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.

Студентка гр. 9382 _____ Круглова В. Д.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Научиться использовать арифметические операции над целыми числами и процедуры в Ассемблере.

Задание.

Вариант 1.1.1

16 бит, с учетом знака, двоичная СС

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.

Ход работы:

В регистр ax задается число, которое необходимо перевести в строку. Процедура TEN_TO_TWO отвечает за перевод регистра в строку. Сначала проверяет знак числа. Если число отрицательное, то оно переводится в прямой код, иначе оно не нуждается в переводе. Затем происходит поиск значащего бита в числе. Если он не найден, то число либо нулевое, либо это самое большое отрицательное число. Если был найден значащий бит, то формируется строка. Биты рассматриваются с помощью битмаски.

Процедура TWO_TO_TEN переводит строку к регистру. Рассмотрены отдельно случаи 0 и самое большое отрицательное число. В других случаях проходим по строке и формируем число с помощью битмаски.

В основной процедуре вызывается сначала TEN_TO_TWO, и выводится представление числа в строке, затем вызывается процедура TWO_TO_TEN, так как вручную ее проверять довольно сложно, вызывается процедура TEN_TO_TWO еще раз, чтобы убедиться, что в регистре после процедуры TWO_TO_TEN хранится правильное значение. Если два строковых представления совпадают, значит программа отработала правильно.

Тестирование.

Входные данные	Выходные данные
ax=F100h	From register to string: -111100000000 From string to register and register for visual to from string: -111100000000
Ax=8000h	From register to string: -1000000000000000 From string to register and register for visual to from string: -1000000000000000
Ax=0h	From register to string: 0

Исходный код программы.

XDD.asm:

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

DATASG  SEGMENT  PARA 'Data'
     KEEP_CS DW 0 ;
     MES1 DB 'From register to string: $'
     MES2 DB 'From string to register and for visual from
register to string: $'
     STRING DB 35 DUP('0')
```

```
DATASG    ENDS


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


TEN_TO_TWO PROC NEAR

; ax – 16 бит, записано в доп. коде (первый бит – знак, далее число)
; di – индекс текущего символа строки
; bx – маска для числа, с ее помощью достаем цифру, первоначальное значение – 8000h (16 бит – 1)
; Отрицательное число записано в доп. коде, в этом случае необходимо инвертировать число и прибавить 1

START:
     SUB di, di                    ; Обнуляем di
     MOV bx, 8000h                 ; Кладем маску в bx
     CMP ax, 0                     ; Сравниваем число с 0
     JGE POSITIVE_NUMBER           ; Если ax >= 0, то число положительное
     MOV STRING[di], '-'           ; Если ax < 0, то число отрицательное
     INC di
     NOT ax
     INC ax                        ; Перевод доп. кода в прямой код
     JMP FIND_SIGNIFICANT

POSITIVE_NUMBER:
     MOV STRING[di], '+'
     INC di
     JMP FIND_SIGNIFICANT

END_REGISTER:
     SUB si, si
     CMP STRING[si], '-'
     JNE NULL_REGISTER
```

```
        MOV cx, 15
        INC si
        MOV STRING[si], '1'

    OVERFLOW_REGISTER:
        MOV si, cx
        INC si
        MOV STRING[si], '0'
        LOOP OVERFLOW_REGISTER
        MOV si, 16

    NULL_REGISTER:
        MOV STRING[si], '0'
        INC si
        MOV STRING[si], '$'
        JMP END_CONVERT

    FIND_SIGNIFICANT:
        SHR bx, 1                   ; Сдвигаем маску вправо
        CMP bx, 0                   ; Сравниваем маску с 0
        JE END_REGISTER             ; Если она 0, то мы полностью
прошли регистр
                                        ; (В этом случае либо
число 0, либо слишком большое)
        MOV cx, bx                  ; Иначе выдергиваем бит
        AND cx, ax
        CMP cx, 0                   ; И сравниваем его с 0
        JE FIND_SIGNIFICANT           ; Если он все еще 0, то
продолжаем искать значащий бит дальше

    MATCHED_DIGIT:
        CMP cx, 0                     ; Смотрим, что находится в
бите
        JE ZERO_DIGIT                 ; Если 0, то записываем 0 в
строку
        MOV STRING[di], '1'         ; Иначе записывает в строку 1
        INC di
        JMP NEXT_DIGIT
```

```
    ZERO_DIGIT:                          ; Если соответствующий бит
равен нулю
         MOV STRING[di], '0'
         INC di

    NEXT_DIGIT:
         SHR bx, 1                      ; Сдвигаем маску вправо
         CMP bx, 0                      ; Сравниваем маску с 0
         JE LOOP_END                     ; Если маска 0, то мы прошли
полностью регистр
         MOV cx, bx
         AND cx, ax                     ; Иначе выдергиваем бит
         JMP MATCHED_DIGIT

    LOOP_END:                            ; Регистр пройден до конца
         MOV STRING[di], '$'            ; добавляем в конец строки
символ конца строки

    END_CONVERT:
         MOV dx, offset STRING          ; Записываем в dx сдвиг до
строки
         ret

    TEN_TO_TWO ENDP


    TWO_TO_TEN PROC FAR
         JMP START_B2D
         SIGN_NUMBER DB 0; Отвечает за знак числа

    ; STRING – Строка, которую будем переводить в число
    ; ax – Результативный регистр

    START_B2D:
         SUB ax, ax                     ; Обнуляем ax
         SUB si, si                           ; Обнуляем si (индекс
текущего элемента строки)
         CMP STRING[si], '-'            ; Сравниваем первый элемент
строки с минусом
```

```
        JNE LENGTH_LOOP                ; Если не равен минусу, то
число положительное
        MOV SIGN_NUMBER, 1             ; Если равен, то число
отрицательное

    LENGTH_LOOP:                       ; Подсчет длины строки
        INC si
        CMP STRING[si], '$'            ; Сравниваем элемент строки с
$
        JNE LENGTH_LOOP                ; Если не равен $ то
возвращаемся в цикл
        CMP si, 11h                    ; Сравниваем длину строки с
17
                                       ; Если длина строки 17 (знак
+ 16), то это последнее отрицательное число
        JGE MAX_NEGATIVE_NUMBER
        CMP si, 1                      ; Если длина строки 1, то это
ноль
        JE BINARY_END
        SUB di, di                     ; Начинаем рассматривать
регистр ax
        MOV bx, 1                      ; Битмаска

    TRAVERSE_LOOP:
        DEC si
        CMP si, 0                      ; Сравниваем si с 0
        JE CHECK_NEGATIVE              ; Все символы пройдены,
завершение
        CMP STRING[si], '1'            ; Сравниваем элемент в строке
в индексе si с единицей
        JNE FORM_BITMASK               ; Если не равно единице
        OR ax, bx                      ; В ax на месте в том,
котором стоит единица в маске BX, ставится 1

    FORM_BITMASK:
        SHL bx, 1                      ; Сдвигаем битмаску влево
        JMP TRAVERSE_LOOP

    MAX_NEGATIVE_NUMBER:
        MOV ax, 8000h
```

```
          JMP BINARY_END

     CHECK_NEGATIVE:
          CMP SIGN_NUMBER, 0
          JE BINARY_END                    ; Если число положительное,
то завершение программы
          NOT ax;                          ; Если число отрицательное,
записываем в доп. коде
          INC ax;                            ; Инвертируем и добавляем
единицу

     BINARY_END:
         ret

     TWO_TO_TEN ENDP


     Main PROC FAR
          MOV ax, DATASG
          MOV ds, ax

         MOV dx, offset MES1
         MOV ah, 09h
          INT 21h

          MOV ax, 1h
          CALL TEN_TO_TWO
          MOV ah, 09h;
          INT 21h;

          MOV dl, 10
          MOV ah, 02h
          INT 21h
          MOV dl, 13
          MOV ah, 02h
          INT 21h

         MOV dx, offset MES2
         MOV ah, 09h;
          INT 21h;
```

```
        CALL TWO_TO_TEN
        CALL TEN_TO_TWO

        MOV ah, 09h
        INT 21h

        MOV ah, 4Ch
        INT 21h
Main      ENDP
CODE      ENDS
END Main
```

Выводы.

В ходе выполнения работы были использованы арифметические операции над целыми числами и процедуры в ассемблере, разработаны 2 процедуры по преобразованию числа в строку в двоичном виде и обратно.