**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 АХ ( или в паре регистров DХ:АX) 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 АХ ( или в пару регистров DХ:АX)

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:

А - 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 | 1А2В | 7 | 2С1А |
| 2 | 1В3С | 8 | 2А3В |
| 3 | 1С2В | 9 | 1А3В |
| 4 | 2А1В | 10 | 1С2А |
| 5 | 2В2А | 11 | 1А3С |
| 6 | 2В3С | 12 | 2С3В |

**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 двоичнe. 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 | -101110111101111111011111011111 | 51080821h |
| D1080821h | +1010001000010000000100000100001 | D1080821h |
| 00000020h | +100000 | 00000020h |
| 00000000h | 0 | 00000000h |
| 80000000h | -100000000000000000000000000000000 | 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 1010001000010000000100000100001\_2

;D1080821\_16 3506964513\_10 11010001000010000000100000100001\_2 -101110111101111111011111011111

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