сравнить строку

CMPSB ; флаги:=CMP DS:[SI],ES:[DI]; DI±=1; SI±=1

CMPSW ; флаги:=CMP DS:[SI],ES:[DI]; DI±=2; SI±=2

Сравнивает значение элемента одной строки (DS:SI) со значением элемента второй строки (ES:DI) и настраивает значения регистров SI, DI на следующие элементы. Результат – сформированные флаги. Обратите внимание, что CMP вычитает операнд-источник из операнда-приемника, а CMPS операнд-приемник из операнда-источника. Это означает, что указываемые после команды CMPS команды условной передачи управления должны отличаться от тех, что в аналогичной ситуации следовали бы за командой CMP.

Команды управления состоянием процессора

К командам управления состоянием процессора относят команды работы с флагами:

CLC – (CLear Carry) сбросить флаг переноса (CF=0).

CMC – (CompliMent Carry) инвертировать значение флага переноса.

STC – (SeT Carry) установить флаг переноса (CF=1).

CLD – сбросить флаг направления DF.

STD – установить флаг направления DF в единицу.

CLI – сбросить флаг прерываний, в результате чего процессор не распознает внешние маскируемые прерывания.

STI – установить флаг разрешения прерываний IF. После этого при завершении работы следующей команды процессор может выполнять обработку внешних прерываний, если эта команда снова не сбросит флаг прерываний.

Команды преобразования типов

CBW – convert byte to word – AL $\Rightarrow$ AX

CWD – convert word to double word – AX $\Rightarrow$ DX:AX

Старший бит AL (AX) воспроизводится во всех битах AH (DX).

Структура программы (шаблон)

```
begin segment
assume cs: begin, ds:dates, ss: komod
   …            ;Подпрограммы
start:
    mov ax, dates
    mov ds, ax
     …           ; Текст программы
    mov ax, 4c00h
    int 21h
begin ends

dates segment
     …           ; Данные программы
dates ends

komod segment stack
   dw 128 dup (?)
komod ends

end start
```

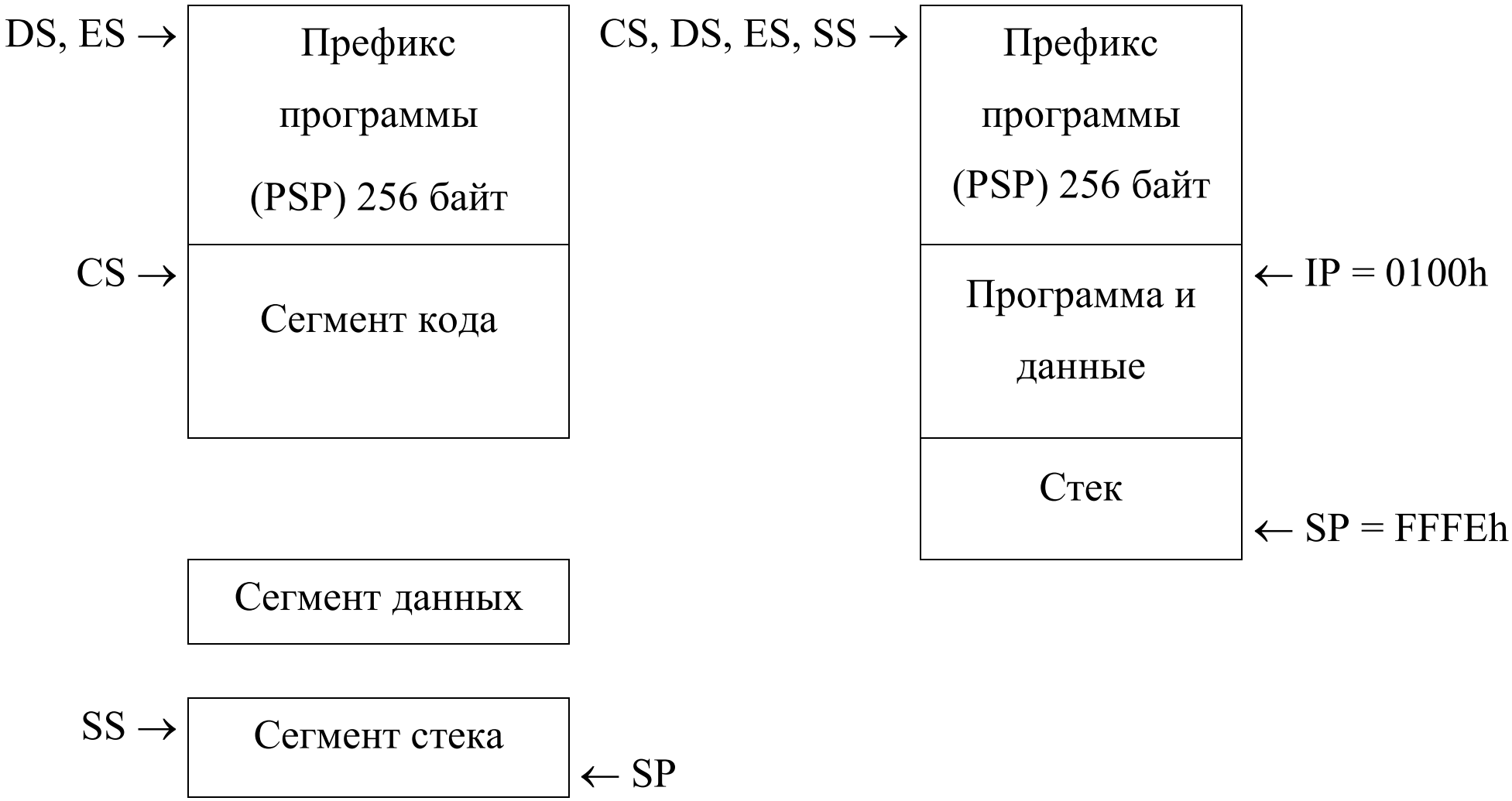
```
begin segment
org 100h
assume cs:begin, ds:begin
start:
                ; Текст программы

                …
                int 20h

                …
                    ; Подпрограммы
                …
                    ; Данные программы



Begin ends
end start
```



Компиляция и выполнение exe-программы:

```
TASM    first,,
TLINK   first,,
TD      first
```

com-программы:

```
TASM    first,,
TLINK  /t  first,,
TD      first.com
```

Директива SEGMENT

Имя **SEGMENT** выравнивание объединение класс размер

выравнивание – определяет способ размещения сегмента в памяти

BYTE — без выравнивания. Сегмент может начинаться с любого адреса памяти;

WORD — сегмент начинается по четному адресу (выравнивание на границу слова);

DWORD — сегмент начинается по адресу, кратному четырем (на границу двойного слова);

PARA — сегмент начинается по адресу, кратному 16 (на границу параграфа) = по умолчанию;

PAGE — сегмент начинается по адресу, кратному 256;

MEMPAGE — сегмент начинается по адресу, кратному 4.

объединение – определяет способ объединения сегментов снабженных одной меткой (из разных модулей) в один сегмент

PRIVATE - сегмент не будет объединяться с другими сегментами вне этого модуля (по умолчанию);

PUBLIC —объединить все сегменты с одинаковыми именами. Размер нового сегмента = сумме объединяемых;

COMMON — расположить все сегменты с одним и тем же именем с одного адреса. Размер полученного сегмента будет равен размеру самого большого сегмента из объединяемых;

STACK — Тип объединения STACK аналогичен типу PUBLIC, за исключением того, что результат объединения будет использоваться под стек.

AT xxxx — Сегмент при загрузке в память будет расположен, начиная с абсолютного адреса параграфа, но для доступа к нему в соответствующий сегментный регистр должно быть загружено заданное в атрибуте значение;

```
Begin SEGMENT para public 'Code' use16
```

Подпрограммы

имя_подпрограмы PROC [модификатор]
    тело_подпрограммы
имя_подпрограмы ENDP

параметр модификатор может принимать значения FAR или NEAR

Пример подпрограммы вычисления n!

```
public Fact                ; Дано в BX n
code   segment             ; Результат в AX
assume      cs:code        ; Портит DX
FACT    proc    Near
        CMP     BX,1
        JA      REPEAT
        MOV     AX,1
        RET
REPEAT: PUSH    BX
        DEC     BX
        CALL    FACT
        POP     BX
        MUL     BX
        RET
        endp    FACT
code   ends
       End
```

Фрагмент программы, вызывающей подпрограмму Fact

```
EXTRN       Fact : Near
Code   segment
assume      cs:code, ss:stek
start:  ...
        MOV     BX,4
        CALL    FACT   ; Вычислить 4!
        ...
Code   ends
        ...
    End    Start
```

Сборка многомодульной программы:

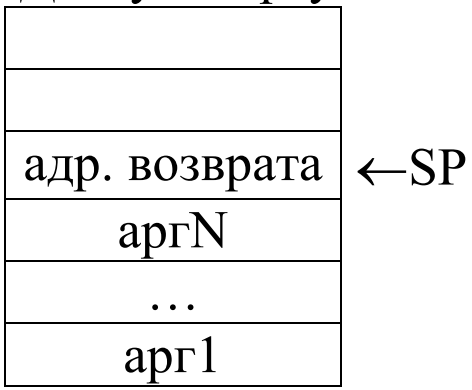
```
TASM    fact,,
TASM    main,,
TLINK   main+fact
TD      main
```

Передача аргументов подпрограмме

По ссылке (передаём адрес данных) или по значению.

В регистрах, стеке, глобальных переменных.

Доступ к аргументам в подпрограмме:



1 способ (неудобный):

– извлечь адрес возврата

– извлечь аргументы

– положить адрес возврата

```
pop bp
pop ax
pop bx
push bp
```

2 способ:

Настроить один из регистров (BP), так, чтобы он указывал на вершину стека. Рассчитать относительно BP местоположение всех аргументов.

```
push bp
mov bp,sp
mov ax,[bp+4]
mov bx,[bp+6]
…
pop bp
```

Связь подпрограмм на ассемблере с программами на ЯВУ (Pascal).

ПЕРЕДАЧА АРГУМЕНТОВ ПОДПРОГРАММЕ

Входные параметры подпрограмме на языке Pascal передаются через стек процессора или сопроцессора. Результаты работы остаются в регистрах, стеке процессора или в стеке сопроцессора. Компилятор Pascal генерирует команды подготовки аргументов в стеке при обработке вызова подпрограммы.

Параметры-значения передаются следующим образом:

- 1 байтные параметры передаются как двухбайтные, дополненные старшим байтом с неопределённым значением;
- 2, 4, 6 байтные в 1, 2, 3 словах в стеке.
- параметры типов single, double, extended, comp – на вершине стека сопроцессора.
- параметр типа множество передаётся указателем на его 32-байтное соответствие.
- все остальные параметры-значения передаются своими полными адресами.

Параметры-переменные передаются по ссылке, т.е. своими полными адресами (сегмент:смещение).

Связь подпрограмм на ассемблере с программами на ЯВУ (Pascal).

ВОЗВРАТ РЕЗУЛЬТАТОВ РАБОТЫ ИЗ ПОДПРОГРАММЫ

Результат функции может быть простого, ссылочного или строкового типа. Результаты первых двух типов возвращаются через регистры:

1 байт	– AL	2 байта	– AX	4 байта	– DX:AX	6 байт	– DX:BX:AX,

Вещественные (кроме real) – на вершине стека сопроцессора.

Под строки выделяется память, и указатель на неё находится в стеке выше (адрес старше) всех передаваемых параметров. Pascal ожидает, что при выходе из подпрограммы указатель на строку-результат из стека удалён не будет, в отличие от всех остальных аргументов подпрограммы, забота об удалении которых из стека возложена на подпрограмму.