5. Строковые и прочие операции

Префиксы повторения строковых операций. Количество повторений – в регистре CX, кроме того, REPZ/REPE повторяет, пока ZF равен нулю, а REPNZ/REPNE – пока не равен.

1-й префикс: REPE (repeat if equal; повторять, пока равно),
REPZ (повторять, пока ноль),
REP (повторять)

2-й префикс: REPNE (repeat if not equal; повторять пока не равно),
REPNZ (повторять, пока не ноль)

Все перечисленные команды являются префиксами для операций над строками. Любой из префиксов выполняет следующую за ним команду строковой обработки столько раз, сколько указано в регистре CX, уменьшая его при каждом выполнении команды на 1. Кроме того, REPZ и REPE прекращают повторения команды, если флаг ZF сброшен в О, а REPNZ и REPNE прекращают повторения, если флаг ZF установлен в 1. Префикс REP обычно используется с командами MOVS, LODS, STOS, а префиксы REPE, REPNE, REPZ и REPNZ - с командами CMPS и SCAS. Поведение префиксов в других случаях не определено.

Сравнение строк: CMPS
CMPS сравнивает пару элементов DS:SI с ES:DI. Также автоматически производит инкремент обоих указателей, при DF – 0.

[DS:SI]=[ES:DI]? ; SI:=SI+d; DI:=DI+d

	DF=0	*DF=1*
CMPSB	+1	-1
CMPSW	+2	-2

```
        CLD
        LEA SI, S1
        LEA DI, S2
        MOV CX, N
L:      CMPSB
        JNE NOEQ
        LOOP L
NOEQ:
```

Сравнивает один байт (CMPSB) или слово (CMPSW) из памяти по адресу DS:SI с байтом или словом по адресу ES:DI и устанавливает флаги аналогично команде CMP. При использовании формы записи CMPS ассемблер сам определяет по типу указанных операндов (принято указывать имена сравниваемых строк, но можно использовать любые два операнда подходящего типа), какую из двух форм этой команды (CMPSB или CMPSW) выбрать.
Применяя CMPS с операндами, можно заменить регистр DS другим, воспользовавшись префиксом замены сегмента (ES:, GS:, FS:, CS:, SS:), регистр ES заменить нельзя. После выполнения команды регистры SI и DI увеличиваются на 1 или 2 (если сравниваются байты или слова), когда флаг DF = 0, и уменьшаются, когда DF = 1.

Команда CMPS с префиксами REPNE/REPNZ или REPE/REPZ выполняет сравнение строки длиной в CX байтов или слов. В первом случае сравнение продолжается до первого совпадения в строках, а во втором — до первого несовпадения.

Сравнение строк: CMPS.

```
        CLD
        LEA SI, S1
        LEA DI, S2
        MOV CX, N
        REPE CMPSB
        JE EQ
NOEQ:
```

Поиск в строке: SCAS. REPNE SCASB ищет в строке элемент, равный AL. REPE SCASB ищет не равный.

AL=[ES:DI]? ; DI:=DI±1

Сравнивает содержимое регистра AL (SCASB) или AX (SCASW) с байтом, словом или двойным словом из памяти по адресу ES:DI и устанавливает флаги аналогично команде CMP.
При использовании формы записи SCAS ассемблер сам определяет по типу указанного операнда (принято указывать имя сканируемой строки, но можно использовать любой операнд подходящего типа), какую из двух форм этой команды (SCASB или SCASW) выбрать. После выполнения команды регистр DI увеличивается на 1 или 2 (если сканируются байты или слова), когда флаг DF = 0, и уменьшается, когда DF = 1. Команда SCAS с префиксами REPNE/REPNZ или REPE/REPZ выполняет сканирование строки длиной в CX байтов или слов. В первом случае сканирование продолжается до первого элемента строки, совпадающего с содержимым аккумулятора, а во втором - до первого отличного.

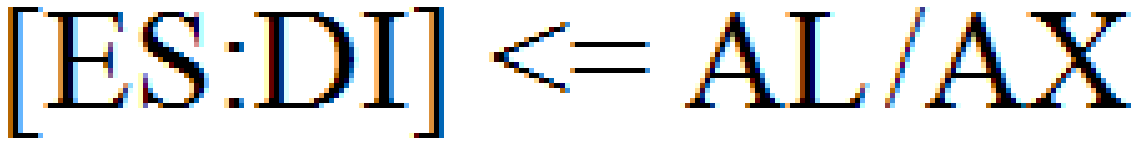
Пересылка строки: MOVS. Копирует байт/слово из DS:SI в ES:DI. Инкрементирует/декрементирует указатели.

[DS:SI] => [ES:DI]; SI:=SI+d; DI:=DI+d

Копирует один байт (MOVSB) или слово (MOVSW) из памяти по адресу DS:SI в память по адресу ES:DI.
При использовании формы записи MOVS ассемблер сам определяет по типу указанных операндов (принято указывать имена
копируемых строк, но можно применять любые два операнда подходящего типа), какую из двух форм этой команды (MOVSB или MOVSW) выбрать. Используя MOVS с операндами, разрешается заменить регистр DS другим с помощью префикса замены сегмента (ES:, GS:, FS:, CS:, SS:), регистр ES заменить нельзя. После выполнения команды регистры SI и DI увеличиваются на 1 или 2 (если копируются байты или слова), когда флаг DF = O, и уменьшаются, когда DF = 1. Команда MOVS с префиксом REP выполняет копирование строки длиной в CX байтов или слов.

```
 CLD
        LEA SI, Y
        LEA DI, X
        PUSH DS
        POP ES
        MOV CX, N
        REP MOVSB
…..
X DB N DUP(?)
Y DB N DUP(?)
```

Сохранение строки: STOS. Сохраняет содержимое AL/AX по адресу ES:DI. Изменяет указатели.

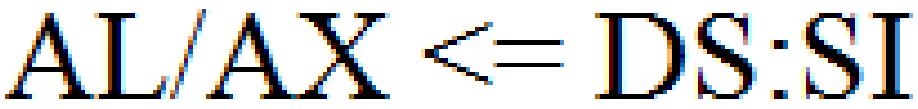


Копирует регистр AL (STOSB) или AX (STOSW) в память по адресу ES:DI. При использовании формы записи STOS ассемблер сам определяет по типу указанного операнда (принято указывать имя строки, но можно использовать любой операнд подходящего типа), какую из двух форм этой команды (STOSB или STOSW) выбрать. После выполнения команды регистр DI увеличивается на 1 или 2 (если копируется байт или слово), когда флаг DF = O, и уменьшается, когда DF = 1. Команда STOS с префиксом REP заполнит строку длиной в CX числом, находящимся в аккумуляторе.

```
CLD
LEA DI, MEM
STOSB                ; сохранить AL

LEA DI, X
CLD
XOR AL, AL
MOV CX, N
REP STOSB            ; обнулить массив
X DB N DUP (?)
```

Загрузка строки: LODS. Считывает из адреса DS:SI в AL/AX.




```
Считать значение в регистр AL/AX из памяти по адресу DS:SI
LEA SI, MEM
LODSB                ; AL = *(BYTE *)MEM
```

Копирует один байт (LODSB) или слово (LODSW) из памяти по адресу DS:SI в регистр AL или AX соответственно. При использовании формы записи LODS ассемблер сам определяет по типу указанного операнда (принято указывать имя строки, но можно использовать любой операнд подходящего типа), какую из двух форм этой команды (LODSB или LODSW) выбрать. Применяя LODS с операндом, можно заменить регистр DS на другой с помощью префикса замены сегмента (ES:, GS:, FS:, CS:, SS:). После выполнения команды регистр SI увеличивается на 1 или 2 (если считывается байт или слово), когда флаг DF = 0, и уменьшается, когда DF = 1. Команда LODS с префиксом REP выполнит копирование строки длиной в CX, и в аккумуляторе окажется последний элемент строки. На самом деле LODS используют без префиксов, часто внутри цикла в паре с командой STOS, так, что LODS считывает число, другие команды выполняют над ним какие-нибудь действия, а затем STOS записывает измененное число на прежнее место в памяти.

Скопировать из массива X в массив Y числа с противоположным знаком.

```
        CLD
        LEA SI, X
        PUSH DS
        POP ES
        LEA DI, Y
        MOV CX, N
L:      LODSB
        NEG AL
        STOSB
        LOOP L
```

Команды управления флагами:
STC/CLC/CMC - установить carry-флаг в 1, 0 или инвертировать его.
STD/CLD - установить флаг направления в 1 или 0. Необходимо для работы со строковыми операциями.
LAHF/SAHF - загрузить флаги в AH/из AH. LAHF копирует младший байт регистра FLAGS в AH, включая флаги SF (бит 7), ZF (бит 6), AF (бит 4), PF (бит 2) и CF (бит 0). Бит 1 устанавливается в 1, биты 3 и 5 - в 0. SAHF загружает флаги SF, ZF, AF, PF и CF из регистра AH значениями битов 7,6,4,2 и 0 соответственно. Зарезервированные биты 1,3 и 5 регистра флагов не изменяются.
PUSHF/POPF - кладет регистр FLAGS в стек или снимает его оттуда. Занимает 2 байта.
CLI/STI - установить флаг interrupt (флаг прерываний) в 0 или 1. При 0 процессор игнорирует все прерывания, 1 - обрабатывает.
SALC - устанавливает AL в OFFh, если флаг CF = 1, и сбрасывает в OOH, если CF = 0.

Загрузка адресных пар.
LDS, LES, LSS приемник, источник.
Второй операнд (источник) для всех этих команд - переменная в памяти размером в 32 бит. Первые 16 бит из этой переменной загружаются в регистр общего назначения, указанный
в качестве первого операнда, а следующие 16 бит - в соответствующий сегментный регистр (DS для LDS, ES для LES и т. д.).
Чтобы закинуть в **S**I 0x1122, а в **D**S – 0x3344:

```
LDS SI, Addr
……..
Addr db 22h,11h,  44h,33h
```

Чтобы закинуть в **D**I 0x1122, а в **E**S – 0x3344:

```
LES DI, Addr
……..
Addr db 22h,11h,  44h,33h
```

Прочие команды
NOP - отсутствие операции
NOP - однобайтная команда (код 90h), которая не выполняет ничего, только занимает место и время. Код этой команды фактически соответствует XCHG AL,AL. Многие команды разрешается записать так, что они не будут приводить ни к каким действиям, например:

```
mov ax, ax; 2 байта
xchg ax, ax; 2 байта
lea bx, [bx+0] ;3 байта, правда многие ассемблеры оптимизируют ее в 2-байтовую lea bx, [bx]
shl ax, 0 ; и так далее.
```

LOCK - префикс блокировки шины данных

На все время выполнения команды, снабженной таким префиксом, будет заблокирована шина данных, и если в системе присутствует другой процессор, он не сможет обращаться к памяти, пока не закончится выполнение команды с префиксом LOCK. Команда XCHG всегда выполняется автоматически с блокировкой доступа к памяти, даже если префикс LOCK не указан. Этот префикс можно использовать только с командами ADD, ADC, AND, DEC, INC, NEG, NOT, OR, SBB, SUB, XOR и XCHG.

HLT – команда останова

Почти каждый современный набор инструкций процессора включает в себя инструкцию ожидания, которая останавливает процессор до тех пор, пока не потребуется какая-либо работа. В процессорах с прерываниями эта инструкция останавливает работу CPU до тех пор, пока не будет получено внешнее прерывание. На большинстве архитектур выполнение такой команды позволяет процессору значительно сократить потребление энергии и, следовательно, тепловую мощность.

IN и OUT – работа с внешними устройствами через порты ввода-вывода

Команда IN принимает данные из порта, а команда OUT записывает данные в порт. Номер порта может задаваться как числом от 0 до 255, так и значением регистра от 0 до 65535. Читать из порта или записывать в порт можно как 8-битные, так и 16-битные данные.

Синтаксис: **IN op1,op2**

Операнды: **op1** - AL, AX
op2 - i8, DX

Назначение: Ввод из порта

Процессор: 8086+

Флаги: Не изменяются

Комментарий: Команда IN пересылает байт, слово или двойное слово из заданного числом во втором операнде порта в регистр (AL, AX или EAX), заданный первым операндом. Доступ к любому порту от 0 до 65535 выполняется путем помещения номера порта в регистр DX и использования команды IN с регистром DX в качестве второго параметра. Эти команды ввода/вывода могут быть несколько короче при использовании ввода/вывода через 8-битовый порт в команде. Старшие восемь битов адреса порта при использовании ввода/вывода через 8-битовый порт будут равны 0.

Ограничения: Нет

Примеры:

```
mov    dx,03DFh
in     al,dx
in     ax,60h
```

Синтаксис: **OUT op1,op2**

Операнды: **op1** - i8, DX
op2 - AL, AX, EAX

Назначение: Ввод из порта

Процессор: 8086+

Флаги: Не изменяются

Комментарий: Команда OUT пересылает байт, слово или двойное слово из заданного вторым операндом регистра (AL, AX или EAX) в выходной порт, номер которого задан первым операндом. Доступ к любому порту от 0 до 65535 выполняется путем

помещения номера порта в регистр DX и использования
команды OUT с регистром DX в качестве первого операнда.
Если команда содержит идентификатор 8-битового порта,
то значение расширяется нулем до 16 битов.

Ограничения: Нет

Примеры:
```
sub  ax,ax
mov  dx,03DFh
out  dx,ax
out  60h,al
```

INT – генерация прерывания
Инструкция на языке ассемблера для процессора архитектуры x86, генерирующая программное прерывание.
Синтаксис инструкции:
int n ,
где n — номер прерывания, которое будет сгенерировано. Как правило, номер прерывания записывается в виде шестнадцатеричного числа, т.е. с суффиксом h.
Часть прерываний зарезервирована для команд процессора, часть — для команд операционной системы MS-DOS (команды с номерами 20h–5Fh). Например, прерывание int 21h отвечает за доступ к большинству команд MS-DOS; перед тем, как вызвать данное прерывание, в регистр процессора ah должен быть помещен номер требуемой функции.
INT 3 — команда процессоров семейства x86, которая несёт функцию т. н. программного breakpoint, или точки останова. Исполнение команды приводит к вызову обработчика прерывания номер 3, зарезервированного для отладочных целей. В отличие от остальных команд вида INT N, которые кодируются двумя байтами, команда INT 3 кодируется только одним байтом с фиксированным кодом, хотя, конечно, двухбайтная инструкция (КОП INT + 03h) тоже будет работать.
Используется главным образом при отладке программ, отладчик может вставлять INT 3 в код отлаживаемой программы в точках останова.
INTO – вызов прерывания 4, если OF=1. Обработчик этого прерывания так или иначе работает с проблемами переполнения.
IRET – возврат из прерывания. По аналогии с CALL/RET, вызов прерывания оставляет на стеке адрес возврата; возврат из функции его оттуда снимает. То же самое происходит и во время INT/IRET, но помимо адреса возврата в виде IP, оттуда также снимаются CS и FLAGS.
RETN/RETF – на самом деле команды RET в ассемблере как таковой не существует, при транслировании она превращается в одну из двух. Они означают возврат из ближней процедуры (в том же сегменте, т.е. меняется только IP) и из дальней (меняется также и CS, т.е. в другом сегменте).

Более интересные приёмы программирования на ассемблере, включая возможность записывать управляющие конструкции языков высокого уровня

Структуры IF...THEN...ELSE
Эти часто встречающиеся управляющие структуры передают управления на один участок программы, если некоторое условие выполняется, и на другой, если оно не выполняется, записываются на ассемблере в следующем общем виде:

```
(проверка условия)
     Jcc else
(команды блока then)
     jmp endif
else:
     (команды блока else)
endif:
```

Для сложных условий часто оказывается, что одной командой условного перехода обойтись нельзя, поэтому реализация проверки может увеличиться. Например, строка на языке С

```
if (((x>y) && (z<t)) || (a!=b)) c=d;
```

может быть представлена на ассемблере так:

```
     mov ax, A
     cmp ax, B
     jne then  ; a!=b – условие выполнено
     mov ax, X
     cmp ax, Y
     jng endif ; x<=y – условие не выполнено
     mov ax, Z
     cmp ax, T
     jnl endif ; z>=t – условие не выполнено
then:
     mov ax, D
     mov C, ax
endif:
```

Структура CASE: проверяет значение некоторой переменной или выражения и передаёт управление на различные участки программы. Кажется очевидным, что такая структура должна реализовываться в виде серии структур IF…THEN…ELSE. Допустим, переменная I принимает значения 0, 1 и 2 и в зависимости от значения нужно выполнить процедуры case0, case1, case2:

```
     mov ax, I
     cmp ax, 0
     jne not0
     call case0
     jmp endcase
not0:
     cmp ax, 1
     jne not1
     call case1
     jmp endcase
not1:
     cmp ax, 2
     jne endcase
     call case2
endcase:
```

Однако, ассемблер в качестве более удобной альтернативы для реализации таких структур предлагает таблицу переходов.

```
     mov bx, I
     jmp cs:jump_table[bx]

jump_table dw foo0, foo1, foo2

foo0:
     call case0
     jmp endcase
foo1:
     call case1
     jmp endcase
foo2:
```

```
     call case2
endcase:
```

Конечно, нужно учитывать факт, что в bx может лежать неподходящее число и такой способ не сработает. Тем не менее, для переменной с большим числом значений способ с таблицей переходов является наиболее быстрым, поскольку не требует многочисленных проверок, а если значения переменной являются следующими друг за другом числами, и, как следствие, в таблице переходов нет пустых участков, то эта реализация структуры CASE окажется ещё и короче.

Несмотря на то, что в наборе команд 8086 есть команды огранизации циклов, они годятся только для циклов типа FOR, выполняемых фиксированное количество раз. В общем же виде любой ассемблерный цикл записывается как условный переход.

```
WHILE-цикл:

     (инициализация)
label:
     IF (не выполняется условие окончания цикла)
     THEN (команды тела цикла)
     ELSE jmp label2
     jmp label
label2:

REPEAT-UNTIL цикл

     (инициализация)
label:
     (тело цикла)
     IF (не выполняется условие окончания цикла)
     THEN jmp label

(такие циклы быстрее выполняются на ассемблере, всегда следует
стремиться переносить проверку условия окончания цикла в конец)

LOOP/ENDLOOP цикл

     (инициализация)
label:
     IF (выполняется условие окончания цикла)
     THEN jmp label2
     (команды тела цикла)
     jmp label
label2:
```

Способы передачи параметров в функции.

Передача параметров по значению

Процедуре передается собственно значение параметра. При этом фактически значение параметра копируется, и процедура использует его копию, так что модификация исходного параметра оказывается невозможной. Этот механизм применяется для передачи небольших параметров, таких как байты или слова, к примеру, если параметры передаются в регистрах:

```
mov     ax,word ptr value       ; Сделать копию значения.
call    procedure               ; Вызвать процедуру.
```

Передача параметров по ссылке

Процедуре передается не значение переменной, а ее адрес, по которому процедура сама прочитает значение параметра. Этот механизм удобен для передачи больших массивов данных и в тех случаях, когда процедура должна модифицировать параметры (хотя он и медленнее из-за того, что процедура будет выполнять дополнительные действия для получения значений параметров).

```
        mov     ax,offset value
        call    procedure
```

Передача параметров по возвращаемому значению

Этот механизм объединяет передачу по значению и по ссылке. Процедуре передают адрес переменной, а процедура делает локальную копию параметра, затем работает с ней, а в конце записывает локальную копию обратно по переданному адресу. Этот метод эффективнее обычной передачи параметров по ссылке в тех случаях, когда процедура должна обращаться к параметру очень большое количество раз, например, если используется передача параметров в глобальной переменной:

```
        mov     global_variable,offset value
        call    procedure
        [...]
procedure       proc    near
        mov     dx,global_variable
        mov     ax, word ptr [dx]
        (команды, работающие с AX в цикле десятки тысяч раз)
        mov     word ptr [dx],ax
procedure       endp
```

Передача параметров по результату

Этот механизм отличается от предыдущего только тем, что при вызове процедуры предыдущее значение параметра никак не определяется, а переданный адрес используется только для записи в него результата.

Более удобным (а также распространенным в языках высокого уровня) является способ передачи параметров через стек, рассмотренный на прошлой лекции.