

Машинно-зависимые языки программирования, лекция 3

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2024 г.



Команда TEST

TEST <приёмник>, <источник>

- Аналог AND, но результат не сохраняется
- Выставляются флаги SF, ZF, PF



CMOVcc - условная пересылка данных

CMOVcc <приёмник>, <источник>

Условия аналогичны Jcc



XCHG - обмен операндов между собой

XCHG <операнд1>, <операнд2>

Выполняется над двумя регистрами либо регистром и переменной



XLAT/XLATB - трансляция в соответствии с таблицей

XLAT [адрес]

XLATB

Помещает в AL байт из таблицы по адресу DS:BX со смещением относительно начала таблицы, равным AL.

Адрес, указанный в исходном коде, не обрабатывается компилятором и служит в качестве комментария.

Если в адресе явно указан сегментный регистр, он будет использоваться вместо DS.



LEA - вычисление эффективного адреса

LEA <приёмник>, <источник>

Вычисляет эффективный адрес источника и помещает его в приёмник.

Позволяет вычислить адрес, описанный сложным методом адресации.

Иногда используется для быстрых арифметических вычислений:

```
lea bx, [bx+bx*4]
```

```
lea bx, [ax+12]
```

Эти вычисления занимают меньше памяти, чем соответствующие MOV и ADD, и не изменяют флаги.

Двоичная арифметика. ADD, ADC, SUB, SBB

ADD, SUB не делают различий между знаковыми и беззнаковыми числами.

ADC <приёмник>, <источник> - сложение с переносом. Складывает приёмник, источник и флаг CF.

SBB <приёмник>, <источник> - вычитание с займом. Вычитает из приёмника источник и дополнительно - флаг CF.

```
add ax, cx  
adc dx, bx
```

```
sub ax, cx  
sbb dx, bx
```


Арифметические флаги - CF, OF, SF, ZF, AF, PF



NEG - изменение знака

NEG <приёмник>

Переводит число в дополнительный код.



Десятичная арифметика

DAA, DAS, AAA, AAS, AAM, AAD

- Неупакованное двоично-десятичное число - байт от 00h до 09h.
- Упакованное двоично-десятичное число - байт от 00h до 99h (цифры A..F не задействуются).
- При выполнении арифметических операций необходима коррекция:
 - $19h + 1 = 1Ah \Rightarrow 20h$

```
inc al
```

```
daa
```



Логический, арифметический, циклический сдвиг. **SAR, SAL, SHR, SHL, ROR, ROL, RCR, RCL**

- SAL тождественна SHL
- SHR зануляет старший бит, SAR - сохраняет (знак)
- ROR, ROL - циклический сдвиг вправо/влево
- RCR, RCL - циклический сдвиг через CF



Операции над битами и байтами

BT, BTR, BTS, BTC, BSF, BSR, SETcc


- BT <база>, <смещение> - считать в CF значение бита из битовой строки
- BTS <база>, <смещение> - установить бит в 1
- BTR <база>, <смещение> - сбросить бит в 0
- BTC <база>, <смещение> - инвертировать бит
- BSF <приёмник>, <источник> - прямой поиск бита (от младшего разряда)
- BSR <приёмник>, <источник> - обратный поиск бита (от старшего разряда)
- SETcc <приёмник> - выставляет приёмник (1 байт) в 1 или 0 в зависимости от условия, аналогично Jcc



Организация циклов

- LOOP <метка> - уменьшает CX и выполняет "короткий" переход на метку, если CX не равен нулю.
- LOOPE/LOOPZ <метка> - цикл "пока равно"/"пока ноль"
- LOOPNE/LOOPNZ <метка> - цикл "пока не равно"/"пока не ноль"

Декрементируют CX и выполняют переход, если CX не ноль и если выполняется условие (ZF).



Строковые операции: копирование, сравнение, сканирование, чтение, запись

Строка-источник - DS:SI, строка-приёмник - ES:DI.

За один раз обрабатывается один байт (слово).

- MOVS/MOVSB/MOVSW <приёмник>, <источник> - копирование
- CMPS/CMPSB/CMPSW <приёмник>, <источник> - сравнение
- SCAS/SCASB/SCASW <приёмник> - сканирование (сравнение с AL/AX)
- LODS/LODSB/LODSW <источник> - чтение (в AL/AX)
- STOS/STOSB/STOSW <приёмник> - запись (из AL/AX)

Пояснение принципа работы на примере команды movsb:

1. Байт копируется из памяти по адресу DS:SI в память по адресу ES:DI.
2. SI и DI инкрементируются (декрементируются, если установлен DF).



Строковые операции: префиксы повторения

Префиксы: REP/REPE/REPZ/REPNE/REP NZ

Пример копирования 10 байт из одного массива в другой:

```
mov cx, 5  
rep movsw
```



Управление флагами

- STC/CLC/CMC - установить/сбросить/инвертировать CF
- STD/CLD - установить/сбросить DF
- LAHF - загрузка флагов состояния в AH
- SAHF - установка флагов состояния из AH
- CLI/STI - запрет/разрешение прерываний (IF)

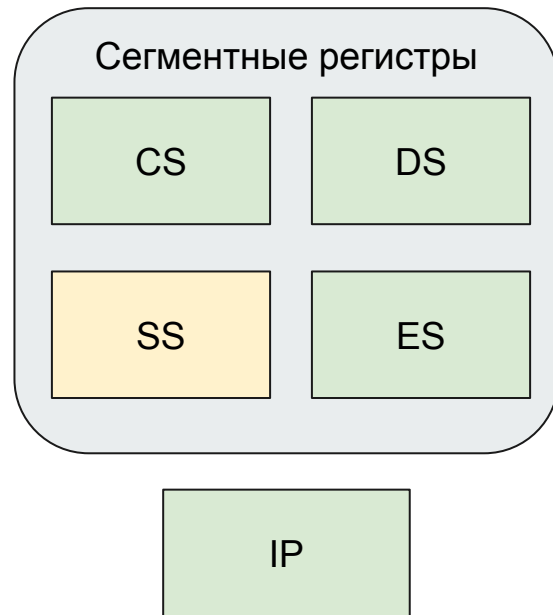
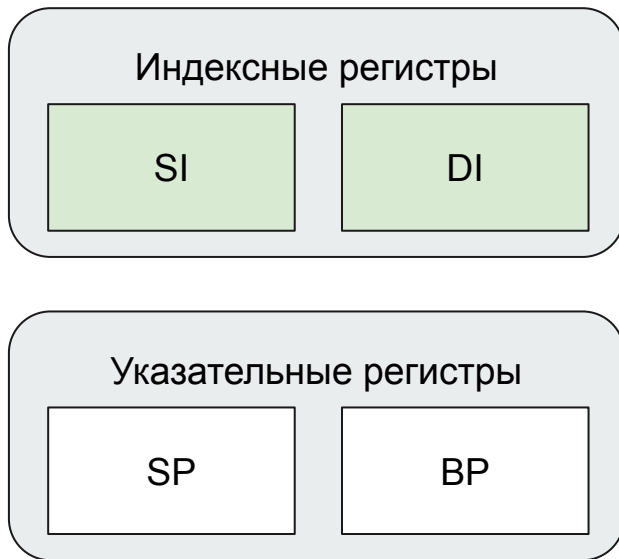
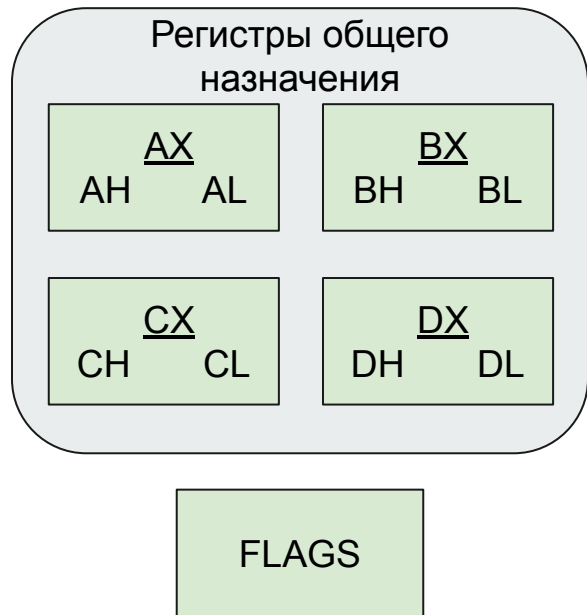


Загрузка сегментных регистров

- LDS <приёмник>, <источник> - загрузить адрес, используя DS
- LES <приёмник>, <источник> - загрузить адрес, используя ES
- LFS <приёмник>, <источник> - загрузить адрес, используя FS
- LGS <приёмник>, <источник> - загрузить адрес, используя GS
- LSS <приёмник>, <источник> - загрузить адрес, используя SS

Приёмник - регистр, источник - переменная

Регистры. Стек






Стек

- LIFO/FILO (last in, first out) - последним пришёл, первым ушёл
- Сегмент стека - область памяти программы, используемая её подпрограммами, а также (вынужденно) обработчиками прерываний
- SP - указатель на вершину стека
- В x86 стек "растёт вниз", в сторону уменьшения адресов. При запуске программы SP указывает на конец сегмента



Команды непосредственной работы со стеком

- PUSH <источник> - поместить данные в стек. Уменьшает SP на размер источника и записывает значение по адресу SS:SP.
- POP <приёмник> - считать данные из стека. Считывает значение с адреса SS:SP и увеличивает SP.
- PUSHA - поместить в стек регистры AX, CX, DX, BX, SP, BP, SI, DI.
- POPA - загрузить регистры из стека (SP игнорируется)
- PUSHF - поместить в стек содержимое регистра флагов
- POPF - загрузить регистр флагов из стека



CALL - вызов процедуры, RET - возврат из процедуры

CALL <операнд>

- Сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента)
- Передаёт управление на значение аргумента.

RET/RETN/RETF <число>

- Загружает из стека адрес возврата, увеличивает SP
- Если указан операнд, его значение будет дополнительно прибавлено к SP для очистки стека от параметров

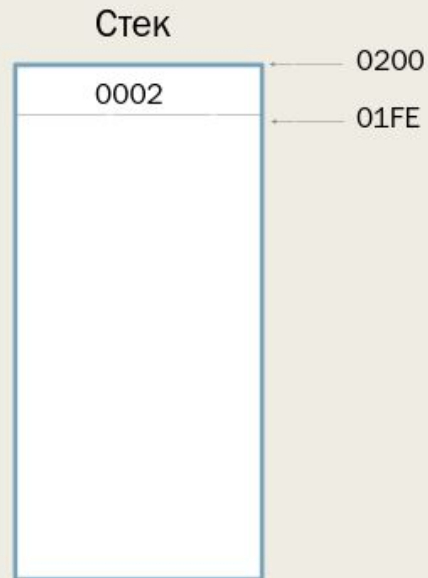


BP – base pointer

- Используется в подпрограмме для сохранения "начального" значения SP
- Адресация параметров
- Адресация локальных переменных

Пример вызова подпрограммы №1

```
0. SP = 0200  
0000: CALL P1    1. SP = 01FE  
0002: MOV BX, AX  
...  
P1:  
0123: MOV AX, 5  
0125: RET        2. SP = 0200
```

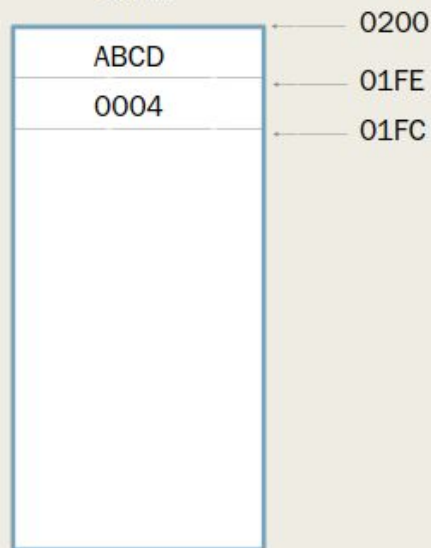


Пример вызова подпрограммы N°2

```
0000: PUSH ABCDh ;передача параметра
0002: CALL P1
0004: POP DX
0006: MOV BX, AX
...
P1:
0123: MOV BP, SP ;ss:[bp] - адрес возврата
; ss:[bp+2] - параметр
...
0223: MOV AX, 5
0225: RET
```

0. SP = 0200
1. SP = 01FE
2. SP = 01FC
4. SP = 0200
3. SP = 01FE

Стек



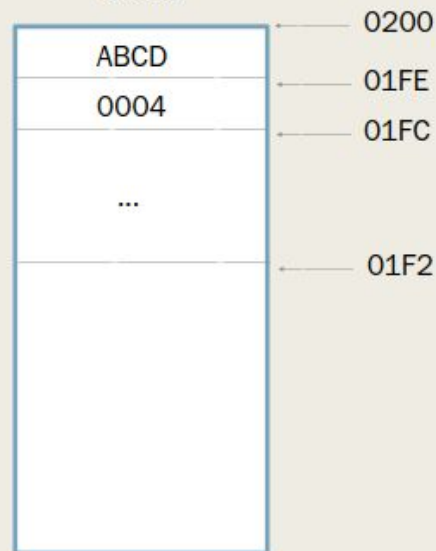
Пример вызова подпрограммы №3

```
0. SP = 0200

0000: PUSH ABCDh ; передача параметра
0002: CALL P1
0004: MOV BX, AX
...
P1:
0123: MOV BP, SP ; ss:[bp] - адрес возврата
; ss:[bp+2] - параметр
0125: SUB SP, 10 ; ss:[bp-1 .. bp-10] - локальные переменные
...
0221: ADD SP, 10
0223: MOV AX, 5
0225: RET 2

3. SP = 01F2
4. SP = 01FC
5. SP = 0200
```

Стек





Использование стека подпрограммами

Стековый кадр (фрейм) — механизм передачи аргументов и выделения временной памяти с использованием аппаратного стека. Содержит информацию о состоянии подпрограммы.

Включает в себя:

- параметры
- адрес возврата (обязательно)
- локальные переменные



Соглашения о вызовах (calling conventions)

Описания технических особенностей вызова подпрограмм, определяющие:

- способ передачи параметров подпрограммам;
- способ передачи управления подпрограммам;
- способ передачи результатов выполнения из подпрограмм в точку вызова;
- способ возврата управления из подпрограмм в точку вызова.



Распространённые соглашения

- cdecl
- pascal
- stdcall (Win32 API)
- fastcall
- thiscall - вызов нестатических методов C++ (this через ECX)
- Microsoft x64