Лабораторная работа №6

Логические команды и команды сдвига

Цель работы: изучение команд поразрядной обработки данных.

**Теоретические сведения**

Система команд центрального процессора имеет следующие команды для логической обработки данных.

AND <операнд\_1>, <операнд\_2>. Команда выполняет поразрядную операцию умножения (конъюнкцию) над битами операндов и изменяет флаги. Результат записывается на место первого операнда: операнд\_1 = операнд\_1 И операнд\_2.

TEST <операнд\_1>, <операнд\_2>. Данная команда также, как и команда AND, выполняет побитовое умножение двух операндов, но она не изменяет первый операнд, а только устанавливает или сбрасывает флаги в зависимости от результата умножения.

OR <операнд\_1>, <операнд\_2>. Команда выполняет поразрядную операцию сложения (дизъюнкцию) над битами операндов и изменяет флаги регистра EFLAGS. Результат записывается на место первого операнда:

операнд\_1 = операнд\_1 ИЛИ операнд\_2.

ХOR <операнд\_1>, <операнд\_2>. Команда выполняет поразрядную операцию исключающего ИЛИ над битами операндов и изменяет флаги регистра EFLAGS.

операнд\_1 = операнд\_1 ИСКЛЮЧАЮЩЕЕ\_ИЛИ операнд\_2.

Команду AND удобно использовать для сброса определённых битов числа, команду OR – для установки, а XOR – для инвертирования.

AND AX, 1111111110111011b; Сброс 3-го и 7-го битов AX

OR CX, 0000100000000100b; Установка 3-го и 12-го битов CX

XOR DL, 00011111b ; Инвертирование младших пяти битов DL

NEG <операнд>. Команда выполняет инвертирование всех битов операнда.

SHL <операнд>, <количество\_сдвигов>. Команда выполняет логический сдвиг влево на количество разрядов, определяемое значением второго операнда. При каждом сдвиге младший бит первого операнда устанавливается в ноль, а старший бит переносится в флаг CF.

CF

0

SHR <операнд>, <количество\_сдвигов>. Команда выполняет логический сдвиг вправо на количество разрядов, определяемое значением второго операнда. При каждом сдвиге младший бит первого операнда переносится в флаг CF, а старший бит устанавливается в ноль.

CF

0

Команды логического сдвига можно использовать для деления или умножения беззнаковых чисел на 2*n*, где *n* – количество сдвигов.

SAL <операнд>, <количество\_сдвигов>. Команда выполняет арифметический сдвиг первого операнда влево. Команда SAL аналогична команде SHL, но в отличие от последней устанавливает флаг OF в случае смены знака (старшего бита) очередным выдвигаемым битом.

1

0

CF

0

OF

1

1

1

CF

0

OF

0

SAR <операнд>, <количество\_сдвигов>. Команда выполняет арифметический сдвиг первого операнда вправо. Старший бит при этом сдвиге остаётся неизменным, поэтому знак числа сохраняется. Младший бит первого операнда попадает в флаг переноса CF.

CF

Команды арифметического сдвига SAL и SAR удобно использовать соответственно для умножения и деления знаковых чисел на 2*n*. Умножение и деление путём сдвига осуществляется быстрее, чем командами MUL/IMUL и DIV/IDIV.

RCL <операнд>, <количество\_сдвигов>. Циклический сдвиг битов первого операнда влево через флаг переноса CF. При сдвиге старший бит операнда помещается в флаг переноса CF, а значение в CF становится младшим битом операнда.

CF

RCR <операнд>, <количество\_сдвигов>. Циклический сдвиг битов операнда вправо через флаг переноса CF. При каждом сдвиге младший бит операнда переносится в флаг CF, а значение CF переносится в старший бит операнда.

CF

ROL <операнд>, <количество\_сдвигов>. Команда выполняет циклический сдвиг разрядов операнда влево. При этом старший бит переносится сразу и в младший разряд операнда и в флаг CF.

CF

ROR <операнд>, <количество\_сдвигов>. Команда выполняет циклический сдвиг разрядов операнда вправо. Младший бит при сдвиге попадает в старший разряд операнда и в флаг переноса CF.

CF

Вышеперечисленные команды сдвига могут адресовать первый операнд как в памяти, так и в регистре. Второй операнд должен быть числом или регистром CL.

SAL EAX, 2 ; Сдвиг содержимого EAX на 2 разряда влево,   
т.е умножение на 22

SAR EAX, 3 ; Сдвиг вправо на 3 разряда, т.е. деление на 23 с сохранением знака

SHL EBX, 4 ; Умножение на 24 без сохранения знака

SHR x, 3 ; Сдвиг числа в памяти вправо, т.е. деление на 23 без сохранения знака

ROL DWORD PTR [ESI], 32 ; Циклический сдвиг числа в памяти размером 4 байта по адресу ESI

ROR EBX, CL ; Циклический сдвиг вправо на CL разрядов

SHLD <операнд\_1>, <операнд\_2>, <количество\_сдвигов>. Команда аналогична SHL, но сдвигает значение операнд\_1:операнд\_2 как единое целое. Содержимое второго операнда не изменяется. Команда, по сути, помещает старшие разряды второго операнда на место младших разрядов первого, сдвигая первый операнд влево на количество\_сдвигов.

…

CF

…

операнд\_1

операнд\_2

SHRD <операнд\_1>, <операнд\_2>, <количество\_сдвигов>. Сдвиг значения в операнд\_1:операнд\_2 вправо. Второй операнд не изменяется. Старшие биты первого операнда заполняются младшими битами первого.

…

CF

…

операнд\_1

операнд\_2

Для команд расширенного сдвига SHLD и SHRD первый операнд может адресоваться как операнд в памяти или в регистре, второй – в регистре, третий – иметь непосредственную адресацию или являться регистром CL.

SHRD EAX, EBX, 8

SHLD DWORD PTR [ESI], ECX, 5

SHRD x, EBX, CL

BT <операнд>, <номер\_бита>. Команда считывает бит с заданным номером из первого операнда в флаг переноса CF.

BTS <операнд>, <номер\_бита>. Значение бита с данным номером переносится в флаг CF, а сам бит устанавливается в 1.

BTR <операнд>, <номер\_бита>. Считывание значения бита с заданным номером в флаг переноса CF и сброс этого бита в 0.

BTC <операнд>, <номер\_бита>. Считывание бита с заданным номером в флаг переноса CF и инвертирование этого бита.

BSF <операнд\_1>, <операнд\_2>. Команда просматривает биты второго операнда от младшего к старшему с целью поиска первого бита, установленного в единицу, и помещает в первый операнд номер этого бита. Если такой бит не находится, т.е. второй операнд равен нулю, то флаг ZF устанавливается в 1, иначе в 0.

BSR <операнд\_1>, <операнд\_2>. Данная команда, также как и команда BSF, сканирует биты второго операнда, но делает это в обратном порядке, т.е. от старшего бита к младшему. Номер искомого единичного бита при этом всё равно отсчитывается от младшего бита. Номер найденного бита помещается в первый операнд. Если бит не находится, то флаг нуля ZF устанавливается в 0, иначе в 1.

Задания для выполнения к работе

1. Написать программу для вывода чисел на экран согласно варианту задания. При выполнении задания №1 все числа считать беззнаковыми. Написать и использовать функцию output(a) для вывода числа a на экран или в файл. Функция должна удовлетворять соглашению о вызовах. В функцию для вывода output передавать в качестве аргумента переменную размерности 32 или 64 бита, которой достаточно для хранения числа. К примеру, если в задании число указано как 15-разрядное, то аргументом функции должно быть число размером двойное слово, если 40-разрядное, то учетверённое слово. Функция должна выводить столько разрядов числа, сколько указано в задании, даже если старшие разряды равны нулю. Не допускается прямой перебор всех чисел с проверкой, удовлетворяет ли оно условию вывода (за исключением вариантов № 8, 12, 13). Числа выводить в порядке, который является удобным. Проверить количество выведенных чисел с помощью формул комбинаторики. В отчёт включить вывод формул и результаты работы программы.

2. Написать подпрограмму для умножения (multiplication) или деления (division) большого целого числа на 2*n* (в зависимости от варианта задания) с использованием команд сдвига. Подпрограммы должны иметь следующие заголовки:

multiplication(char\* a, int n, char\* res);

division(char\* a, int n, char\* res).

Входные параметры: *a* – адрес первого числа в памяти, *n* – степень двойки. Выходные параметры: *res* – адрес массива, куда записывается результат. В случае операции умножения, для массива *res* зарезервировать в два раза больше памяти, чем для множителей *a* и *b*. Числа *a*, *b*, *res* вывести на экран в 16-ричном виде. Подобрать набор тестовых данных для проверки правильности работы подпрограммы.

Пример выполнения задания:

|  |  |  |
| --- | --- | --- |
| Вари-ант | Задание №1 | Задание №2 |
| # | Вывести все 8-разрядные числа, в двоичном представлении которых есть одна единица, остальные – нули.  1: 00000001  2: 00000010  3: 00000100  … | 12 байт  умножение  деление  со знаком |

Напишем программу для решения задания №1.

.486

.model flat, stdcall

option casemap: none

include d:\masm32\include\kernel32.inc

include d:\masm32\include\msvcrt.inc

includelib d:\masm32\lib\kernel32.lib

includelib d:\masm32\lib\msvcrt.lib

.data

number\_format db "%d: ", 0

new\_line\_format db 13, 10, 0

.code

; Процедура для вывода двоичного представления 8-битного числа

; void output (unsigned int a). Процедура в качестве аргумента принимает не 8-разрядное, а 32-разрядное целое число, но в процедуре используется только младший байт числа, остальные игнорируются

output proc

;Сохранить в стеке значения регистров, которые будут использованы

PUSH EAX ; Запомнить EAX

PUSH EBX ; Запомнить EBX

PUSH ECX ; Запомнить ECX

XOR EBX, EBX ; Обнулить EBX

MOV AL, [ESP+4\*4] ; Взять из стека аргумент, т.е. число, которое нужно вывести в двоичном представлении

MOV ECX, 8 ; Чтобы вывести 8-битное число, необходим цикл. Помещаем в ECX количество итераций

j1:

ROL AL, 1 ; Сделать циклический сдвиг числа на один разряд влево. Таким образом старший бит попадёт на место младшего

MOV BL, AL ; BL = AL

AND BL, 00000001b ; Оставить только младший бит, остальные обнулить

ADD BL, '0' ; Прибавить к BL код символа "0"

PUSH EAX ; Команда для вывода символа на экран crt\_putch изменяет регистры EAX и ECX, поэтому нужно сохранить их в стеке

PUSH ECX

PUSH EBX ; Поместить выводимый символ в стек, т.е. передать его в качестве аргумента функции crt\_\_putch

CALL crt\_\_putch ; Вызвать функцию

ADD ESP, 4 ; Удалить аргумент из стека, так как функция crt\_\_putch этого не делает

POP ECX ; Восстановить ECX

POP EAX ; Восстановить EAX

LOOP j1 ; ECX = ECX - 1. Выполнять цикл пока ECX ≠ 0

POP ECX ; Восстановить ECX

POP EBX ; Восстановить EBX

POP EAX ; Восстановить EAX

RET 4 ; Возврат к основной программе и очистка стека от аргумента размером 4 байта

output endp

start:

MOV AL, 1 ; Поместить в AL первое выводимое число

MOV ESI, 1 ; ESI - счётчик итераций

j2:

PUSH ECX ; Сохранить в стеке ECX

PUSH EAX ; Сохранить в стеке EAX. Функция crt\_printf изменяет ECX и EAX

PUSH ESI ; Передать функции crt\_printf аргумент

PUSH offset number\_format

CALL crt\_printf

; Восстановить стек и регистры

ADD ESP, 8

POP EAX

POP ECX

INC ESI ; Увеличить счётчик итераций

; Вызов функции output с одним аргументом

PUSH EAX

CALL output

; Следующее число

ROL AL, 1 ; Сдвиг числа на один разряд влево

PUSH EAX

PUSH ECX

PUSH offset new\_line\_format

CALL crt\_printf

ADD ESP, 4

POP ECX

POP EAX

CMP ESI, 9

JNE j2 ; Выполнять цикл, пока ESI <> 9

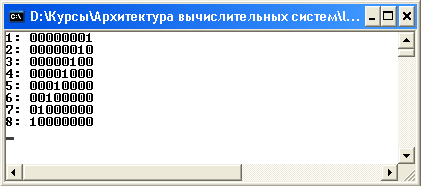
call crt\_\_getch ; Задержка ввода с клавиатуры

push 0

call ExitProcess ; Выход из программы

end start

Результат работы программы:



Если число 8-разрядное, то всего возможно 8 чисел, у которых в двоичном представлении только одна единица, значит результат работы программы верный.

|  |  |  |
| --- | --- | --- |
| Вари-ант | Задание №1 | Задание №2 |
| 1 | Вывести все 15-разрядные числа, в восьмеричном представлении которых есть одна цифра "**5**", одна цифра "7", остальные – "1".  1: 11157  2: 11517  3: 15117  … | 16 байт  умножение  со знаком |
| 2 | Вывести все 30-разрядные числа, в двоичном представлении которых есть две единицы, остальные нули.  1: 0000000000 0000000000 0000000011  2: 0000000000 0000000000 0000000101  3: 0000000000 0000000000 0000001001  … | 20 байт  деление  со знаком |
| 3 | Вывести все 20-разрядные числа, в 16-ричном представлении которых есть одна цифра "1", остальные – "E" или "F".  1: EEEE1  2: EEEF1  3: EEFE1  … | 18 байт  умножение  без знака |
| 4 | Вывести все 24-разрядные числа, в восьмеричном представлении которых есть пара цифр "1" и "2", стоящих рядом (цифра "2" следует за цифрой "1"), остальные – "3" или "5".  1: 33333312  2: 33333512  3: 33335312  … | 40 байт  деление  без знака |
| 5 | Вывести все 32-разрядные числа, в двоичном представлении которых есть два нуля, остальные единицы.  1: 1111111111111111 1111111111111100  2: 1111111111111111 1111111111111010  3: 1111111111111111 1111111111110110  … | 35 байт  умножение  со знаком |
| 6 | Вывести все 28-разрядные числа, в 16-ричном представлении которых есть одна цифра "1", одна цифра "2", остальные – "F".  1: FFFFF12  2: FFFF1F2  3: FFF1FF2  … | 30 байт  деление  со знаком |
| 7 | Вывести все 27-разрядные числа, в восьмеричном представлении которых есть две цифры "1", остальные – "7".  1: 777777711  2: 777777171  3: 777771771  … | 48 байт  умножение  без знака |
| 8 | Вывести все 12-разрядные числа, в двоичном представлении которых есть три единицы, остальные нули.  1: 000000 000111  2: 000000 001011  3: 000000 010011  … | 36 байт  деление  без знака |
| 9 | Вывести все 48-разрядные числа, в 16-ричном представлении которых есть только одна из цифр "3", "5", "7", "F", остальные – "A".  1: AAAAAAAAAAA7  2: AAAAAAAAAA7A  3: AAAAAAAAA7AA  … | 42 байта  умножение  со знаком |
| 10 | Вывести все 30-разрядные числа, в восьмеричном представлении которых есть две или одна цифры "5", остальные – "1".  1: 1111111115  2: 1111111155  3: 1111111515  … | 32 байта  деление  со знаком |
| 11 | Вывести все 16-разрядные числа, в двоичном представлении которых есть две или одна единицы, остальные – ноль.  1: 0000000000000001  2: 0000000000000011  3: 0000000000000101  … | 40 байт  умножение  без знака |
| 12 | Вывести все 16-разрядные числа, в 16-ричном представлении которых сумма цифр равна 5.  1: 1112  2: 0122  3: 0023  … | 30 байт  деление  без знака |
| 13 | Вывести все 18-разрядные числа, сумма цифр которых в восьмеричном представлении равна 4.  1: 001111  2: 000112  3: 000022  … | 35 байт  умножение  со знаком |
| 14 | Вывести все 34-разрядные числа, в двоичном представлении которых есть только 2, 3, 4, 5, 6 или 7 подряд идущих единиц, остальные – нули.  1: 0000000000 0000000000 0000000000 0011  2: 0000000000 0000000000 0000000000 0111  …  …: 0000000000 0000000000 1111111000 0000 | 45 байт  деление  со знаком |
| 15 | Вывести все 32-разрядные числа-палиндромы, в 16-ричном представлении которых только две цифры "A", "B", "C", "D", "E" или "F", остальные – "1".  1: A111 111A  2: 1A11 11A1  3: 11A1 1A11  …  …: B111 111B  … | 40 байт  умножение  без знака |
| 16 | Вывести все 30-разрядные числа, восьмеричное представление которых содержит 2, 3, 4, 5 или 6 подряд идущих одинаковых цифр от "1" до "7", остальные – нули.  1: 0000000011  2: 0000000110  3: 0000001100  …  …: 7777770000 | 32 байта  деление  без знака |
| 17 | Вывести все 50-разрядные числа, в двоичном представлении которых есть только 7, 8, 9 или 10 подряд идущих нуля, остальные – единицы.  1: 1111111111 1111111111 1111111111 1111111111 1110000000  2: 1111111111 1111111111 1111111111 1111111111 1100000001  …  …: 1111111111 0000000000 1111111111 1111111111 1111111111 | 28 байт  умножение  со знаком |
| 18 | Вывести все 32-разрядные числа, в 16-ричном представлении которых только одна или две цифры "E", остальные – "7".  1: 7777 777E  2: 7777 77EE  3: 7777 7EE7  …  …: EE77 7777  … | 37 байт  деление  со знаком |
| 19 | Вывести все 63-разрядные числа, в восьмеричном представлении которых есть 5 или 7 подряд идущих цифр "1", остальные – "7".  1: 7777777777 7777777777 7777777111 11  2: 7777777777 7777777777 7777711111 11  …  …: 7777777777 1111111777 7777777777 77  … | 22 байта  умножение  без знака |
| 20 | Вывести все 16-разрядные числа, двоичная запись которых является палиндромом и содержит 4 единицы, остальные – нули.  1: 11000000 00000011  2: 01100000 00000110  3: 10100000 00000101  … | 47 байт  деление  без знака |
| 21 | Вывести все 64-разрядные числа, в 16-ричном представлении которых есть только 3 или 4 подряд идущих цифры "5", остальные – "B".  1: BBBBBBBB BBBBB555  2: BBBBBBBB BBBB555B  3: BBBBBBBB BBB555BB  …  …: BBBBBB55 55BBBBBB  … | 36 байт  умножение  со знаком |
| 22 | Вывести все 36-разрядные числа, восьмеричное представление которых является палиндромом и содержит четыре цифры "2" и восемь цифр "3".  1: 223333 333322  2: 232333 333232  …  …: 233233 332332  … | 45 байт  деление  со знаком |
| 23 | Вывести все 32-разрядные числа, двоичная запись которых является палиндромом и содержит две или одну единицу, остальные – нули.  1: 10000000 00000000 00000000 00000001  2: 01000000 00000000 00000000 00000010  3: 11000000 00000000 00000000 00000011  … | 34 байта  умножение  без знака |
| 24 | Вывести все 24-разрядные числа, в 16-ричном представлении которых есть одна или две цифры "5", остальные – или все "F", или все E.  1: FFFFFF5  2: FFFFF55  3: EEEEEE5  4: EEEEE55  … | 38 байт  деление  без знака |
| 25 | Вывести все 42-разрядные числа, в восьмеричном представлении которых есть две цифры "0", остальные – или все "5", или все "7".  1: 55555 55555 5500  2: 55555 55555 5050  …  …: 77777 77777 7700  … | 22 байта  умножение  со знаком |