МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №7
по дисциплине «Организация ЭВМ и систем»
Тема: Использование арифметических операций над целыми числами и процедур в Ассемблере

Студент гр. 9382 _____ Субботин М.О.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Научиться использовать арифметические операции над целыми числами и процедуры в Ассемблере

Основные теоретические положения.

Разработать на языке Ассемблер процессора IntelX86 две процедуры:

1) Одна - выполняет прямое преобразование целого числа, заданного в регистре AX (или в паре регистров DX:AX) в строку, представляющую его символьное изображение в заданной системе счисления (с учетом или без учета знака в зависимости от варианта задания).
2) Другая - обратное преобразование строки, представляющей символьное изображение числа в заданной системе счисления в целое число, помещаемое в регистр AX (или в пару регистров DX:AX)
   Строка должна храниться в памяти, а также выводиться на экран для индикации.
   Отрицательные числа при представлении с учетом знака должны в памяти храниться в дополнительном коде, а на экране изображаться в прямом коде с явным указанием знака или в символьном виде со знаком.

Ход выполнения:

Вариант 1.

32 бита, с учетом знака, двоичная СС.

Первая процедура – near, только через РОНы, вторая процедура – far(в данном сегменте), через РОНы и общедоступные переменные

Для начала в регистры dx:ax задается число. Процедура DEC_TO_BIN должна обработать эти регистры и вывести число в двоичной системе в символьном представлении. В процедуре DEC_TO_BIN сначала проверяется знак числа, если отрицательный, то число инвертируется и к нему прибавляется

единица. Затем ищется самый левый значащий бит, с этого бита и происходит запись числа в выходную строку. Как только регистр DX пройден, делаем проход по регистру ax. Также рассматриваются отдельно число 0, и число 80000000h (самое большое отрицательное число).

Процедура BIN_TO_DEC наоборот переводит символьное представление к представлению в регистрах. Также отдельно рассматриваем случаи для нуля и 80000000h. Во всех остальных случаях проходим по строке, считываем символы и переводим их в представление для регистров. Также отслеживаем момент, когда нужно поменять выходной регистр.

В основной процедуре вызывается сначала процедура DEC_TO_BIN, и выводится представление числа в строке, затем вызывается процедура BIN_TO_DEC, т.к. ее вручную каждый раз проверять довольно сложно, вызывается процедура DEC_TO_BIN еще раз, чтобы убедиться, что в регистрах после процедуры BIN_TO_DEC хранятся правильные значения. Если два строковых представления совпадают, значит программа отработала правильно.

Тестирование.

№	Входные данные	Выходные данные
1	DX = 0ffffh AX = 0ffddh	-100011 -100011
2	DX = 0000h AX = 0023h	+100011 +100011
3	DX = 0000h AX = 0000h	0 0
4	DX = 8000h AX = 0000h	-10000000000000000000000000000000 -10000000000000000000000000000000
5	DX = 0ffffh AX = 0ffffh	-1 -1

№	Входные данные	Выходные данные
6	DX = 0000h AX = 05AAh	+10110101010 +10110101010
7	DX = 0ffffh AX = 0ff9ch	-1100100 -1100100

Выводы.

Были изучены арифметические операции над целыми числами и процедуры в Ассемблере.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
STACKSG SEGMENT  PARA STACK 'Stack'
      DW       512 DUP(?)
STACKSG  ENDS

DATASG  SEGMENT  PARA 'Data'                    ;SEG DATA
        KEEP_CS DW 0 ;
        in_str DB 35 DUP('0')
DATASG    ENDS                                          ;ENDS DATA


CODE      SEGMENT                                ;SEG CODE
ASSUME  DS:DataSG, CS:Code, SS:STACKSG


DEC_TO_BIN PROC NEAR; DX:AX - число; в DX - сдвиг для выходной строки
        jmp start
        REGISTER_COUNTER DW 1; переменная для отслеживания
рассматриваемого регистра
start:
        mov REGISTER_COUNTER,1; записываем в REGISTER_COUNTER
единицу
        mov DI,0h ; записываем в DI, 0
        mov in_str[DI],'+' ; в строку на 0-й индекс ставим +
        cmp DX,0 ; сравниваем dx с нулем
        mov BX,8000h; записываем бит-маску 8000h в bx (это 1 и 15 нулей в
бинарной СС) т.е. битмаска на most-significant bit
        jge positive_1 ; если dx >=0 то число положительное
        ;если dx < 0
        mov in_str[DI],'-'; меняем + в 0-м индексе строки на -
        add DI,1 ; инкрементируем индекс
        ;т.к. число отрицательно, инвертируем его модуль и прибавляем единицу
        not DX;
        not AX;
        add AX,1;
        jnc no_carry; если после предыдущей операции флаг CF равен 0, т.е.
результат
        ;уместился в ax и переноса из старшего бита не было
        add DX,1; если же перенос случился, добавим в dx 1
no_carry:
        jmp scan
reg_skip:;прошли регистр
```

```
        cmp REGISTER_COUNTER,1 ; сравниваем REGISTER_COUNTER с 1
        jne ZERO_OR_OVERFLOW ; если REGISTER_COUNTER не равен 1, т.е.
мы прошлись по двум регистрам
        ;и в обоих случаях битмаска равняется нулю - тогда либо число 0, либо
введенное
        ;число слишком большое и произошло переполнение
        mov SI,AX ; записываем в si, ax
        sub REGISTER_COUNTER,1 ; вычитаем из REGISTER_COUNTER 1
        mov BX,8000h ; записываем в bx битмаску 8000h
        jmp find_leftmost;

positive_1:
        mov in_str[0],'+'
        add DI,1
scan:
        mov SI,DX ; записываем в si, dx
shift_mask: ;двигаем маску вправо
        shr BX,1 ; побитовый сдвиг вправо
find_leftmost: ;ищем
        cmp BX,0 ; сравниваем битмаску с 0
        je reg_skip ; если битмаска равна 0, то это значит, что мы прошлись по
одному регистру полностью
        mov CX,BX; записываем бит-маску в cx
        and CX,SI; в cx на месте единственного бита теперь бит соответствующий
SI
        cmp CX,0 ; сравниваем cx с нулем
        je shift_mask; если cx == 0
        ;если cx!=0 то в cx самый левый значащий бит

digit_loop:
        mov CX,BX; в cx записываем бит-маску
        and CX,SI; в единственном бите cx теперь бит из SI
        cmp CX,0; сравниваем cx и 0
        je zero_digit ; если cx == 0
        ;ненулевое число
        mov in_str[DI], '1' ; записываем в строку единицу
        inc DI; инкрементируем индекс
        jmp next_digit
zero_digit: ; если соответствующий бит равен нулю
        mov in_str[DI],'0' ; записываем в строку нуль
        inc DI; инкрементируем индекс

next_digit: ;endif
        shr BX,1 ; делаем побитовый сдвиг битмаски вправо
```

```
        cmp BX,0 ; сравниваем битмаску с нулем
        jne digit_loop ; если битмаска не равна нулю продолжаем проходиться
        ;иначе, если битмаска равна нулю, то мы прошлись по одному из
регистров
        cmp REGISTER_COUNTER,1 ;сравниваем REGISTER_COUNTER с
единицей
        jne loop_end; если REGISTER_COUNTER не равна единице, то это
значит, что мы прошлись по ax, а ax - второй регистр
        ;если REGISTER_COUNTER равен единице, то мы прошлись по первому
регистру dx, и надо еще пройти по ax
        mov SI,AX; записываем в si, ax
        sub REGISTER_COUNTER,1; декреминтируем REGISTER_COUNTER
        mov bx,8000h; записываем в bx битмаску
        jmp digit_loop ; начинаем проход
loop_end: ; когда прошли все регистры
        mov in_str[DI],'$' ; добавляем в конец строки символ конца строки
        jmp in_end
zero_or_overflow: ;если введенное число было нулем, либо слишком большим
        mov SI,0 ; записываем в si, 0
        cmp in_str[SI],'-' ;сравниваем первый элемент строки с -
        jne in_zero ; если не -
; если все же -, это значит что число задано 80000000h
        inc SI ; инкрементируем индекс
        mov in_str[SI],'1' ; записываем единицу в строку в индекс si
        mov CX,31 ; записываем в cx, 31
overflow_loop: ; ставим 31 нуль
        mov SI,CX;
        add SI,1;
        mov in_str[SI],'0'
        loop overflow_loop
        mov SI,32

in_zero:
        mov in_str[SI],'0' ; записываем в индекс SI 0
        inc SI ; инкрементируем индекс
        mov in_str[SI],'$' ; записываем во второй индекс символ конец строки

in_end:
        mov DX,offset in_str ; записываем в dx сдвиг строки

        ret
DEC_TO_BIN ENDP
;------------------------BIN_TO_DEC-------------------------------------
```

```
BIN_TO_DEC PROC FAR; in_str - строка, которую будем читать; DX:AX - здесь будет число
        jmp start_bin_to_dec
        NEG_NUMB DB 0 ; отвечает за знак числа
        REG_NUMB DB 1; отвечает за номер регистра

start_bin_to_dec:
        mov AX,0 ; обнуляем ax
        mov DX,0 ; обнуляем dx
        mov SI,0 ; за индекс строки будет отвечать si
        cmp in_str[SI],'-' ; сравниваем первый элемент строки с минусом
        jne positive; если не равен минусу, то число положительное
        ;если равен то отрицательное
        mov NEG_NUMB,1; в NEG_NUMB записываем 1

positive: ; если число положительно
        sub ax,ax; обнуляем ax
        mov SI,0 ;кладем в SI 0
        sub SI,1 ; вычитаем из si, 1

len_loop: ; считаем длину строки
        add SI,1
        cmp in_str[SI],'$' ; сравниваем элемент строки с $
        jne len_loop ; если не равен $ то возвращаемся в цикл

        mov REG_NUMB, 1
        cmp SI,21h; сравниваем равен ли индекс конечного символа 33(знак+32+$-1)
        je max_neg_numb ; если равна

        cmp SI,1; сравниваем конечный индекс с 1
        je zero_numb ; если равна

        mov DI,AX ;начинаем рассматривать регистр ax,
        mov BX,1; bx - битмаска

traverse_loop:
        dec SI ; декреминтим SI
        cmp SI,0 ; сравниваем SI с 0
        jle post_parse ; SI <= 0

        cmp in_str[SI],'1' ; сравниваем элемент в строке в индексе SI с единицей
        jne parse_zero ; если не равно единице
```

```
        or DI,BX; в di на месте в том, котором стоит единица в маске BX,
ставится 1
        jmp post_parse

parse_zero: ; если элемент не равен единице
        not BX; в bx единица стоит на текущем элементе
        ;инвертация ставит 0 на этом элементе, а во всех остальных по единице
        and DI,BX; при логическом и, в DI получаем на рассматриваемом
элементе 0
        not BX; возвращаем маску в исходное положение

post_parse:
        shl BX,1 ; сдвигаем битмаску влево
        cmp BX,0 ; сравниваем bx с нулем
        je parse_overflow ; если bx == 0
        cmp BX,8000h
        jb traverse_loop ; если bx < маски для последнего элемента

;если маска >=8000h
parse_overflow:
        cmp REG_NUMB,0;
        je end_of_second_reg; если REG_NUMB == 0
;если REG_NUMB == 1
        cmp BX,8000h
        je traverse_loop

        mov AX,DI; результат вставляем в первый регистр
        mov DI,0; очищаем регистр DI
        mov REG_NUMB,0; теперь получаем результат для регистра DX
        mov BX,1; обновляем маску
        jmp traverse_loop

end_of_second_reg:
        mov DX,DI;записываем результат во второй регистр
        jmp check_neg

max_neg_numb: ;максимально отрицательное число
        mov AX,0
        mov DX,8000h;
        jmp binary_end

zero_numb: ;случай для нуля
        mov AX,0;
        mov DX,0;
```

```
        jmp binary_end

check_neg:
        cmp NEG_NUMB,0
        je binary_end ; если NEG_NUMB == 0, т.е. число положительное
;если число отрицательное
;инвертируем и добавляем единицу
        not DX;
        not AX;
        add AX,1;
        jnc binary_end; если нет перехода
        add DX,1;

binary_end:

        ret
BIN_TO_DEC ENDP


Main     PROC  FAR
        mov  ax, DATASG
        mov  ds, ax

        mov DX,0ffffh
        mov AX,0ff9ch
        call DEC_TO_BIN

        mov ah,09h;
        int 21h;

        mov dl, 10
        mov ah, 02h
        int 21h
        mov dl, 13
        mov ah, 02h
        int 21h

        mov ax,0
        mov dx,0

        call BIN_TO_DEC;
        call DEC_TO_BIN

        mov ah,09h
```

```
        int 21h

        mov ah,4Ch;
        int 21h;


Main    ENDP
CODE    ENDS
END Main
```