# Системное программное обеспечение

*Лекция № 5 «Система команд процессора 8086»*

# Система команд процессора 8086

*Все команды процессора 8086 можно разделить на шесть групп:*

1. *Команды передачи данных*
2. *Арифметические команды*
3. *Логические команды*
4. *Команды управления потоком вычислений*
5. *Команды управления процессором*
6. *Команды для работы со строками*

# Команды передачи данных

| Формат | Назначение | Алгоритм работы | Флаги | | | | | | | | | Особенности | Примеры |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | of | df | if | tf | sf | zf | af | pf | cf | | |
| Команды общего назначения | | | | | | | | | | | | | |
| MOV dest, src (move – переслать) | Передача данных между регистрами или регистрами и памятью, а также передача непосредственных значений в регистр или память | Копирование источника в приемник | – | – | – | – | – | – | – | – | – | Запрещено:<br>1) передача из памяти в память<br>2) загрузка регистра сегмента непосредственными значениями<br>3) прямая передача между регистрами сегментов<br>4) CS в качестве приемника | MOV AX, SRC<br>MOV DST, AX<br>---<br>MOV AX, DAT<br>MOV DS, AX<br>---<br>MOV AX, ES<br>MOV DS, AX |
| PUSH src | Помещение значения источника в стек | Уменьшить значение указателя стека sp на 2; записать источник в вершину стека (адресуемую парой ss:sp) | – | – | – | – | – | – | – | – | – | 1) В стек помещается слово (регистр или ячейка памяти)<br>2) Стек растет в сторону уменьшения адресов памяти | PUSH ES<br>POP DS |
| POP dest | Извлечение данного из стека в приемник | Загрузить в приемник содержимое вершины стека (адресуется парой ss:sp); увеличить содержимое sp на 2 | – | – | – | – | – | – | – | – | – | 1) Из стека извлекается слово<br>2) Запрет CS в качестве приемника | |
| XCHG dest, src (exchange – обменять) | Обмен данными между регистрами или между регистрами и памятью | Копирование источника в приемник, а приемника в источник | – | – | – | – | – | – | – | – | – | Можно использовать для изменения порядка следования байт в слове, слов в двойном слове | CD DW 0f85ch<br>.....<br>MOV AL, CD<br>XCHG CD+1, AL<br>MOV CD, AL |
| XLAT tbl_src<br>XLATB (translate – переводить) | Извлечение элемента из таблицы байтов длиной до 256 элементов | **tbl_src** – имя таблицы. Перед исполнением XLAT ее начальный адрес надо загрузить в BX, а номер извлекаемого байта – AL. Результат помещается в AL | – | – | – | – | – | – | – | – | – | Можно использовать для выполнения перекодировок символов | mov al,10<br>mov bx,offset tab<br>xlatb |
| Команды ввода–вывода | | | | | | | | | | | | | |
| IN acc, port (input – ввести) | Ввод значения из порта ввода–вывода | Передать данное из порта в AL при обмене байтами или в AX при обмене словами. Операнд **port** или десятичное значение от 0 до 255 или значение в DX | – | – | – | – | – | – | – | – | – | Команды применяются для взаимодействия с периферийными устройствами системы | IN AL, 61H<br>OR AL, 3<br>OUT 61H, AL |
| OUT port, acc (output – вывести) | Вывод значения в порт ввода–вывода | Передать данное в порт из AL при обмене байтами или из AX при обмене словами. Операнд **port** или десятичное значение от 0 до 255 или значение в DX | – | – | – | – | – | – | – | – | – | | |

# Команды передачи данных

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Адресные команды** | | | | | | | | | | | | |
| LEA r16, m16 (load effective address – загрузить исполнительный адрес) | Получение эффективного адреса (смещения) ячейки памяти | В 16–разрядный регистр приемник загружается 16–битное значение смещения операнда источник, представляющего собой слово памяти | – | – | – | – | – | – | – | – | – | Альтернатива оператору ассемблера OFFSET. В отличие от OFFSET LEA допускает индексацию операнда **m16**, что позволяет более гибко организовать адресацию | M DB 10 DUP(0)<br>.....<br>MOV DI, 4<br>LEA BX,[M+DI] |
| LDS r16,m32 (load DS – загрузитьDS) | Формирование логического адреса ячейки памяти в виде пары сегмент-смещение | Считывает из памяти 32–битовое двойное слово и загружает первые 16 битов в заданный регистр, а следующие 16 битов – в регистр сегмента данных DS | – | – | – | – | – | – | – | – | – | Таким образом, с помощью данной команды в паре регистров ds и приемнике оказывается полный адрес некоторой ячейки памяти | H_F DD HERE<br>.....<br>LDS BX, H_F<br>; DS:[BX] |
| LES r16,m32 (load ES – загрузитьES) | | Считывает из памяти 32–битовое двойное слово и загружает первые 16 битов в заданный регистр, а следующие 16 битов – в регистр сегмента данных ES | – | – | – | – | – | – | – | – | – | Таким образом, с помощью данной команды в паре регистров es и приемнике оказывается полный адрес некоторой ячейки памяти | H_F DD HERE<br>.....<br>LES BX, H_F<br>; ES:[BX] |
| **Команды флагов** | | | | | | | | | | | | |
| LAHF (Load AH from Flags – загрузить AH флагами) | Чтение младшего байта регистра flags | Команда загружает регистр ah флагами cf, pf, af, zf и sf, находящимися в тех же позициях, что и в регистре flags | – | – | – | – | – | – | – | – | – | Из–за того, что регистр флагов непосредственно недоступен, команду lahf можно применять для анализа состояния некоторых флагов регистра flags | lahf<br>and ah,11111110b<br>sahf |
| SAHF (Store AH into Flags – загрузить флаги из AH) | Запись в младший байт регистра flags | Команда загружает младший байт регистра flags содержимым регистра ah. В битах 7, 6, 4, 2 и 0 регистра ah должны, соответственно, содержаться новые значения флагов sf, zf, af, pf и cf | – | – | – | – | * | * | * | * | * | Из–за того, что регистр флагов непосредственно недоступен, команду sahf можно применять для изменения состояния некоторых флагов регистра flags | |
| PUSHF | Размещение в вершине стека (ss:sp) содержимого регистра флагов flags | Уменьшить значение указателя стека sp на 2; поместить в вершину стека содержимое регистра flags | – | – | – | – | – | – | – | – | – | Позволяет получить доступ к регистру флагов как к содержимому обычного регистра | pushf<br>pop ax<br>xor ax,01h<br>push ax<br>popf |
| POPF | Извлечение из стека слова и восстановление его в регистр флагов flags | Извлечь из вершины стека слово и поместить его в регистр flags; увеличить значение указателя стека sp на 2 | * | * | * | * | * | * | * | * | * | | |

# Арифметические команды

| Формат | Назначение | Алгоритм работы | Флаги | | | | | | | | | Особенности | Примеры |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | of | df | if | tf | sf | zf | af | pf | cf | | |
| Команды сложения | | | | | | | | | | | | | |
| ADD dest, src (add – сложить) | Сложение целых чисел размерностью байт или слово | dest := dest + src | * | – | – | – | * | * | * | * | * | Вместе могут использоваться при сложении длинных двоичных чисел | ADD AX, CX<br>ADC BX, DX |
| ADC dest, src (add with carry – сложить с переносом) | Сложение целых чисел размерностью байт или слово с учетом переноса | dest := dest + src + cf | * | – | – | – | * | * | * | * | * | | |
| AAA (ASCII adjust after addition – ASCII коррекция после сложения) | Подгоняет двоичную сумму двух неупакованных BCD–цифр к неупакованному BCD–формату, который легко преобразуется в ASCII | Корректируемое значение должно быть в al. Если предыдущая инструкция ADD сгенерировала перенос или al больше 9, тогда ah увеличивается на 1, cf и af устанавливаются в 1; иначе af и cf устанавливаются в 0. Четыре старших бита al всегда обнуляются | ? | – | – | – | ? | ? | * | ? | * | Каждая цифра неупакованного BCD–числа занимает младший полубайт. Если результат сложения двух одноразрядных BCD–чисел больше 9, то число в младшем полубайте результата не есть BCD–число. Поэтому результат нужно корректировать командой aaa | MOV AH,08H<br>MOV AL,05H<br>ADD AL,AH<br>; AL=0DH<br>; не BCD–число<br>XOR AH,AH<br>AAA<br>; AH=01H,<br>; AL=03H<br>OR AX,3030H |
| DAA (decimal adjust after addition – десятичная коррекция после сложения) | Подгоняет двоичную сумму двух упакованных BCD–цифр к упакованному BCD–формату | Корректируемое значение должно быть в al. Если af и cf равны 1, тогда сумма больше десятичного 99. Если af=1, а cf=0, значит, сумма двух младших цифр была больше 9 и перенос автоматически учитывается в старшей цифре результата. Если af и cf равны 0, перенос не производился | ? | – | – | – | * | * | * | * | * | Эту команду следует применять после сложения двух упакованных BCD–чисел с целью корректировки получающегося двоичного результата сложения в правильное двузначное десятичное число. | MOV AL, 053H<br>MOV BL, 018H<br>ADD AL, BL<br>; AL = 06BH<br>DAA<br>; AL=071H |
| INC dest (increment – прирастить) | Уеличение значения операнда в памяти или регистре на 1 | dest := dest + 1 | * | – | – | – | * | * | * | * | – | Команда не влияет на флаг cf. | MOV DX, 0<br>LBL:<br>.....<br>INC DX<br>CMP DX,1000<br>JNE LBL |
| Команды вычитания | | | | | | | | | | | | | |
| SUB dest, src (subtract – вычесть) | Вычитание целых чисел размерностью байт или слово | dest := dest – src | * | – | – | – | * | * | * | * | * | Вместе могут использоваться при вычитании длинных двоичных чисел | SUB AX, CX<br>SBB BX, DX |
| SBB dest, src (subtract with borrow – вычесть с заемом) | Вычитание целых чисел размерностью байт или слово с учетом заема | dest := dest – src – cf | * | – | – | – | * | * | * | * | * | | |

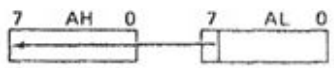
# Арифметические команды

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AAS (ASCII adjust after subtraction – ASCII коррекция после вычитания) | Подгоняет двоичную разность двух BCD неупакованных чисел к неупакованному формату BCD, который легко преобразуется в ASCII | Корректируемое значение должно быть в al. Если предшествующая операция sub требует заимствования, тогда aas также вычитает 1 из ah и устанавливает af и cf в 1; в противном случае ah остается неизменным и два флага устанавливаются в 0 | ? | – | – | – | ? | ? | * | ? | * | Каждая цифра неупакованного BCD–числа занимает младший полубайт. Если результат вычитания двух одноразрядных BCD–чисел больше 9, то число в младшем полубайте результата не есть BCD–число. Поэтому результат нужно корректировать командой aas | MOV AH, 01H<br>MOV AL, 04H<br>MOV BL, 07H<br>SUB AL, BL<br>AAS<br>; AX = 0007H<br>OR AX, 3030H |
| DAS (decimal adjust after subtraction – десятичная коррекция после вычитания) | Подгоняет двоичную разность двух упакованных BCD–цифр к упакованному BCD–формату | Корректируемое значение должно быть в al. Если af и cf равны 0, то во время вычитания потребовалось заимствование. Если cf=0 и af=l, то потребовалось заимствование для двух младших цифр и результат подгоняется соответствующим образом. Если cf=l, то результат является отрицательным десятичным дополнением | ? | – | – | – | * | * | * | * | * | Команду das следует применять после вычитания двух упакованных BCD–чисел с целью корректировки получающегося двоичного результата вычитания в правильное двузначное десятичное число | MOV AX 0107h<br>MOV BL, 14h<br>SUB AL, BL<br>DAS<br>; AL = 093H |
| DEC dest (decrement – уменьшить) | Уменьшение значения операнда в памяти или регистре на 1 | dest := dest – 1 | * | – | – | – | * | * | * | * | – | Команда не влияет на флаг cf. | MOV CX, 100<br>LBL:<br>.....<br>DEC CX<br>JNZ LBL |
| NEG dest (negate – отрицать) | Изменение знака (получение двоичного дополнения) операнда в памяти или регистре | dest := 0 – dest | * | – | – | – | * | * | * | * | * | Действие команды эквивалентно изменению всех битов с 0 на 1, с 1 на 0 и прибавлению затем 1 | MOV AL, 1<br>NEG AL<br>; AL = 0FFH |
| CMP dest, src (compare – сравнить) | Сравнение двух операндов | Вычесть src из dest; в зависимости от результата установить флаги, результат не сохранять | * | – | – | – | * | * | * | * | * | Оба операнда одновременно не могут быть ссылками к памяти. Обычно вслед за CMP следует инструкция перехода, осуществляя соответствующее действие, основанное на результатах сравнения | LEN EQU 10<br>...<br>CMP AX, LEN<br>JNE M1<br>JMP M2 |

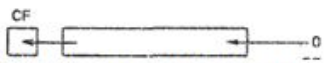
# Арифметические команды

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Команды умножения | | | | | | | | | | | | |
| MUL src (multiply - умножить) | Умножение двух целых чисел без знака | 1) если источник – байт, то второй сомножитель должен располагаться в al, а результат в ax;<br>2) если источник – слово, то второй сожножитель должен располагаться в ax, а результат в паре ds:ax | * | – | – | – | ? | ? | ? | ? | * | 1) После исполнения команды MUL флаги CF и OF равны 0, если старшая половина произведения равна 0; в противном случае оба этих флага равны 1<br>2) Источником не может быть непосредственное значение | MOV DX, 10<br>MUL DX |
| IMUL src (integer multiply - умножить целые числа) | Умножение двух целых чисел со знаком | | * | – | – | – | ? | ? | ? | ? | * | 1) После исполнения команды IMUL флаги CF и OF равны 0, если старшая половина произведения – расширение знака младшей половины; в противном случае оба этих флага равны 1<br>2) Источником не может быть непосредственное значение | MOV AL, 4<br>MOV BL, -2<br>IMUL BL<br>; AX = 0FFF8H,<br>; CF = OF = 0 |
| AAM (ASCII adjust after multiplication) | 1) Корректировка результата умножения двух неупакованных BCD-чисел;<br>2) преобразование двоичного числа меньшего $99_{10}$ в его неупакованный BCD-эквивалент | 1) корректируемое значение должно быть в al;<br>2) разделить значение регистра al на 10;<br>3) записать частное в регистр ah, остаток — в регистр al | ? | – | – | – | * | * | ? | * | ? | Чтобы команда AAM работала правильно, исходные множимое и множитель должны быть правильными неупакованными байтами | MOV AH, 08H<br>MOV AL, 09H<br>MUL AH<br>; AL=72<br>AAM<br>; AH=07H,<br>; AL=02H<br>OR AX,3030H |
| Команды деления | | | | | | | | | | | | |
| DIV src (divide – разделить) | Деление двух целых чисел без знака | 1) если делитель байт, то делимое должно быть расположено в регистре ax. После операции частное помещается в al, а остаток — в ah;<br>2) если делитель слово, то делимое должно быть расположено в паре регистров dx:ax. После операции частное помещается в ax, а остаток — в dx | ? | – | – | – | ? | ? | ? | ? | ? | 1) При выполнении операции деления возможно возникновение прерывания типа 0 — деление на 0. Эта ситуация возникает в одном из двух случаев: делитель равен 0 или частное слишком велико для его размещения в регистре ax/al<br>2) Источником не может быть непосредственное значение | MOV AX,10234<br>MOV BL,154<br>DIV BL |
| IDIV src (integer divide – разделить целые числа) | Деление двух целых чисел со знаком | | ? | – | – | – | ? | ? | ? | ? | ? | 1) При выполнении операции деления возможно возникновение прерывания типа 0 — деление на 0. Эта ситуация возникает в одном из двух случаев: делитель равен 0 или частное слишком велико для его размещения в регистре ax/al<br>2) Источником не может быть непосредственное значение<br>3) Остаток всегда имеет знак делимого | MOV AX, 100<br>MOV BL, -3<br>IDIV BL<br>; AL=-33, AH=1<br>NEG AL |

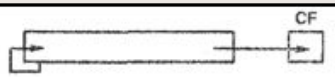
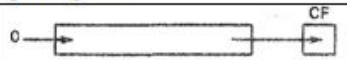
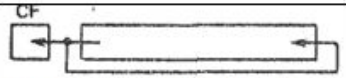
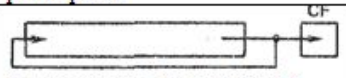
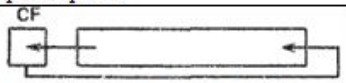
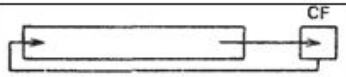
# Арифметические команды

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AAD (ASCII adjust before division – ASCII коррекция перед делением) | 1) Подготовка двух неупакованных BCD-чисел для операции деления;<br>2) преобразование двузначного неупакованного BCD-числа меньшего $99_{10}$ в двоичное представление | 1) Корректируемое значение в ax;<br>2) умножить значение регистра ah на 10 и сложить полученное значение с содержимым регистра al: (ah*10)+al;<br>3) присвоить регистру al значение (ah*10)+al;<br>4) обнулить регистр ah | ? | – | – | – | * | * | ? | * | ? | Наибольшее возможное значение, которое может преобразовывать aad — 0909H | MOV AX,0108H<br>MOV BL,09H<br>AAD<br>; AL=18<br>DIV BL<br>; AL=2, AH=0<br>OR AL,30H |
| Команды расширения знака | | | | | | | | | | | | |
| CBW (convert byte to word – преобразовать байт в слово) | Расширяет байт, имеющий знак, на слово, имеющее знак |  | – | – | – | – | – | – | – | – | – | Данные команды используются для приведения операндов к нужной размерности с учетом знака | CBW<br>ADD AX, BX<br>; AL + BX |
| CWD (convert word to double word – преобразовать слово в двойное слово) | Расширяет слово, обладающее знаком, до двойного слова, обладающего знаком |  | – | – | – | – | – | – | – | – | – | | CWD<br>IDIV BX<br>; AX/BX |

# Логические команды

| Формат | Назначение | Алгоритм работы | Флаги | | | | | | | | | Особенности | Примеры |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | of | df | if | tf | sf | zf | af | pf | cf | | |
| Логические операции | | | | | | | | | | | | | |
| AND dest, src (logical AND – логическое И) | Операция логического умножения приемника и источника | Побитовое «И» между приемником и источником с записью в приемник | 0 | – | – | – | * | * | ? | * | 0 | Удобно использовать для принудительного сброса определенных битов приемника | AND DL, 0FH<br>; 4 старших бита<br>; в 0 |
| OR dest, src (logical OR – логическое ИЛИ) | Операция логического сложения приемника и источника | Побитовое «ИЛИ» между приемником и источником с записью в приемник | 0 | – | – | – | * | * | ? | * | 0 | Удобно использовать для принудительной установки определенных битов приемника | MOV AL, 01H<br>OR BL, AL<br>; нулевой бит в 1 |
| XOR dest, src (logical XOR – исключающее ИЛИ) | Операция логического исключающего «ИЛИ» приемника и источника | Побитовое исключающее «ИЛИ» между приемником и источником с записью в приемник | 0 | – | – | – | * | * | ? | * | 0 | Удобно использовать для инвертирования определенных битов приемника | XOR BL, 0FFH<br>; инвертирование<br>; всех битов<br>XOR AX, AX<br>; обнуление AX |
| NOT dest (logical NOT – логическое НЕ) | Операция логического отрицания приемника и источника | Побитовое «НЕ» между приемником и источником с записью в приемник | – | – | – | – | – | – | – | – | – | Выполняет для операнда дополнение до 1 | FLAG DB 0FFH<br>...<br>NOT FLAG |
| TEST dest,src (test — проверить) | Операция логического умножения приемника и источника | Побитовое «И» между приемником и источником без записи в приемник | 0 | – | – | – | * | * | ? | * | 0 | Удобно использовать для получения информации о состоянии заданных битов операнда приемник | TEST AL, 01H<br>JNZ ML<br>; переход, если<br>; нулевой бит AL<br>; равен 1 |
| Команды сдвига | | | | | | | | | | | | | |
| SAL dest, cnt (shift arithmetic left – сдвиг арифметический влево) | Арифметический сдвиг приемника влево на число бит, указанное в счетчике |  <br>Счетчик может быть цифрой 1 или значением без знака в регистре CL | * | – | – | – | * | * | ? | * | * | Удобно использовать для умножения целочисленных операндов без знака на степени 2 | MOV CL, 4<br>SHL AX, CL<br>; x 16 |
| SHL dest, cnt (shift logical left – сдвиг логический влево) | Логический сдвиг приемника влево на число бит, указанное в счетчике | | | | | | | | | | | | |

# Логические команды

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SAR dest, cnt (shift arithmetic right – сдвиг арифметический вправо) | Арифметический сдвиг приемника вправо на число бит, указанное в счетчике |  Счетчик может быть цифрой 1 или значением без знака в регистре CL | * | – | – | – | * | * | ? | * | * | Можно использовать для деления целочисленных операндов со знаком на степени 2 | MOV CL, 4<br>SAR AX, CL<br>; /16 |
| SHR dest, cnt (shift logical right – сдвиг логический вправо) | Логический сдвиг приемника вправо на число бит, указанное в счетчике |  Счетчик может быть цифрой 1 или значением без знака в регистре CL | * | – | – | – | 0 | * | ? | * | * | Можно использовать для деления целочисленных операндов без знака на степени 2 | MOV CL, 4<br>SHR AX, CL<br>; /16 |
| Команды циклического сдвига | | | | | | | | | | | | |
| ROL dest, cnt (rotate left – сдвинуть влево циклически) | Циклический сдвиг приемника влево на число бит, указанное в счетчике |  Счетчик может быть цифрой 1 или значением без знака в регистре CL | * | – | – | – | – | – | – | – | * | Удобно использовать для передвижения битов в заданные позиции | MOV AX,0FF00H<br>MOV CL, 8<br>ROL AX, CL<br>или<br>ROR AX, CL<br>; AX = 00FFH |
| ROR dest, cnt (rotate right – сдвинуть вправо циклически) | Циклический сдвиг приемника вправо на число бит, указанное в счетчике |  Счетчик может быть цифрой 1 или значением без знака в регистре CL | * | – | – | – | – | – | – | – | * | | |
| RCL dest, cnt (rotate left through carry – сдвинуть влево циклически вместе с флагом переноса) | Циклический сдвиг приемника влево через флаг переноса на число бит, указанное в счетчике |  Счетчик может быть цифрой 1 или значением без знака в регистре CL | * | – | – | – | – | – | – | – | * | Особенность этих сдвигов в том, что он происходит с некоторой задержкой, так как очередной сдвигаемый бит оказывается на некоторое время вне операнда. В это время можно произвести его извлечение и (или) подмену | MOV CX, 16<br>XOR AL, AL<br>LBL:<br>RCL BX, 1<br>или<br>RCR BX, 1<br>ADC AL, 0<br>LOOP LBL |
| RCR dest, cnt (rotate right through carry — сдвинуть вправо циклически вместе с флагом переноса) | Циклический сдвиг приемника вправо через флаг переноса на число бит, указанное в счетчике |  Счетчик может быть цифрой 1 или значением без знака в регистре CL | * | – | – | – | – | – | – | – | * | | |

# Команды управления потоком вычислений

| Формат | Назначение | Алгоритм работы | Флаги | | | | | | | | | Особенности | Примеры |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | of | df | if | tf | sf | zf | af | pf | cf | | |
| Команды безусловной передачи управления | | | | | | | | | | | | | |
| CALL addr (call a procedure - вызвать процедуру) | Передача управления близкой или дальней процедуре с запоминанием в стеке адреса точки возврата | 1) процедура близкая (с атрибутом NEAR)— текущее ip сохраняется в стеке и затем заменяется смещением адреса процедуры;<br>2) процедура дальняя (с атрибутом FAR)— текущие cs и ip сохраняются в стеке и затем заменяются значениями сегмента и смещения адреса процедуры | – | – | – | – | – | – | – | – | – | Форматы вызова call:<br>1) прямой ближний (в пределах текущего сегмента кода);<br>2) прямой дальний (процедура в другом сегменте кода);<br>3) косвенный ближний (в пределах текущего сегмента кода с использованием переменной с адресом перехода);<br>4) косвенный дальний (процедура в другом сегменте кода с использованием переменной с адресом перехода) | sn proc near<br>.....<br>sn endp<br>sf proc far<br>.....<br>sf emdp<br>adrn dw sn<br>adrf dd sf<br>1) call sn<br>2) call far ptr sf<br>3) call ds:adrn<br>lea bx, adrn<br>call word ptr [bx]<br>4)<br>call far ptr ds:adrf<br>lea bx, adrf<br>call dword ptr [bx] |
| RET [cnt] (return from procedure - возвратиться из процедуры) | Возврат управления из процедуры вызывающей программе | Работа команды зависит от типа процедуры:<br>1) для процедур ближнего типа — восстановить из стека содержимое ip;<br>2) для процедур дальнего типа — последовательно восстановить из стека содержимое ip и сегментного регистра cs;<br>3) если команда имеет операнд, то указатель стека после выталкивания адреса возврата еще сдвигается на число байт, равное значению операнда | – | – | – | – | – | – | – | – | – | Транслятор определяет ближний или дальний вариант возврата из процедуры по способу описания процедуры | SUB PROC<br>.<br>.<br>.<br>RET ;RETN<br>SUB ENDP |
| RETN [cnt] (return from near procedure - возвратиться из ближней процедуры) | Возврат управления из ближней процедуры вызывающей программе | 1) восстановить из стека содержимое ip;<br>2) если команда имеет операнд, то указатель стека после выталкивания адреса возврата еще сдвигается на число байт, равное значению операнда | – | – | – | – | – | – | – | – | – | Для процедур с атрибутом NEAR или без атрибута | |
| RETF [cnt] (return from far procedure - возвратиться из дальней процедуры) | Возврат управления из дальней процедуры вызывающей программе | 1) последовательно восстановить из стека содержимое ip и сегментного регистра cs;<br>2) если команда имеет операнд, то указатель стека после выталкивания адреса возврата еще сдвигается на число байт, равное значению операнда | – | – | – | – | – | – | – | – | – | Для процедур с атрибутом FAR | |

# Команды управления потоком вычислений

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| JMP addr (jump unconditionally - перейти безусловно) | Безусловный переход как внутри текущего сегмента команд, так и за его пределы | 1) переход ближний — текущее ip заменяется смещением адреса перехода;<br>2) переход дальний — текущие cs и ip заменяются значениями сегмента и смещения адреса перехода | – | – | – | – | – | – | – | – | – | Форматы вызова JMP:<br>1) прямой ближний;<br>2) прямой дальний;<br>3) косвенный ближний;<br>4) косвенный дальний | 1) jmp addr<br>jmp short addr<br>jmp near ptr addr<br>2) jmp far ptr addr<br>3) jmp bx<br>jmp word ptr [bx]<br>4)<br>jmp dword ptr bx] |
| Команды условной передачи управления | | | | | | | | | | | | | |
| JA addr | Jump if Above — перейти, если выше | 1) если проверяемое условие истинно, то перейти к ячейке, обозначенной операндом;<br>2) если проверяемое условие ложно, то передать управление следующей команде;<br>3) адрес перехода должен находиться не далее -128 или +127 байтов от команды условной передачи управления | – | – | – | – | – | – | – | – | – | Условие перехода: CF=0 и ZF=0 | CMP AL, BL<br>JZ ZERO |
| JAE addr | Jump if Above or Equal — перейти, если выше или равно | | – | – | – | – | – | – | – | – | – | Условие перехода: CF=0 | |
| JB addr | Jump If Below — перейти, если ниже | | – | – | – | – | – | – | – | – | – | Условие перехода: CF=1 | |
| JBE addr | Jump If Below or Equal — перейти, если ниже или равно | | – | – | – | – | – | – | – | – | – | Условие перехода: CF=1 или ZF=1 | |
| JC addr | Jump If Carry — перейти, если перенос | | – | – | – | – | – | – | – | – | – | Условие перехода: CF=1 | |
| JCXZ addr | Jump if CX Is Zero — перейти, если значение регистра CX ранно 0 | | – | – | – | – | – | – | – | – | – | Условие перехода: CX=0 | |
| JE addr | Jump if Equal — перейти, если равно | | – | – | – | – | – | – | – | – | – | Условие перехода: ZF=1 | |
| JG addr | Jump if Greater — перейти, если больше | | – | – | – | – | – | – | – | – | – | Условие перехода: ZF=0 и SF=OF<br>Для чисел со знаками | |
| JGE addr | Jump if Greater or Equal — перейти, если больше или равно | | – | – | – | – | – | – | – | – | – | Условие перехода: SF=OF<br>Для чисел со знаками | |
| JL addr | Jump if Less — перейти, если меньше | | – | – | – | – | – | – | – | – | – | Условие перехода: SF≠OF<br>Для чисел со знаками | |
| JLE addr | Jump if Less or Equal — перейти, если меньше или равно | | – | – | – | – | – | – | – | – | – | Условие перехода: ZF=1 или SF≠OF<br>Для чисел со знаками | |
| JNA addr | Jump if Not Above — перейти, если не выше | | – | – | – | – | – | – | – | – | – | Условие перехода: CF=1 или ZF=1 | |
| JNAE addr | Jump if Not Above nor Equal — перейти, если не выше и не равно | | – | – | – | – | – | – | – | – | – | Условие перехода: CF=1 | |
| JNB addr | Jump if Not Below — перейти, если не ниже | | – | – | – | – | – | – | – | – | – | Условие перехода: CF=0 | |

# Команды управления потоком вычислений

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| JNBE addr | Jump if Not Below nor Equal — перейти, если не ниже и ие равно | | – | – | – | – | – | – | – | – | – | Условие перехода: CF=0 и ZF=0 |
| JNC addr | Jump if No Carry — перейти, если пет переноса | | – | – | – | – | – | – | – | – | – | Условие перехода: CF=0 |
| JNE addr | Jump if Not Equal — перейти, если не равно | | – | – | – | – | – | – | – | – | – | Условие перехода: ZF=0 |
| JNG addr | Jump if Not Greater — перейти, если не больше | | – | – | – | – | – | – | – | – | – | Условие перехода: ZF=1 или SF≠OF<br>Для чисел со знаками |
| JNGE addr | Jump if Not Greater nor Equal — перейти, если не больше и не равно | | – | – | – | – | – | – | – | – | – | Условие перехода: SF≠OF<br>Для чисел со знаками |
| JNL addr | Jump if Not Less — перейти, если не меньше | | – | – | – | – | – | – | – | – | – | Условие перехода: SF=OF<br>Для чисел со знаками |
| JNLE addr | Jump if Not Less nor Equal — перейти, если не меньше и не равно | | – | – | – | – | – | – | – | – | – | Условие перехода: ZF=0 и SF=OF<br>Для чисел со знаками |
| JNO addr | Jump if No Overflow — перейти, если нет переполнения | | – | – | – | – | – | – | – | – | – | Условие перехода: OF=0<br>Для чисел со знаками |
| JNP addr | Jump if No Parity — перейти, если нет четности | | – | – | – | – | – | – | – | – | – | Условие перехода: PF=0 |
| JNS addr | Jump if No Sign - перейти, если знаковый разряд нулевой | | – | – | – | – | – | – | – | – | – | Условие перехода: SF=0<br>Для чисел со знаками |
| JNZ addr | Jump if Not Zero — перейти, если не нуль | | – | – | – | – | – | – | – | – | – | Условие перехода: ZF=0 |
| JO addr | Jump if Overflow - перейти, если переполнение | | – | – | – | – | – | – | – | – | – | Условие перехода: OF=1<br>Для чисел со знаками |
| JP addr | Jump if Parity — перейти, если есть четность | | – | – | – | – | – | – | – | – | – | Условие перехода: PF=1 |
| JPE addr | Jump if Parity Even — перейти, если сумма битов четная | | – | – | – | – | – | – | – | – | – | Условие перехода: PF=1 |

# Команды управления потоком вычислений

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| JPO addr | Jump if Parity Odd — перейти, если сумма битов нечетная | | – | – | – | – | – | – | – | – | – | Условие перехода: PF=0 | |
| JS addr | Jump if Sign — перейти, если знаковый бит равен 1 | | – | – | – | – | – | – | – | – | – | Условие перехода: SF=1<br>Для чисел со знаками | |
| JZ addr | Jump if Zero — перейти, если нуль | | – | – | – | – | – | – | – | – | – | Условие перехода: ZF=1 | |
| Команды управления циклами | | | | | | | | | | | | | |
| LOOP addr (loop until Count complete — повторять цикл до конца счетчика) | Организация цикла со счетчиком в регистре cx | Уменьшает cx на 1, а затем осуществляет переход по указанному адресу, если cx не равен 0. Иначе следующая команда после LOOP | – | – | – | – | – | – | – | – | – | 1) Смещение метки, являющейся операндом loop, не должно выходить из диапазона -128...+127 байт<br>2) для предотвращения выполнения цикла при нулевом *cx* используется команду jcxz | MOV CX, 100<br>START:<br>.....<br>LOOP START |
| LOOPE addr (loop if equal — повторять цикл, если равно) | Организация цикла со счетчиком в регистре cx и с учетом флага zf | Уменьшает cx на 1, а затем осуществляет переход по указанному адресу, если cx не равен 0 и zf=1. Иначе следующая команда после LOOP | – | – | – | – | – | – | – | – | – | | |
| LOOPZ addr (loop if zero — повторять цикл, если нуль) | | | | | | | | | | | | | |
| LOOPNE addr (loop if not equal — повторять цикл, если не равно) | | Уменьшает cx на 1, а затем осуществляет переход по указанному адресу, если cx не равен 0 и zf=0. Иначе следующая команда после LOOP | – | – | – | – | – | – | – | – | – | | |
| LOOPNZ addr (loop if not zero — повторять цикл, если не нуль) | | | | | | | | | | | | | |

# Команды управления потоком вычислений

| Команды прерывания | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INT num (interrupt — прерывать) | Вызов подпрограммы обслуживания прерывания с номером прерывания, заданным операндом команды | 1. Помещает в стек регистр флагов.<br>2. Обнуляет флаги TF и IF для исключения пошагового режима исполнения команд и блокировки других маскируемых прерываний.<br>3. Помещает в стек значение регистра CS.<br>4. Вычисляет адрес вектора прерывания, умножая номер прерывания на 4.<br>5. Загружает второе слово вектора прерывания в регистр CS.<br>6. Помещает в стек значение указателя команд IP.<br>7. Загружает в указатель команд IP первое слово вектора прерывания. | – | – | 0 | 0 | – | – | – | – | – | Таблица векторов прерывания занимает первый килобайт памяти, т.е. область памяти с абсолютными адресами от 0 до 3FFH | MOV DL, 49<br>MOV AH,2<br>INT 21H |
| INTO (interrupt if overflow - прервать при переполнении) | Инициирование прерывания с номером 4, если установлен флаг of | 1) если of=0, то никаких действий производить не нужно — передать управление на следующую команду;<br>2) если of=1, то дальнейшие действия, как при команде int | – | – | 0 | 0 | – | – | – | – | – | Если предыдущая команда в программе может в результате своей работы установить флаг переполнения of (к примеру, арифметические команды), то для обнаружения и обработки такой ситуации можно использовать команду into | MOV BL, 100<br>IMUL BL<br>INTO |
| IRET (interrupt return - возврат после обработки прерывания) | Возврат из подпрограммы обслуживания прерывания | Последовательно извлекает из стека и затем восстанавливает содержимое регистров ip, cs, flags. Далее прерванная программа продолжается с точки прерывания | * | * | * | * | * | * | * | * | * | Используется одна и та же инструкция iret, независимо от того, было прерывание сгенерировано программно или аппаратно | My_1C PROC<br>……<br>IRET<br>My_1C ENDP |

# Команды управления процессором

| Формат | Назначение | Алгоритм работы | Флаги | | | | | | | | | Особенности | Примеры |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | of | df | if | tf | sf | zf | af | pf | cf | | |
| Управление флагами | | | | | | | | | | | | | |
| STC (set carry flag) | Управление флагом переноса | Установка флага переноса cf в 1 | – | – | – | – | – | – | – | – | 1 | Используется при работе с командами сдвига, арифметическими командами либо действиями по индикации обнаружения ошибок и различных ситуаций в программе | AnyProc PR0C<br>; Код процедуры<br>ErrExit: STC<br>RET<br>NoErrExit:<br>CLC<br>RET<br>AnyProc ENDP |
| CLC (clear carry flag) | | Сброс флага переноса cf | – | – | – | – | – | – | – | – | 0 | | |
| CMC (complement carry flag) | | Инвертирование значения флага переноса cf | – | – | – | – | – | – | – | – | * | | AnyProc PR0C<br>; Код процедуры<br>Exit: CMC<br>RET<br>AnyProc ENDP |
| STD (set direction flag) | Управление флагом направления | Установка флага направления df в 1 | – | 1 | – | – | – | – | – | – | – | Флаг используется для указания направления обработки строк. Если флаг DF равен 0, то после каждой операции над строкой значения индексных регистров SI и DI увеличиваются; если флаг DF равен 1, то они уменьшаются | STD ; DF = 1 |
| CLD (clear direction flag) | | Сброс флага направления df | – | 0 | – | – | – | – | – | – | – | | CLD ; DF = 0 |
| STI (set interrupt flag) | Управление флагом прерывания | Установка флага прерывания if в 1 | – | – | 1 | – | – | – | – | – | – | Часто используются в программах обработки прерываний | CLI<br>....<br>STI |
| CLI (clear interrupt flag) | | Сброс флага прерывания if | – | – | 0 | – | – | – | – | – | – | | |
| Внешняя синхронизация | | | | | | | | | | | | | |
| HLT (halt — остановка) | Остановка процессора до прерывания или перезагрузки | Переход в состояние остановки. Из этого состояния его можно вывести сигналами на входах RESET, NMI, INTR | – | – | – | – | – | – | – | – | – | Если для возобновления работы микропроцессора используется прерывание, то сохраненное значение пары cs:ip указывает на команду, следующую за hlt | CLI<br>HLT |
| WAIT (wait — ожидать) | Перевод процессора на холостой ход | Останов процессора до тех пор, пока контакт процессора "ЗАНЯТ" не будет активирован | – | – | – | – | – | – | – | – | – | В этом состоянии процессор способен обрабатывать прерывание, но после обработки возвращается в состояние холостого хода | CLI<br>WAIT |
| ESC op, src (escape — убежать) | Передача инструкции сопроцессору | Передача кода инструкции и спецификация приемника или источника для инструкции | – | – | – | – | – | – | – | – | – | Обеспечивает другим процессорам системы возможность получения своих команд из потока команд процессора | WAIT<br>ESC 8, AX |

# Команды управления процессором

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LOCK (lock the bus — замкнуть шину) | Назначает сигнал запирания шины для следующей инструкции | Заставляет процессор активизировать сигнал LOCK своей шины на все время исполнения этой команды. А пока сигнал LOCK активен, никакой другой процессор системы не может использовать шину | – | – | – | – | – | – | – | – | – | lock (замок) используется в качестве префикса инструкций, которые обращаются к памяти, разделяемой более, чем одним процессором | LOCK XCHG SEM, AL |
| Холостой ход | | | | | | | | | | | | | |
| NOP (no operation — нет операции) | Пустая команда | Не производит никаких действий | – | – | – | – | – | – | – | – | – | Команда nop, занимая один байт (90H), может использоваться для резервирования места в сегменте кода, организации программной задержки, для удаления кода без перетрансляции, трассировки программы и т.д. | nop ≡ xchg ax, ax |

*Команды обработки строк позволяют производить действия над блоками байтов или слов памяти на основе следующих общих для этой группы команд принципов:*

1. *Процессор 8086 предполагает, что строка-приемник находится в дополнительном сегменте, а строка-источник - в сегменте данных*
2. *Процессор адресует строку-приемник через регистр DI, а строку-источник - через регистр SI*

# Команды обработки строк

3. *Команды этой группы обрабатывают строку поэлементно (по одному байту или слову). Адрес текущего элемента строки-приемника ES:[DI], строки-источника DS:[SI]*
4. *Бит флага направления DF в регистре флагов процессора 8086 определяет направление обработки строк – от начала к концу или от конца к началу, т.е. будут ли 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 после исполнения каждой команды; если флаг DF равен 1, то они уменьшаются. Если элемент строки - байт, то значение изменяется на 1, если слово – на 2*

# Команды обработки строк

*Команды обработки строк предоставляют возможность выполнения пяти основных операций, называемых примитивами. Каждый примитив представлен тремя разными командами. Первая из них имеет один или два операнда, а две остальные не имеют операндов. Процессор 8086 может исполнять только те команды обработки строк, которые не имеют операндов. При трансляции программы Ассемблер всегда преобразует команду с операндами в одну из команд без операндов.*

*Эти примитивы (пересылка, сравнение, сканирование, загрузка и сохранение) описаны в таблице*

# Команды обработки строк

| Формат | Назначение | Алгоритм работы | Флаги | | | | | | | | | Особенности | Примеры |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | of | df | if | tf | sf | zf | af | pf | cf | | |
| **Префиксы повторения** | | | | | | | | | | | | | |
| REP | Циклическое повторение команды обработки строк со счетчиком в регистре cx | Уменьшает cx на 1, а затем повторяет команду справа, если cx не равен 0 | – | – | – | – | – | – | – | – | – | Префиксы имеют смысл только для команд обработки строк, заставляя их циклически выполняться и тем самым без организации внешнего цикла обрабатывать последовательности элементов фиксированной длины | |
| REPE<br>REPZ | Циклическое повторение команды обработки строк со счетчиком в регистре cx и с учетом флага zf | Уменьшает cx на 1, а затем повторяет команду справа, если cx не равен 0 и zf=1 | – | – | – | – | – | – | – | – | – | | |
| REPNE<br>REPNZ | | Уменьшает cx на 1, а затем повторяет команду справа, если cx не равен 0 и zf=0 | – | – | – | – | – | – | – | – | – | | |
| **Пересылка** | | | | | | | | | | | | | |
| MOVS dest, src (move string — переместить строку) | Пересылка строки-источника в строку-приемник | Копирование элемента данных из строки-источника в строку-приемник. Размер пересылаемого элемента данных зависит от спсособа описания строк-операндов в программе на Ассемблере | – | – | – | – | – | – | – | – | – | Для того чтобы эти команды можно было использовать для пересылки всей последовательности элементов, имеющих размерность байт или слово, необходимо использовать префикс повторения REP | CLD<br>LEA SI, SRC<br>LEA DI, ES:DST<br>MOV CX, 100<br>REP MOVSB |
| MOVSB | Пересылка строки байтов из источника в приемник | Копирование байта из строки-источника в строку-приемник | – | – | – | – | – | – | – | – | – | | |
| MOVSW | Пересылка строки слов из источника в приемник | Копирование слова из строки-источника в строку-приемник | – | – | – | – | – | – | – | – | – | | |
| **Сравнение** | | | | | | | | | | | | | |
| CMPS dest, src (compare strings — сравнить строки) | Сравнение строки-источника со строкой-приемником | Сравнение текущих элементов строки-источника и строки-приемника путем вычитания значения приемника из значения источника. Размер сравниваемых элементов данных зависит от спсособа описания строк-операндов в программе на Ассемблере | * | – | – | – | * | * | * | * | * | Для того чтобы эти команды можно было использовать для сравнения последовательности элементов, имеющих размерность байт или слово, необходимо использовать один из префиксов REPE/REPZ или REPNE/REPNZ | CLD<br>LEA SI, SRC<br>LEA DI, ES:DST<br>MOV CX, 100<br>REPNE CMPSB<br>JNE NOT_FND<br>……<br>……<br>NOT_FND: ……<br>…… |
| CMPSB | Сравнение строк байтов источника и приемника | Сравнение текущих байтов строки-источника и строки-приемника путем вычитания значения приемника из значения источника | * | – | – | – | * | * | * | * | * | | |
| CMPSW | Сравнение строк слов источника и приемника | Сравнение текущих слов строки-источника и строки-приемника путем вычитания значения приемника из значения источника | * | – | – | – | * | * | * | * | * | | |

# Команды обработки строк

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Сканирование | | | | | | | | | | | | |
| SCAS dest (scan string — сканировать строку) | Сканирование строки-приемника с целью поиска заданного элемента | Заданный элемент находится в аккумуляторе (для строк байтов в AL, для строк слов в AX). Сравнение элемента строки-приемника со значением в аккумуляторе осуществляется путем вычитания значения аккумулятора из значения приемника. Размер сравниваемых элементов данных зависит от способа описания строки-приемника в программе на Ассемблере | * | – | – | – | * | * | * | * | * | Для того чтобы эти команды можно было использовать для поиска значения в последовательности элементов, имеющих размерность байт или слово, необходимо использовать один из префиксов REPE/REPZ или REPNE/REPNZ | CLD<br>LEA DI, ES:DST<br>MOV CX, 100<br>MOV AL, ' '<br>REPNE SCASB<br>JNE NOT_FND<br>.....<br>.....<br>NOT_FND: .....<br>..... |
| SCASB | Сканирование строки байтов приемника с целью поиска заданного элемента | Заданный элемент находится в AL. Сравнение байта строки-приемника со значением в AL осуществляется путем вычитания значения AL из значения приемника | * | – | – | – | * | * | * | * | * | | |
| SCASW | Сканирование строки слов приемника с целью поиска заданного элемента | Заданный элемент находится в AX. Сравнение слова строки-приемника со значением в AX осуществляется путем вычитания значения AX из значения приемника | * | – | – | – | * | * | * | * | * | | |
| Загрузка и сохранение | | | | | | | | | | | | |
| LODS src (load string — загрузить строку) | Загрузка элемента из строки-источника | Загрузка элемента из строки-источника в аккумулятор: байта в AL, слова в AX. Размер загружаемого элемента зависит от способа описания строки-источника в программе на Ассемблере | – | – | – | – | – | – | – | – | – | Перед этими командами можно указать префикс повторения rep, но в этом нет особого смысла, так как обычно эту команду используют в некотором цикле для просмотра некоторой цепочки с элементами фиксированного размера | CLD<br>LEA SI, STR<br>MOV CX, 20<br>START:<br>LODSB<br>CALL processChr<br>LOOP START |
| LODSB | Загрузка байта из строки-источника | Загрузка байта из строки-источника в AL | – | – | – | – | – | – | – | – | – | | |
| LODSW | Загрузка слова из строки-источника | Загрузка слова из строки-источника в AX | – | – | – | – | – | – | – | – | – | | |
| STOS dest (store string — сохранить строку) | Сохранение элемента в строке-приемнике | Сохранение элемента из аккумулятора в строке-приемнике: байта из AL, слова из AX. Размер сохраняемого элемента зависит от способа описания строки-приемника в программе на Ассемблере | – | – | – | – | – | – | – | – | – | Перед этими командами можно указать префикс повторения rep, в этом случае появляется возможность работы с блоками памяти, заполняя их значениями в соответствии с содержимым аккумулятора | CLD<br>MOV AL, ' '<br>LEA DI, STR<br>MOV CX, 10<br>REP STOSB |
| STOSB | Сохранение байта в строке-приемнике | Сохранение байта из AL в строке-приемнике | – | – | – | – | – | – | – | – | – | | |
| STOSW | Сохранение слова в строке-приемнике | Сохранение слова из AX в строке-приемнике | – | – | – | – | – | – | – | – | – | | |