**ЛАБОРАТОРНАЯ РАБОТА 2 (ЧАСТЬ 2)**

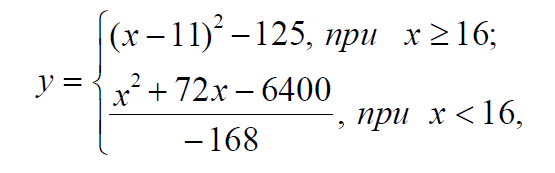
Для реализации алгоритмов, пути в которых зависят от исходных данных, используют команды условной передачи управления, которые рассмотрены в указаниях к первой части.

Во второй части лабораторной работы по программированию ветвящихся алгоритмов рассмотрим подробнее команды для выполнения арифметических операций.



Рисунок 1 – Алгоритм вычисления значения F(x)

В качестве примера рассмотрим программу вычисления функции



Cхема алгоритма решения задачи показана на рис.1.

**Арифметические команды**

*Команды сложения*

1. Команды **ADD** (add – сложить) и **ADC** (add with carry – сложить с переносом) могут складывать 8-, 16- и 32-битовые операнды. Формат команд: *ADD приемник, источник*

Команда ADD складывает содержимое операнда-источника и операнда-приемника и помещает результат в операнд-приемник. В символической нотации ее действия можно описать как

*приемник = приемник + источник*

Команда ADC делает то же, что и команда ADD, но при сложении использует также флаг переноса CF, что можно записать следующим образом:

*приемник = приемник + источник + перенос*

Пример. Если операнды имеют длину более 32 битов, то можно воспользоваться следующим приемом. Пусть складываются 64-битовое число, находящееся в регистрах EСХ и EDX, с 64-битовым числом, находящимся в регистрах EАХ и EВХ.

ADD EАX, ECX ; Сначала сложить младшие 32 бита,

ADC EBX, EDX ; а затем старшие 32 бита

Использованная здесь команда ADC добавляет к (EDX)+(EBX) любой перенос от сложения (EСХ)+(EАХ).

Можно добавлять находящийся в памяти операнд к регистру и наоборот или добавлять непосредственный операнд к регистру или операнду, находящемуся в памяти. Однако нельзя добавить значение одной ячейки памяти к другой или использовать в качестве приемника непосредственное значение.

2. Команда **XADD** имеет формат *XADD приемник, источник*

Команда сначала производит обмен данных (выполняет команду XCHG), а затем выполняет команду ADD.

3. Команда **INC** (increment – прирастить) добавляет 1 к содержимому регистра или ячейки памяти, но не воздействует на флаг переноса CF (в отличие от команды ADD).

*Команды вычитания*

1. Команды **SUB** (substract – вычесть) и **SBB** (substract with borrow – вычесть с заемом) аналогичны соответственно командам сложения ADD и ADC. Однако при вычитании флаг переноса CF действует как признак заема. Формат команд: *SUB приемник, источник*

Команда SUB вычитает операнд-источник из операнда-приемника и возвращает результат в операнд-приемник, т.е.

*приемник = приемник – источник*

Команда SBB делает то же самое, но дополнительно вычитает значение флага переноса CF:

*приемник = приемник – источник – перенос*

Как и в случае сложения, команды вычитания выполняют две отдельные функции. Первая команда SUB вычитает числа размером в байт, слово или двойное слово, а также младшие биты чисел повышенной точности. Другая команда SBB вычитает старшие биты чисел повышенной точности. Например, команда SUB АХ, СХ вычитает содержимое регистра СХ из содержимого регистра АХ и возвращает результат в регистр АХ.

Если размеры операндов превышают 32 бита, то пользуйтесь последовательностью команд вида:

SUB EАХ, EСХ ; Вычесть младшие 32 бита

SBB EBX, EDX ; а затем – старшие 32 бита

Здесь мы вычитаем 64-битовое число, помещенное в регистры EСХ и EDX, из 64-битового числа, помещенного в регистры EАХ и EВХ. При вычитании содержимого регистра EDX из содержимого регистра EВХ команда SBB учитывает возможность заема при выполнении первого вычитания.

Можно вычитать из содержимого регистра содержимое ячейки памяти (и наоборот) или вычитать из содержимого регистра либо ячейки памяти непосредственное значение. Нельзя непосредственно вычесть значение одной ячейки из другой или использовать непосредственное значение как приемник.

2. Команда **DEC** (decrement – уменьшить) вычитает 1 из содержимого регистра или ячейки памяти, но при этом не воздействует на флаг переноса CF (в отличие от команды SUB).

*Команда обращения знака*

Команда **NEG** вычитает значение операнда-приемника из нулевого значения. Команда NEG оказывает на флаги то же действие, что и команда SUB.

Команда NEG полезна для вычитания значения регистра или ячейки памяти из непосредственного значения.

Пример. Необходимо вычесть значение регистра AL из 100. Так как непосредственное значение не может служить приемником, то команда

SUB 100, AL

недопустима. В качестве альтернативы можно обратить знак содержимого регистра AL и добавить к нему 100:

NEG AL

ADD AL, 100

*Команды умножения*

Команда **MUL** (multiply – умножить) умножает числа без знака, a **IMUL** (integermultiply – умножить целые числа) – числа со знаком. Обе команды могут умножать байты, слова и двойные слова.

Эти команды имеют формат

*MUL источник*

*IMUL источник*

где источник – регистр общего назначения или ячейка памяти размером в байт, слово, или двойное слово.

В качестве второго операнда команды MUL и IMUL используют содержимое регистра AL (при операциях над байтами), регистра АХ (при операциях

над словами) или регистра EАХ (при операциях над двойными словами).

Произведение имеет двойной размер и возвращается следующим образом:

Умножение байтов возвращает 16-битовое произведение в регистрах АН (старший байт) и AL (младший байт).

Умножение слов возвращает 32-битовое произведение в регистрах DX (старшее слово) и АХ (младшее слово).

Умножение двойных слов возвращает 64-битовое произведение в регистрах EDX (старшее слово) и EАХ (младшее слово).

По завершении исполнения этих команд флаги переноса CF и переполнения OF показывают, какая часть произведения существенна для последующих операций.

После исполнения команды MUL флаги CF и OF равны 0, если старшая половина произведения равна 0; в противном случае оба этих флага равны 1.

После исполнения команды IMUL флаги CF и OF равны 0, если старшая половина произведения представляет собой лишь расширение знака младшей половины. В противном случае они равны 1.

Команды MUL и IMUL не позволяют в качестве операнда использовать непосредственное значение. Такое значение перед умножением надо загрузить в регистр или в ячейку памяти.

Пример. В результате исполнения команд

MOV DX, 10

MUL DX

содержимое регистра АХ будет умножено на 10.

Команда IMUL имеет также следующие расширенные форматы.

В команде IMUL с двумя операндами первый операнд является первым сомножителем, в него же записывается результат, второй операнд является вторым сомножителем. При этом первый операнд может быть 16- или 32-разрядным регистром, а второй операнд может быть регистром, ячейкой памяти или непосредственным значением того же размера (непосредственное значение может быть также 8-разрядным).

В команде IMUL с тремя операндами первый операнд является результатом умножения, второй и третий операнды являются сомножителями. При этом первый операнд может быть 16- или 32-разрядным регистром, а второй операнд может быть регистром или ячейкой памяти того же размера, третий операнд может быть только непосредственным значением того же размера или 8-разрядным.

Если результат выполнения этих команд слишком большой, то он усекается, а флаги OF и CF становятся равными 1, в противном случае флаги OF и CF становятся равными 0.

*Команды деления*

Команда DIV (divide – разделить) выполняет деление чисел без знака, а команда IDIV (integer divide – разделить целые числа) выполняет деление чисел со знаком.

Команда **DIV** выполняет деление целого числа без знака, находящегося в регистрах АХ (в случае деления на байт) или DX:AX (в случае деления на слово), на операнд-источник (целое число без знака). Размер делимого в два раза больше размеров делителя и остатка.

Для однобайтовых операций делимое помещается в регистр АХ; после выполнения операции частное записывается в регистр AL, остаток - в регистр АН.

Для двухбайтовых операций делимое помещается в регистры DX:AX (в DX - старшая часть, в АХ - младшая); после выполнения операции частное записывается в регистр АХ, остаток - в регистр DX.

В качестве операнда-делителя команды div можно указывать регистр (кроме сегментного) или ячейку памяти; не допускается деление на непосредственное значение. Если делитель равен 0, или если частное не помещается в назначенный регистр, возбуждается прерывание с вектором 0. Команда не воздействует на флаги процессора.

Команду div можно использовать для целочисленного деления неупакованного двоично-десятичного числа в регистре АХ не неупакованный двоично-десятичный делитель, если перед ней выполнить команду aad (см. пример 3).

Пример 1

*mov AX,506 ;Делимое*

*mov BL,50 ;Делитель*

*div BL ;AL=0Ah (частное), AH=06h (остаток)*

Пример 2

*; В полях данных*

*long dd 65537 ;Делимое*

*;В программном сегменте*

*mov DX,word ptr long+2 ;DX=0001h, старшая*

*;часть делимого*

*mov AX,word ptr long ;AX=0001h, младшая*

*;часть делимого*

*mov CX,256 ;Делитель*

*div CX ;AX=0100h (частное),*

*;DX=0001h (остаток)*

Пример З

*mov AX,0807h ;Неупакованное BCD 87*

*mov DL,09h ;Неупакованное BCD 9*

*aad ;AX=0057h=87*

*div DL ;AX=0609h, т.е. 9 и 6 в остатке*

Допустимо использование 32-битовых операндов и дополнительных режимов адресации 32-разрядных процессоров. При этом, если делитель представляет 32-битовую величину, то возможен только один вариант команды деления, когда делимое находится в парс регистров EDX:EAX. В этом случае частное будет помещено в регистр ЕАХ, остаток - в EDX. Для того, чтобы 32-битное делимое удвоить в размере, необходимо сформировать нулевую «старшую» часть, которую и поместить в регистр EDX (см. пример 4).

Пример 4

*mov ЕАХ, 0FFFFFFFh ;Младшая часть делимого*

*mov EDX,0 ;Старшая часть делимого*

*mov EBX,256 ;Делитель*

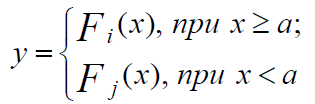
*div EBX ;Частное в EAX=000FFFFFh,*

*;Остаток в EDX=000000FFh*

Команды DIV и IDIV не позволяют прямо разделить на непосредственное значение; его надо предварительно загрузить в регистр или ячейку памяти.

***ЗАДАНИЕ***

Разработать программу вычисления и вывода значения функции



Параметр Х вводится в консоли.

Таблица 1 - Функции для индивидуальных заданий

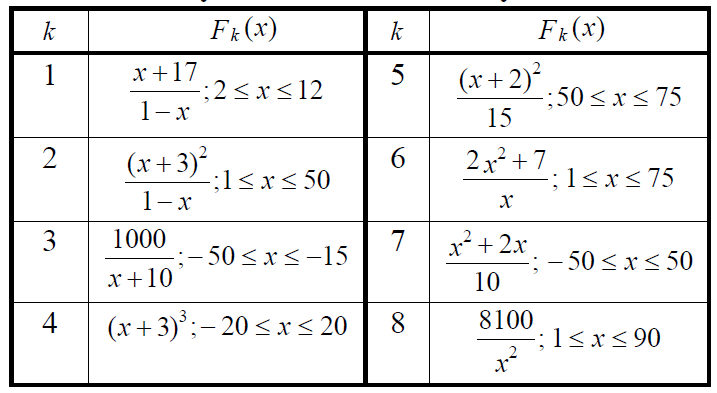


Таблица 2 – Варианты заданий

