**Лабораторная работа № 2**

Тема: Целочисленные арифметические операции. Обработка массивов данных. Интерфейс с языками высокого уровня.

1. Цель работы

Ознакомиться с арифметическими операциями над целочисленными данными,

обработкой массивов чисел, ознакомиться с правилами оформления ассемблерных процедур, в том числе вызываемых из Си-программ.

2. Краткие теоретические сведения

Основные арифметические операции над целыми числами выполняются с помощью следующих команд:

1)Сложение – ADD, ADC. Команда ADD выполняет арифметическое сложение

приемника и источника, помещает сумму в приемник, не изменяя содержимое

источника. Приемник может быть регистром или переменной, источник может быть числом, регистром или переменной, но нельзя использовать переменную одновременно и для источника, и для приемника. Команда ADD никак не различает числа со знаком и без знака, но, употребляя значения флагов CF

(перенос при сложении чисел без знака), OF (перенос при сложении чисел со

знаком) и SF (знак результата), можно использовать ее и для тех, и для других. Команда ADC во всем аналогична ADD, кроме того, что она выполняет арифметическое сложение приемника, источника и флага СF. Пара команд ADD/ADC используется для сложения чисел повышенной точности.

2)Вычитание – SUB, SBB. Команда SUB вычитает источник из приемника и

помещает разность в приемник. Приемник может быть регистром или переменной, источник может быть числом, регистром или переменной, но нельзя использовать переменную одновременно и для источника, и для приемника. Точно так же, как и команда ADD, SUB не делает различий между числами со знаком и без знака, но

флаги позволяют использовать ее как для тех, так и для других. Команда SBB во

всем аналогична SUB, кроме того, что она вычитает из приемника значение

источника и дополнительно вычитает значение флага CF.

3)Умножение – MUL, IMUL. Команда MUL выполняет умножение содержимого

источника (регистр или переменная) и регистра AL, АХ, ЕАХ (в зависимости от размера источника) и помещает результат в АХ, DX:AX, EDX:EAX соответственно.

Если старшая половина результата (АН, DX, EDX) содержит только нули (результат целиком поместился в младшую половину), флаги CF и OF устанавливаются в 0, иначе — в 1. Значение остальных флагов (SF, ZF, AF и PF)

не определено. Команда IMUL выполняет умножение с учетом знака. Эта команда имеет три формы, различающиеся числом операндов:

-- IMUL источник: источник (регистр или переменная) умножается на AL, АХ или

ЕАХ (в зависимости от размера операнда), и результат располагается в АХ,

DX:AX или EDX:EAX соответственно.

-- IMUL приемник,источник: источник (число, регистр или переменная)

умножается на приемник (регистр), и результат заносится в приемник.

-- IMUL приемник,источник1,источник2: источник 1 (регистр или переменная)

умножается на источник 2 (число), и результат заносится в приемник (регистр).

Во всех трех вариантах считается, что результат может занимать в два раза больше места, чем размер источника. В первом случае приемник автоматически оказывается достаточно большим, но во втором и третьем случаях могут произойти переполнение и потеря старших бит результата. Флаги OF и CF будут

равны единице, если это произошло, и нулю, если результат умножения

поместился целиком в приемник (во втором и третьем случаях) или в младшую

половину приемника (в первом случае). Значения флагов SF, ZF, AF и PF после

команды IMUL не определены.

4)Деление – DIV, IDIV. Команда DIV выполняет целочисленное деление без

знака AL, АХ или ЕАХ (в зависимости от размера источника) на источник (регистр

или переменная) и помещает результат в AL, АХ или ЕАХ, а остаток — в АН, DX или EDX соответственно. Результат всегда округляется в сторону нуля, абсолютное значение остатка всегда меньше абсолютного значения делителя. Значения флагов CF, OF, SF, ZF, AF и PF после этой команды не определены, а переполнение или деление на ноль вызывает прерывание 0. Команда IDIV выполняет целочисленное деление со знаком AL, АХ или ЕАХ (в зависимости от

размера источника) на источник (регистр или переменная) и помещает результат

вAL, АХ или ЕАХ, а остаток — в АН, DX или EDX соответственно. В остальном IDIV соответствует DIV.

Процедура в ассемблере – это аналог функции C, процедур и функций PASCAL

и т.п. Ассемблер не накладывает на процедуры никаких ограничений — на любой

адрес программы можно передать управление командой CALL, и оно вернется к вызвавшей процедуре, как только встретится команда RET. Такая свобода выражения легко может приводить к трудночитаемым программам, и в язык

ассемблера были включены директивы логического оформления процедур:

|  |  |
| --- | --- |
| метка proc | язык тип USES регистры |

...

ret

метка endp

Описание операндов PROC:

--*метка*– название процедуры.

*--тип*может принимать значения NEAR и FAR, и если он указан, все команды RET в теле процедуры будут заменены соответственно на RETN и RETF. По умолчанию подразумевается, что процедура имеет тип NEAR в моделях памяти

TINY, SMALL и COMPACT.

--*язык*действует аналогично такому же операнду директивы .MODEL, определяя взаимодействие процедуры с языками высокого уровня. В некоторых

ассемблерах директива PROC позволяет также считать параметры,

передаваемые вызывающей программой. В этом случае указание языка необходимо, так как различные языки высокого уровня используют разные способы передачи параметров.

*--USES регистры*— список регистров, значения которых изменяет процедура. Ассемблер помещает в начало процедуры набор команд PUSH, а перед командой RET — набор команд POP, так что значения перечисленных регистров будут восстановлены.

Параметры в процедуры можно передавать в регистрах, в глобальных переменных, в стеке, в потоке кода, в блоке параметров. Одна из простых передач – передача параметров через регистры:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| mov | ax,word ptr value | ; | сделать | копию значения |
| call | procedure | ; | вызвать | процедуру |

Большинство языков высокого уровня, например С, передают параметры

вызываемой процедуре в стеке и ожидают возвращения параметров в регистре

АХ (ЕАХ) (иногда используется DX:AX (EDX:EAX), если результат не умещается в

одном регистре, и ST(0), если результат число с плавающей запятой). Для чтения параметров из стека обычно используют не команду POP, а регистр ВР, в который помещают адрес вершины стека после входа в процедуру:

|  |  |  |
| --- | --- | --- |
| push | parameter1 | ; поместить параметр в стек |
| push | parameter2 |  |
| call | procedure |  |
| add | sp,4 | ; освободить стек от параметров |
| [...] |  |  |
| procedure | proc near |  |
| push | bp |  |
| mov | bp,sp |  |
| (команды, которые могут использовать стек) | | |
| mov | ax,[bp+4] | ; считать параметр 2. |

;Его адрес в сегменте стека ВР + 4, потому что при выполнении

;команды CALL в стек поместили адрес возврата - 2 байта для процедуры

;типа NEAR (или 4 - для FAR), а потом еще и ВР - 2 байта

|  |  |  |
| --- | --- | --- |
| mov | bx,[bp+6] | ; считать параметр 1 |
| (остальные команды) |  |  |
| рор | bp |  |
| ret |  |  |
| procedure | endp |  |

Для удобства ссылок на параметры, переданные в стеке, внутри функции иногда используют директивы EQU, чтобы не писать каждый раз точное смещение параметра от начала активационной записи (то есть от ВР), например так:

|  |  |  |
| --- | --- | --- |
|  | push | X |
|  | push | Y |
|  | push | Z |
|  | call | xyzzy |
|  | [...] |  |
| func | proc | near |
| func\_z | equ | [bp+8] |
| func\_y | equ | [bp+6] |
| func\_x | equ | [bp+4] |
|  | push | bp |
|  | mov | bp,sp |

(команды, которые могут использовать стек)

|  |  |  |
| --- | --- | --- |
| mov | ax,func\_x | ; считать параметр X |
| (остальные команды) |  |  |
| pop | bp |  |
| ret | 6 |  |
| func endp |  |  |

Язык С использует следующий порядок формирования параметров –

параметры помещаются в стек в обратном порядке, и удаление параметров из стека выполняет вызывающая процедура. При этом запись

some\_proc(a,b,c,d,e)

превращается в

push e

push d

push с

push b

push a

call some\_proc

|  |  |  |
| --- | --- | --- |
| add | sp,10 | ; освободить стек |

Вызванная таким образом процедура может инициализироваться так:

some\_proc proc push bp

|  |  |
| --- | --- |
| mov bp,sp | ; создать стековый кадр |
| a equ [bp+4] | ; определения для простого доступа к параметрам |
| b equ [bp+6] |  |
| с equ [bp+8] |  |
| d equ [bp+10] |  |
| e equ [bp+12] |  |
| ... | ; текст процедуры, использующей параметры a, b, с, d, e |

pop bp ret

some\_proc endp

Ассемблеры поддерживают и такой формат вызова при помощи усложненной формы директивы proc с указанием языка С:

.model small,с

some\_proc proc near uses si di, a:word, b:word local x:word, y:word

...

...

ret

some\_proc endp

Функции func соответствует следующий прототип в Си-программе:

int some\_proc(int a, int b);

При входе в ассемблерную процедуру в стеке будут сохранены регистры SI и

DI и размещены локальные переменные х и у. Доступ к этим данным организуется с помощью адресации по базе с использованием регистра ВР. При этом нет

необходимости вычислять смещения вручную, поскольку ассемблер

автоматически генерирует макроподстановки типа

a EQU <WORD PTR [bp+6]>

...

поэтому в тексте программы в качестве операндов можно использовать имена локальных переменных и передаваемых параметров. По команде RET автоматически генерируются команды восстановления регистров SI, DI, ВР, SP и затем только выполняется возврат в вызывающую программу.

В ассемблерной процедуре можно свободно использовать регистры AX, BX,

CX, DX. Остальные регистры должны быть сохранены и затем восстановлены.

# 3. Порядок выполнения работы

Написать программу на ассемблере, выполняющую арифметические

операции над целочисленными данными согласно заданиям, приведенным ниже. Общие требования:

--вид буфера для хранения массива и адресация в нем выбирается

самостоятельно;

--числовые данные вводятся с клавиатуры в виде строк символов (по

умолчанию используется 10 с/c), при этом производится проверка на

переполнение (по умолчанию на знаковые 16 битные данные);

--константы задать с помощью директивы EQU;

--вычисления проверять на возникновение ошибок и переполнений;

--старт программы, ввод-вывод данных и обработку ошибок оформлять

выводом в консоли поясняющих строк.

Задание:

|  |  |  |
| --- | --- | --- |
|  | Вычислить значения выражений. Все величины в выражении однобайтовые. Результат выдать на экран. | Вычислить значения выражений. Все величины в выражении однобайтовые. Результат выдать на экран. Если задать величины x, y, z. |
| 1. | 41X2+4Y2+2М2 | 2\*z+x\*y+15\*y\*z+3 x=1, y=2, z=3 |
| 2. | 5X2+5Y2+5М2 | z+x\*y-11\*x\*z+11 x=2, y=4, z=6 |
| 3. | 4X2+21Y2+9М2 | 3\*x+x\*z-17\*y\*z+30 x=3, y=9, z=27 |
| 4. | 3X2+35Y2+33М2 | 9\*z+3\*x\*y-15\*x\*y\*z+33 x=4, y=8, z=16 |
| 5. | X2+8Y2+4М2 | 2\*y+x\*z-5\*x\*y\*z-20 x=1, y=8, z=4 |
| 6. | 4X2+8Y2+8М2 | 5\*z+x\*y-35\*x\*z-90 x=4, y=8, z=8 |
| 7. | 11X2+22Y2+33М2 | 7\*x+x^2-15\*y\*z+75 x=11, y=22, z=33 |
| 8. | 1X2+2Y2+3М2 | 5\*y+z\*y-4\*x\*z+24 x=1, y=4, z=2 |
| 9. | 4X2+5Y2+6М2 | 15\*z-x\*y+21\*x\*y\*z+100 x=1, y=2, z=3 |
| 10. | 10X2+5Y2+2М2 | 8\*y+x\*z+99\*x-y\*z-100 x=2, y=4, z=6 |
| 11. | 24X2+18Y2+М2 | x+x\*z-73\*y\*z-7 x=3, y=9, z=27 |
| 12. | 2X2+4Y2+8М2 | 16\*y+x\*z-15\*x\*z+40 x=4, y=8, z=16 |
| 13. | 8X2+9Y2+21М2 | 9\*z+8\*x\*y+5\*x\*z+70 x=1, y=8, z=4 |

Вычислить выражение 18\*z\*x\*y-15\*x\*y\*z+30+19\*(x-y). Если задать величины x, y, z=1, то ответ? Все величины в выражении однобайтовые. Результат выдать на экран.   
  
Программа № 1 (образец)   
format PE GUI 4.0   
include 'include\WIN32AX.INC'   
  
.data   
formath db "%0lx",0 ; Задание формата вывода 16   
formatd db "%0d",0 ; Задание формата вывода 10   
result db 256 dup(?) ;В этой строке сохраняется результат   
;после его преобразования   
x db 2 ; 20h   
y db 2 ; 40h   
z db 2 ; 50h ; 18\*z\*x\*y-15\*x\*y\*z+30+19\*(x-y)= , если ;x=y=z=1   
c db 7   
.code   
start:   
xor AX,AX   
xor dX,dX   
MOV AL,18   
MUL [x]   
MUL [y]   
MUL [z]   
mov bx,ax   
MOV AL,15   
MUL [x]   
MUL [y]   
MUL [z]   
sub bx,ax   
add bx,30   
MOV CL,19   
MOV al,[x]   
sub al,[y]   
MUL cl   
add ax,bx   
invoke wsprintf,result,formatd,ax,dx ;Преобразуем число в строку   
invoke MessageBox,0,result,"перемнож,слож и вычит байт",MB\_OK ;Вывод   
  
invoke ExitProcess,0 ;Выход из программы   
.end start