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: Преобразование целых чисел. Использование процедур в Ассемблере.

Студент гр. 9382 _____ Михайлов Д.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы:

Создание ассемблерных процедур, использующих арифметические операции и различные способы передачи параметров.

Формулировка задания:

Разработать на языке Ассемблер IBM PC две процедуры: прямого и обратного преобразования целого числа, заданного в регистре AX в строку, представляющую его символьное изображение в заданной системе счисления (с учетом или без учета знака в зависимости от варианта задания). Строка должна храниться в памяти, а также выводиться на экран для индикации.

1. 1-я цифра задает длину целого числа - 16 бит;

2. 2-я цифра задает вид представления числа - без учета знака;

3. 3-я цифра задает систему счисления для символьного изображения числа - шестнадцатеричная.

4. Написать простейшую головную программу для иллюстрации корректности выполнения заданных преобразований. При этом вызываемые процедуры должны быть типа – near.

Ход работы:

1. Передача данных через общую область памяти выглядит следующим образом:

- в вызывающей процедуре: PUBLIC <имена переменных>;
- в вызываемой процедуре: EXTRN <имя параметра: тип>.

2. Но также необходимо, чтобы процедура была типа near, т. е. должна находится в одном сегменте с вызывающей процедуре. Для этого процедуру поместим в отдельный модуль, в котором она будет находится в сегменте с таким же именем и атрибутом **PUBLIC**, как и в вызывающей процедуре. В результате компоновщик объединит их в один сегмент. В этом можно будет убедиться, посмотрев карту памяти после компоновки.

В результате во втором модуле можно напрямую обращаться к переменным PUBLIC из первого модуля.

3. Вторая процедура должна помещаться в отдельном сегменте. Данный сегмент с процедурой помещается также во второй модуль.

Параметры в процедуру передаются через кадр стека. В вызывающей процедуре добавляется код:

push DX– передача смещения строки в процедуру.

call FAR PTR STR_TO_INT– вызов процедуры, в стек помещается адрес след. команды**.**

add SP, 2 –освобождение стека от передаваемых параметров.

Внутри процедуры **STR_TO_INT** также добавляется код:

push BP - сохранение текущего регистра BP.

mov BP, SP– BP настраивается на текущий указ. стека.

mov SI, [BP+6] – Извлечение переменной со смещением строки. На вершине стека находится старый BP, а также адрес возврата из данной процедуры, т.е. 2 слова. В итоге 3 слова.

В конце процедуры добавляется код:

mov SP, BP– восстанавливается указатель на стек.

pop BP– восстановление BP

Модуль 1 и 2 приведены в приложении.

4. С помощью следующих команд транслируются модули:

>MASM 1.ASM

>MASM 2.ASM

При этом создаются файлы листинга, представленные в приложении.

5. Далее производится компоновка с созданием карты памяти с помощью команды:

>LINK 1.OBJ+2.OBJ

6. Тестирование на различных входных данных:

Входное число	Строка
48671	BE1F
65520	FFF0
4671	123F
57005	DEAD

Вывод :

В результате выполнения данной лабораторной работы были получены навыки написания процедур с различными способами передачи данных, в которых использовались различные арифметические операции. Данные процедуры были помещены в отдельный модуль, из которого они вызывались.

ПРИЛОЖЕНИЕ

mod1.asm:

```
page 120, 200; Main module
EXTRN INT_TO_STR:NEAR; Defined in mod2.asm
EXTRN STR_TO_INT:FAR; Defined in mod2.asm
PUBLIC OUT_STR
; uint 16 bit HEX
STA_CK SEGMENT STACK
     DW 20 DUP(0)
STA_CK ENDS

DATA SEGMENT
     INP_INT DW 48671; Int32
     OUT_STR DB 4 DUP(0), '$'; String (length=32)
DATA ENDS

CODE SEGMENT  PUBLIC
     ASSUME CS:CODE, DS:DATA, SS:STA_CK

MAIN PROC FAR
     PUSH DS; PSP
     XOR AX,AX
     PUSH AX

     MOV AX, DATA ; Load DS
     MOV DS, AX
     MOV ES, AX ; For STOSB and LODSB.

     MOV AX, WORD PTR INP_INT; low part 32 number
     CALL INT_TO_STR; Int to string

     LEA DX, OUT_STR ; Output string
     MOV AH, 09H ; Stdout
     INT 21H ; Call

     PUSH DX; String to stack
     CALL FAR PTR STR_TO_INT; String to int
     ADD SP, 2; Clear temp

     RET
MAIN ENDP

CODE ENDS

     END MAIN
```

Mod2.asm:

```
page 120, 200; Logic module
ACHAR EQU 41h
EXTRN OUT_STR:BYTE; Defined in mod1.asm
PUBLIC INT_TO_STR
PUBLIC STR_TO_INT

CODE SEGMENT PUBLIC
     ASSUME CS:CODE

INT_TO_STR PROC NEAR
     MOV BX, AX; Save input
     LEA DI, OUT_STR; Load String adress
     MOV CX, 4; Counter
TO_STR:
     MOV AL, BH
     SHR AL, 1; Shift 4 bits right
     SHR AL, 1
     SHR AL, 1
     SHR AL, 1
     AND AL, 15
     ADD AL, 30h; '0' char
     CMP AL, 3Ah; '9' char
          JB TEN_CHAR_LOWER
     ADD AL, 7h; '0' to 'A' char
TEN_CHAR_LOWER:
     STOSB; Write AL to ES:DI then DI++
     SHL BX, 1; Shift 4 bits left
     SHL BX, 1
     SHL BX, 1
     SHL BX, 1
     LOOP TO_STR
     RET
INT_TO_STR ENDP
CODE ENDS

CODE_2 SEGMENT PUBLIC
     ASSUME CS:CODE_2

STR_TO_INT PROC FAR; Returns result to DX:AX
     PUSH BP; Save BP
     MOV BP, SP
     MOV SI, [BP+6]; Get string adress (0 - old BP; 2,4 - CALL
return)
```

```
     MOV CX, 4; Counter
     MOV BX, 0; Result
TO_INT:
     LODSB; AL char from ES:SI
     SUB AL, 30h; '0' char
     CMP AL, 10h; ':' (above '9')
          JB TEN_LOWER
     SUB AL, 7h; '0' to 'A' char
TEN_LOWER:
     SHL BX, 1; Shift 4 bits left
     SHL BX, 1
     SHL BX, 1
     SHL BX, 1
     ADD BL, AL
     LOOP TO_INT
     MOV AX, BX; Result
     MOV SP, BP; Resore stack
     POP BP; Restore BP
     RET
STR_TO_INT ENDP

CODE_2 ENDS

     END
```