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

ОТЧЕТ

по лабораторной работе №7

по дисциплине «Организация ЭВМ и систем»

Тема: Использование арифметических операций над целыми числами и процедур в Ассемблере

Студент гр. 8383 _____ Ларин А.

Преподаватель _____ Ефремов М.А,

Санкт-Петербург

2019

Цель работы.

Научится обрабатывать целые числа пир помощи арифметических операций на языке ассемблера. Научится представлять числа в строковом виде в выбранной системе счисления. Научится реализовывать функции для обработки чисел используя разные методы передачи аргументов

Задание

Разработать на языке Ассемблер процессора IntelX86 две процедуры:

- 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);
- 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.

Пример для однобайтовых чисел:

Десятичное число в символьном виде	Двоично-десятичное упакованное число
+ 35	00110101
- 35	11001011

Варианты заданий для выполнения преобразования определяются шифром, выбираемым по табл.7 и состоящим из 3-х цифр:

- 1-я цифра задает длину целого числа:

16 бит, 2- 32 бита;

- 2-я цифра задает вид представления числа:

1- с учетом знака, 2- без учета знака;

- 3-я цифра задает систему счисления для символьного изображения числа:

1- двоичная, 2- восьмеричная, 3-десятичная, 4- шестнадцатиричная.

Таблица 7

№ бригады	Шифр Задания	№ бригады	Шифр Задания
1	2.1.1	7	1.1.4
2	2.2.1	8	2.2.4
3	1.2.3	9	2.1.3
4	1.1.3	10	2.2.3
5	2.2.2	11	2.1.4
6	1.1.2	12	1.1.1

Написать простейшую головную программу для иллюстрации корректности выполнения заданных преобразований. При этом вызываемые процедуры могут быть одного из следующих типов:

1 - near, 2 – far (в данном сегменте), 3 – far (в другом сегменте).

Связь по данным между основной программой и подпрограммами может осуществляться следующими способами:

A - только через РОНы;

B - через РОНы и общедоступные переменные;

C - через кадр стека.

Шифры, определяющие типы процедур и способы связи по данным, приведены в табл.8.

Таблица 8

№ бригады	Шифр Задания	№ бригады	Шифр Задания
1	1A2B	7	2C1A
2	1B3C	8	2A3B
3	1C2B	9	1A3B
4	2A1B	10	1C2A
5	2B2A	11	1A3C
6	2B3C	12	2C3B

Выполнение

Вариант 1 (13)

32 бита, с учетом знака, двоичная система

1-я процедура — near, аргументы через РОНы,

2-я процедура — far, один сегмент, РОНы и общие переменные.

Программа содержит две процедуры — dtob для перевода числа в строку в двоичне. системе счисления и btod для обратного преобразования.

DTOB PROC NEAR; DX:AX - Number; DX - Final binary string

Переводит 32-битное число записанное в регистрах DX:AX в строку в двоичном виде с явным указанием знака в прямом коде и возвращает ее смещение через DX.

В начале проверяется знак числа. Если число отрицательное, то оно превращается в положительное, а первым символом в строку записывает знак '-'. В противном случае записывается '+'. Задается двоичная маска. В SI устанавливается активный регистр — DX. Далее в цикле слева пропускаются все незначимые нулевые разряды, маска сдвигается вправо. При пропуске всего регистра активным регистром задается AX(т. е. В SI помещается значение AX). Когда первый значимый разряд найден происходит обход числа в цикле, каждый разряд извлекается при помощи битовой маски и записывается в строку. Если один регистр пройден процесс повторяется со вторым. В регистр DX записывается смещение результирующей строки, выполняется возврат.

BTOD PROC FAR;TPL - string to process; DX:AX - result number

Переводит строку из сегмента данных с меткой TPL содержащую двоичные число в прямом коде с явным указанием знака в 32-битное целое число, записанное в регистры DX:AX.

В начале по первому символу определяется является ли число отрицательным, для этого в функции есть флаг NEGT DB. Далее считается длинна строки, индекс SI устанавливается в ее конец и начинается ее обход от младших разрядов к старшим. Каждый символ сравнивается с символом '1', '0'. соответствующий разряд устанавливается в соответсвующий регистр при помощи битовой маски. Процедура повторяется для каждым регистром. В результате в регистрах DX:AX хранится результирующее число. Происходит возврат управления в вызвавшую программу.

Написана головная программа для тестирования написанных функций. В ней заданные регистру переводятся в строки первой функцией, затем результирующей функцией строки переводятся обратно в числа, что позволяет оценить правильность работы функций.

Тестирование.

DX:AX	DX = TPL = DTOB(DX:AX)	DX:AX = BTOD(TPL)
51080821h	-10110111101111111011111011111	51080821
D1080821h	+10100010000100000001 00000100001	D1080821h
00000020h	+100000	00000020h
00000000h	0	00000020h
80000000h	-10000000000000000000000000000000	80000000h

Выводы.

В результате работы были разобраны некоторые базовые концепции языка ассемблера. Были изучены методы работы с целочисленными

данными. Была написана рабочая программа для перевода чисел в строковое представление.

ПРИЛОЖЕНИЕ

LR7.ASM

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

DATASG  SEGMENT  PARA 'Data'       ;SEG DATA
 KEEP_CS DW 0 ;
 TPL DB '-0123456789abcdef0123456789abcdefG$'
DATASG  ENDS               ;ENDS DATA


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


DTOB PROC NEAR; DX:AX - Number; DX - Final binary string
 jmp dtob_start
 BASE DW 1;Keep track of registers order
dtob_start:
 mov BASE,1;
 mov DI,0h
 mov TPL[DI],'+'
 cmp DX,0
 mov BX,8000h;bit mask
 jge posit
;negative
 mov TPL[DI],'-'
 add DI,1
 ;neg
 not DX;
 not AX;
 add AX,1;
 jnc no_carry
 add DX,1;
no_carry:
 jmp scan
reg_skip:;skept register
 cmp BASE,1
 jne ZERO_OR_OVERFLOW
 mov SI,AX
 sub BASE,1
 mov BX,8000h
 jmp kostyl;Like not_yet but no initial bit skip
;positive
posit:
```

```
 mov TPL[0],'+'
 add DI,1
 ;DX
 ;WHILE FIRST NON-ZERO DIGIT NOT FOUND
scan:
 mov SI,DX
not_yet:
 shr BX,1
kostyl:;Like not_yet but no initial bit skip
 cmp BX,0
 je reg_skip
 mov CX,BX;CX <- mask
 and CX,SI;CX <- digit of SI
 cmp CX,0
 je not_yet;LOOP DIGIT SKIP
 ;CX - first left nonzero digit
digit_loop:;SI - REGISTER TO PROCESS
 mov CX,BX; CX <- mask
 and CX,SI; CX <- digit of SI
 cmp CX,0; if zero digit
 je zero_digit
;non-zero digit
 mov TPL[DI],'1'
 inc DI;
 jmp digit_fi
zero_digit:
 mov TPL[DI],'0'
 inc DI;
 jmp digit_fi
digit_fi:;endif
 shr BX,1
 cmp BX,0
 jne digit_loop
 ;some register processed
 cmp BASE,1
 jne loop_end;AX register processed, we're done
 mov SI,AX;AX not processed
 sub BASE,1
 mov bx,8000h;
 jmp digit_loop
loop_end:
 mov TPL[DI],'$'
 jmp dtob_end
ZERO_OR_OVERFLOW:
 mov SI,0
 cmp TPL[SI],'-'
 jne EZERO
;OVERFLOW
```

```
 inc SI
 mov TPL[SI],'1'
 mov CX,32
OF_LOOP:
 mov SI,CX;
 add SI,1;
 mov TPL[SI],'0'
 loop OF_LOOP
 mov SI,33
 mov TPL[SI],'$'
EZERO:
 mov TPL[SI],'0'
 inc SI
 mov TPL[SI],'$'

dtob_end:
 mov DX,offset TPL

 ret
DTOB ENDP

BTOD PROC FAR;TPL - string to process; DX:AX - result number
 jmp btod_start
 NEGT DB 0
 REGN DB 1; REg number: 1, then 0

btod_start:
 mov AX,0
 mov DX,0
 mov SI,0;Si-index in str
 cmp TPL[SI],'-'
 jne positive;
;negative
 mov NEGT,1
 inc SI;
positive:
 ;cld
 sub ax,ax;
 mov SI,0
 sub SI,1
len_loop:
 add SI,1
 cmp TPL[SI],'$'
 jne len_loop

;start parsing
 mov REGN,1
 cmp SI,22h;Max len => -(2^32)
```

```
 je max_d
 cmp SI,1;Min len
 je zero_d
 mov DI,AX ;DI - active register
 mov BX,1;BX - bit mask

parse_loop:
 dec SI
 cmp SI,0
 jle parse_fi

parse_cont:
;parse_if
 cmp TPL[SI],'1'
 jne parse_zero
;parse_one
 or DI,BX;set one by mask BX
 jmp parse_fi
parse_zero:
 not BX;set zero by inverted mask BX
 and DI,BX;
 not BX;
parse_fi:
 shl BX,1
 cmp BX,0
 je parse_overflow
 ;;BX == 0 - overflow
 cmp BX,8000h
 jb parse_loop
;BX 8000 or more (or overflow)
parse_overflow:

 cmp REGN,0;
 je parse_pool;;EXIT CONDITION HERE
;regn 1
;check 8000h as well
 cmp BX,8000h
 je parse_loop

 mov AX,DI;Set processed register
 mov DI,0;Reset active register
 mov REGN,0;Set 0 register number(DX)
 mov BX,1;Reset bit mask
 jmp parse_loop
parse_pool:
 mov DX,DI;Set processed register
 jmp btod_check_neg
```

```
      max_d:;max negative number
        mov AX,0
        mov DX,8000h;
        jmp btod_end
      zero_d:;Zero number
        mov AX,0;
        mov DX,0;
        jmp btod_end

      btod_check_neg:
        cmp NEGT,0
        je btod_end
      ;btod_neg
        not DX;
        not AX;
        add AX,1;
        jnc btod_end;no carry
        add DX,1;
      ;no carry
      ;btod_pos
      btod_end:

        ret
      BTOD ENDP


      Main      PROC  FAR
          mov  ax, DATASG                        ;ds setup
          mov  ds, ax
      ;51080821_16 1359480865_10 10100010000100000001000001000001_2
      ;D1080821_16                                      3506964513_10
110100010000100000001000001000001_2                                  -
10111011110111111101111101111
```

```
        mov DX,0000h
        mov AX,0821h
        call DTOB

        mov ah,09h;
        int 21h;

        mov ax,0
        mov dx,0
        call BTOD;

        ;---

        mov DX,8000h
```

```
 mov AX,0000h
 call DTOB

 mov ah,09h;
 int 21h;

 mov ax,0
 mov dx,0
 call BTOD;

 ;---

 mov DX,0000h
 mov AX,0020h
 call DTOB

 mov ah,09h;
 int 21h;

 mov ax,0
 mov dx,0
 call BTOD;

 ;---
 mov ah,4Ch;
 int 21h;



Main      ENDP
CODE      ENDS
END Main                    ;ENDS CODE
```