**Цель работы**

Изучить и приобрести практические навыки работы с основными командами языка Ассемблера, функциями ввода-вывода, регистрами и символьными данными.

**Постановка задачи**

По предложенному преподавателем варианту разработать программу на языке Ассемблера, решающую поставленные задачи:

1. ввод с клавиатуры числа в заданной системе счисления (**Шестнадцатеричная** система счисления);
2. вывод введенного числа в десятичной системе счисления;
3. вычисление значения полинома вида **4x^2 + 19x - 2** (в предположении, что результат вычислений не приводит к переполнению регистров);
4. вывод результата в заданной системе счисления (**Восьмеричная** система счисления);
5. вывод результата в десятичной системе счисления.

**Алгоритм**

**Этап 1: Инициализация Среды**

1. **Настройка кодировки консоли:**  
   1.1. Вызвать функцию WinAPI SetConsoleCP с аргументом 1251, чтобы консоль корректно воспринимала ввод кириллических символов в кодировке Windows-1251.  
   1.2. Вызвать функцию WinAPI SetConsoleOutputCP с аргументом 1251, чтобы консоль корректно отображала вывод кириллических символов в кодировке Windows-1251.
2. **Получение дескрипторов (handles) консоли:**  
   2.1. Вызвать функцию WinAPI GetStdHandle для получения дескриптора стандартного потока вывода (STDOUT). Сохранить дескриптор в глобальную переменную.  
   2.2. Вызвать функцию WinAPI GetStdHandle для получения дескриптора стандартного потока ввода (STDIN). Сохранить дескриптор в глобальную переменную.

**Этап 2: Ввод и Преобразование Числа X**

1. **Приглашение к вводу:**  
   1.1. Вывести на консоль строку-приглашение: "Введите шестнадцатеричное число x: ".
2. **Чтение строки с клавиатуры:**  
   2.1. Вызвать функцию WinAPI ReadConsoleA, передав ей адрес буфера input\_buffer для сохранения введенной строки.  
   2.2. Прочитать из памяти 32-битное значение, возвращенное ReadConsoleA (количество прочитанных символов).  
   2.3. Поставить в конец введенной строки нуль-терминатор (\0), чтобы создать корректную C-строку.  
   2.4. Удалить из конца строки управляющие символы перевода каретки и новой строки (\r\n), заменив их на нуль-терминатор.
3. **Преобразование строки в число:**  
   3.1. Вызвать пользовательскую процедуру HexStrToInt64 для преобразования 16-ричной строки из input\_buffer в 64-битное знаковое целое число.  
   \* **Алгоритм HexStrToInt64:**  
   1. Обнулить 64-битный регистр-аккумулятор (RAX).  
   2. Проверить первый символ строки. Если это '-', установить флаг отрицательного числа и сдвинуть указатель на следующий символ.  
   3. В цикле обрабатывать каждый символ строки до нуль-терминатора:  
   4.1. Преобразовать ASCII-символ ('0'-'9', 'a'-'f', 'A'-'F') в его числовое значение (0-15).  
   4.2. Сдвинуть аккумулятор влево на 4 бита (эквивалентно умножению на 16).  
   4.3. Прибавить к аккумулятору полученное числовое значение.  
   5. После цикла, если был установлен флаг отрицательного числа, инвертировать знак аккумулятора (neg rax).  
   3.2. Сохранить полученное число в глобальную переменную input\_x.

**Этап 3: Вычисление Полинома Y**

1. **Расчет y = 4x² + 19x - 2:**  
   1.1. Загрузить значение x из переменной input\_x в регистр RAX.  
   1.2. Вычислить x², выполнив знаковое умножение imul rax, rax.  
   1.3. Умножить результат на коэффициент a (4): imul rax, [coeff\_a]. В RAX теперь находится 4x².  
   1.4. Загрузить значение x в регистр RBX.  
   1.5. Умножить x на коэффициент b (19): imul rbx, [coeff\_b]. В RBX теперь находится 19x.  
   1.6. Сложить два полученных значения: add rax, rbx. В RAX теперь 4x² + 19x.  
   1.7. Прибавить (вычесть) коэффициент c (-2): add rax, [coeff\_c]. В RAX теперь финальное значение y.  
   1.8. Сохранить результат из RAX в глобальную переменную result\_y.

**Этап 4: Преобразование и Вывод Результатов**

1. **Вывод x в десятичной системе:**  
   1.1. Вывести на консоль поясняющую строку.  
   1.2. Вызвать пользовательскую процедуру Int64ToStr для преобразования числа input\_x в строку с основанием 10.  
   \* **Алгоритм Int64ToStr:**  
   1. Проверить знак числа. Если отрицательное, записать в буфер символ '-' и инвертировать знак числа.  
   2. В цикле, пока число не равно нулю:  
   3.1. Выполнить операцию деления числа на основание (10 или 8).  
   3.2. Остаток от деления — это следующая цифра. Преобразовать ее в ASCII-символ.  
   3.3. Поместить полученный символ в стек.  
   3.4. Заменить число результатом целочисленного деления.  
   4. После цикла извлекать символы из стека и записывать их в буфер (таким образом они оказываются в правильном порядке).  
   5. Поставить в конец строки в буфере нуль-терминатор.  
   6. Вывести полученную строку на консоль.
2. **Вывод y в восьмеричной системе:**  
   2.1. Вывести на консоль поясняющую строку.  
   2.2. Вызвать процедуру Int64ToStr для преобразования числа result\_y в строку с основанием 8.  
   2.3. Вывести полученную строку на консоль.
3. **Вывод y в десятичной системе:**  
   3.1. Вывести на консоль поясняющую строку.  
   3.2. Вызвать процедуру Int64ToStr для преобразования числа result\_y в строку с основанием 10.  
   3.3. Вывести полученную строку на консоль.

**Этап 5: Завершение Работы**

1. **Корректный выход:**  
   1.1. Вывести на консоль символ новой строки для форматирования.  
   1.2. Вызвать функцию WinAPI ExitProcess с кодом завершения 0, что сигнализирует об успешном выполнении программы.

**Код программы (main.asm)**

; ------------------------------------------------------------------------------

; Объявление внешних функций из библиотеки Kernel32.dll, которые мы будем использовать.

; ------------------------------------------------------------------------------

EXTERN GetStdHandle: PROC

EXTERN WriteConsoleA: PROC

EXTERN ReadConsoleA: PROC

EXTERN ExitProcess: PROC

EXTERN SetConsoleCP: PROC

EXTERN SetConsoleOutputCP: PROC

; ==============================================================================

; Секция .DATA: здесь объявляются все инициализированные и неинициализированные

; глобальные переменные и константы.

; ==============================================================================

.DATA

; --- Дескрипторы стандартных потоков ---

std\_input\_handle QWORD ? ; Переменная для хранения дескриптора консоли для ввода.

std\_output\_handle QWORD ? ; Переменная для хранения дескриптора консоли для вывода.

; --- Коэффициенты полинома ---

coeff\_a DQ 4

coeff\_b DQ 19

coeff\_c DQ -2

; --- Переменные для хранения чисел ---

input\_x DQ ?

result\_y DQ ?

; --- Строки для взаимодействия с пользователем ---

msg\_prompt\_x DB "Введите шестнадцатеричное число x: ", 0

msg\_x\_dec DB 13, 10, "Введенное число x в десятичной системе: ", 0

msg\_y\_oct DB 13, 10, "Результат y в восьмеричной системе: ", 0

msg\_y\_dec DB 13, 10, "Результат y в десятичной системе: ", 0

msg\_newline DB 13, 10, 0 ; Строка для перевода курсора. 13=CR, 10=LF.

; --- Буферы и вспомогательные переменные ---

BUFFER\_SIZE EQU 256

input\_buffer DB BUFFER\_SIZE DUP(?) ; Буфер для чтения строки с клавиатуры.

output\_buffer DB BUFFER\_SIZE DUP(?) ; Буфер для преобразования чисел в строку для вывода.

; --- Вспомогательные переменные для чтения строки

chars\_read DD ? ; DWORD (32 бита) в соответствии с API

chars\_written DD ? ; DWORD (32 бита) в соответствии с API

; ==============================================================================

; Секция .CODE: здесь находится исполняемый код программы.

; ==============================================================================

.CODE

; ------------------------------------------------------------------------------

; Главная процедура программы (точка входа).

; ------------------------------------------------------------------------------

main PROC

; Пролог: выделяем 40 байт на стеке. 32 байта - это "теневое пространство",

; обязательное по соглашению о вызовах x64, плюс 8 для выравнивания.

sub rsp, 40

; --- ЭТАП 1: ИНИЦИАЛИЗАЦИЯ СРЕДЫ ---

; Устанавливаем кодовую страницу 1251 для корректной работы с кириллицей.

mov rcx, 1251 ; Аргумент 1: ID кодовой страницы (1251 для Windows Cyrillic)

call SetConsoleCP ; Устанавливаем кодировку для ввода.

mov rcx, 1251

call SetConsoleOutputCP ; Устанавливаем кодировку для вывода.

; Получаем дескриптор стандартного потока вывода (экран).

mov rcx, -11 ; Аргумент 1: -11 это константа STD\_OUTPUT\_HANDLE.

call GetStdHandle

mov [std\_output\_handle], rax ; Сохраняем полученный дескриптор.

; Получаем дескриптор стандартного потока ввода (клавиатура).

mov rcx, -10 ; Аргумент 1: -10 это константа STD\_INPUT\_HANDLE.

call GetStdHandle

mov [std\_input\_handle], rax ; Сохраняем дескриптор.

; --- ЭТАП 2: ВВОД И ПРЕОБРАЗОВАНИЕ ЧИСЛА X ---

; Выводим приглашение к вводу.

lea rdx, msg\_prompt\_x ; Аргумент 1 (RDX): адрес строки для вывода.

call PrintString

; Читаем строку с клавиатуры.

lea rdx, input\_buffer ; Аргумент 1 (RDX): адрес буфера для сохранения строки.

call ReadString

; Преобразуем 16-ричную строку в 64-битное число.

lea rcx, input\_buffer ; Аргумент 1 (RCX): адрес строки для преобразования.

call HexStrToInt64

mov [input\_x], rax ; Сохраняем результат (из RAX) в переменную.

; --- ЭТАП 3: ВЫВОД РЕЗУЛЬТАТОВ ---

lea rdx, msg\_x\_dec ; Выводим X в десятичной системе.

call PrintString

mov rcx, [input\_x] ; Аргумент 1: число для преобразования.

lea rdx, output\_buffer ; Аргумент 2: буфер для результата.

mov r8, 10 ; Аргумент 3: основание системы счисления (10).

call Int64ToStr

lea rdx, output\_buffer ; Загружаем адрес полученной строки в RDX.

call PrintString ; Печатаем.

; --- ЭТАП 4: ВЫЧИСЛЕНИЕ ПОЛИНОМА Y ---

mov rax, [input\_x] ; rax = x

mov rbx, [input\_x] ; rbx = x

imul rax, rbx ; rax = x \* x (x^2)

imul rax, [coeff\_a] ; rax = rax \* a (a\*x^2)

mov rbx, [input\_x] ; rbx = x

imul rbx, [coeff\_b] ; rbx = rbx \* b (b\*x)

add rax, rbx ; rax = a\*x^2 + b\*x

add rax, [coeff\_c] ; rax = a\*x^2 + b\*x + c

mov [result\_y], rax ; Сохраняем финальный результат в переменную Y.

; Выводим Y в восьмеричной системе.

lea rdx, msg\_y\_oct

call PrintString

mov rcx, [result\_y] ; Аргумент 1: число Y.

lea rdx, output\_buffer ; Аргумент 2: буфер для результата.

mov r8, 8 ; Аргумент 3: основание системы счисления (8).

call Int64ToStr

lea rdx, output\_buffer

call PrintString

; Выводим Y в десятичной системе.

lea rdx, msg\_y\_dec

call PrintString

mov rcx, [result\_y] ; Аргумент 1: число Y.

lea rdx, output\_buffer ; Аргумент 2: буфер для результата.

mov r8, 10 ; Аргумент 3: основание (10).

call Int64ToStr

lea rdx, output\_buffer

call PrintString

; --- ЭТАП 5: ЗАВЕРШЕНИЕ РАБОТЫ ---

lea rdx, msg\_newline

call PrintString

; Корректно завершаем процесс.

mov rcx, 0 ; Аргумент 1: код возврата (0 = успех).

call ExitProcess

main ENDP

; ------------------------------------------------------------------------------

; Процедура: PrintString

; Назначение: Вывод на консоль строки, завершающейся нулем.

; Вход: RDX = адрес строки.

; ------------------------------------------------------------------------------

PrintString PROC

; Сохраняем регистры, которые могут быть изменены внутри процедуры.

push rbx

; Выделяем 40 байт на стеке для теневого пространства и выравнивания.

sub rsp, 40

; Цикл для определения длины строки.

mov rbx, rdx ; rbx - указатель на текущий символ.

xor rcx, rcx ; rcx - счетчик длины.

strlen\_loop:

cmp byte ptr [rbx], 0 ; Проверяем, не конец ли строки.

je strlen\_end

inc rbx

inc rcx

jmp strlen\_loop

strlen\_end:

; Подготовка аргументов для вызова WriteConsoleA.

mov r8, rcx ; Аргумент 3 (R8): длина строки.

mov rcx, [std\_output\_handle] ; Аргумент 1 (RCX): дескриптор вывода.

; RDX уже содержит адрес строки (Аргумент 2).

lea r9, [chars\_written] ; Аргумент 4 (R9): адрес переменной для кол-ва записанных байт.

mov qword ptr [rsp+32], 0 ; Аргумент 5 (на стеке): зарезервировано (NULL).

call WriteConsoleA

; Восстанавливаем стек и регистры.

add rsp, 40

pop rbx

ret

PrintString ENDP

; ------------------------------------------------------------------------------

; Процедура: ReadString

; Назначение: Чтение строки с клавиатуры.

; Вход: RDX = адрес буфера для сохранения строки.

; ------------------------------------------------------------------------------

ReadString PROC

; Сохраняем RBX, так как это нелетучий регистр, и мы его используем.

push rbx

; Выделяем 40 байт на стеке для теневого пространства и выравнивания.

sub rsp, 40

; Сохраняем адрес буфера (из RDX) в безопасный,

; нелетучий регистр RBX, потому что RDX будет уничтожен вызовом ReadConsoleA.

mov rbx, rdx

; Подготовка аргументов для вызова ReadConsoleA.

mov rcx, [std\_input\_handle] ; Аргумент 1 (RCX): дескриптор ввода.

mov rdx, rbx ; Аргумент 2 (RDX): адрес буфера.

mov r8, BUFFER\_SIZE - 1 ; Аргумент 3 (R8): максимальное кол-во символов для чтения.

lea r9, [chars\_read] ; Аргумент 4 (R9): адрес переменной для кол-ва прочитанных байт.

mov qword ptr [rsp+32], 0 ; Аргумент 5 (на стеке): зарезервировано (NULL).

call ReadConsoleA

; Читаем 32-битный DWORD в EAX. RAX автоматически обнуляется в старших битах.

mov eax, [chars\_read]

; Обработка прочитанной строки: установка нуль-терминатора

cmp rax, 0 ; Если ничего не прочитано, выходим.

je done\_reading

mov byte ptr [rbx + rax], 0 ; Ставим \0 в самый конец (за последний символ).

; Удаляем символы CR (\r) и LF (\n) из конца строки

cmp rax, 2

jl done\_reading

cmp byte ptr [rbx + rax - 2], 0Dh

jne done\_reading

cmp byte ptr [rbx + rax - 1], 0Ah

jne done\_reading

mov byte ptr [rbx + rax - 2], 0 ; Заменяем CR на \0, укорачивая строку.

done\_reading:

; Восстанавливаем стек и регистр.

add rsp, 40

pop rbx

ret

ReadString ENDP

; ------------------------------------------------------------------------------

; Процедура: Int64ToStr

; Назначение: Преобразование 64-битного знакового числа в строку.

; Вход: RCX = число, RDX = буфер для строки, R8 = основание системы.

; ------------------------------------------------------------------------------

Int64ToStr PROC

push rdi

push rsi

push rbx

push rcx

mov rdi, rdx ; rdi = указатель на буфер.

mov rbx, r8 ; rbx = основание.

mov rax, rcx ; rax = число.

xor rsi, rsi ; rsi = счетчик цифр.

cmp rax, 0 ; Отдельно обрабатываем случай, если число - ноль.

jne check\_sign

mov byte ptr [rdi], '0'

inc rdi

mov byte ptr [rdi], 0

jmp end\_conversion\_final

check\_sign:

test rax, rax ; Проверяем знак числа.

jns conversion\_loop ; Если не отрицательное, переходим к циклу.

mov byte ptr [rdi], '-' ; Иначе, ставим минус в начало строки.

inc rdi

neg rax ; И инвертируем число.

conversion\_loop:

xor rdx, rdx ; Обнуляем RDX перед делением.

div rbx ; rax = rax / rbx, rdx = остаток.

add dl, '0' ; Преобразуем остаток (0-9) в символ ('0'-'9').

push rdx ; Сохраняем символ цифры в стеке.

inc rsi ; Увеличиваем счетчик цифр.

cmp rax, 0 ; Повторяем, пока число не станет нулем.

jne conversion\_loop

pop\_digits\_loop:

pop rax ; Извлекаем цифры из стека (они будут в обратном порядке).

mov [rdi], al ; Записываем символ в буфер.

inc rdi

dec rsi ; Уменьшаем счетчик.

cmp rsi, 0

jne pop\_digits\_loop

end\_conversion\_final:

mov byte ptr [rdi], 0 ; Ставим нуль-терминатор в конце строки.

pop rcx

pop rbx

pop rsi

pop rdi

ret

Int64ToStr ENDP

; ------------------------------------------------------------------------------

; Процедура: HexStrToInt64

; Назначение: Преобразование 16-ричной строки в 64-битное знаковое число.

; Вход: RCX = адрес строки.

; Выход: RAX = результат.

; ------------------------------------------------------------------------------

HexStrToInt64 PROC

push rbx

push rsi

push rcx

xor rax, rax ; rax = аккумулятор результата.

mov rsi, rcx ; rsi = указатель на текущий символ.

xor rbx, rbx ; rbx = флаг отрицательного числа.

cmp byte ptr [rsi], '-' ; Проверяем наличие знака '-'.

jne parse\_loop

mov rbx, 1 ; Устанавливаем флаг.

inc rsi ; Пропускаем символ знака.

parse\_loop:

movzx rcx, byte ptr [rsi] ; Читаем байт-символ в RCX, обнуляя старшие биты.

cmp cl, 0

je parse\_end

; Преобразуем ASCII-символ в числовое значение (0-15).

cmp cl, '9'

jle is\_digit

cmp cl, 'F'

jle is\_upper\_hex

cmp cl, 'f'

jle is\_lower\_hex

jmp parse\_end ; Если символ некорректный, завершаем.

is\_digit:

sub cl, '0'

jmp accumulate

is\_upper\_hex:

sub cl, 'A' - 10

jmp accumulate

is\_lower\_hex:

sub cl, 'a' - 10

accumulate:

shl rax, 4 ; Умножаем аккумулятор на 16.

add rax, rcx ; Добавляем значение новой цифры.

inc rsi ; Переходим к следующему символу.

jmp parse\_loop

parse\_end:

cmp rbx, 1 ; Если был флаг отрицательного числа...

jne done

neg rax ; ...инвертируем результат.

done:

pop rcx

pop rsi

pop rbx

ret

HexStrToInt64 ENDP

END

**Компиляция и сборка**

Компиляция программы осуществлялась с использованием ассемблера ml64.exe, а сборка с linker.exe. Они идут в MSVC v143 – VS 2022 C++ x64/x86 Build Tools.

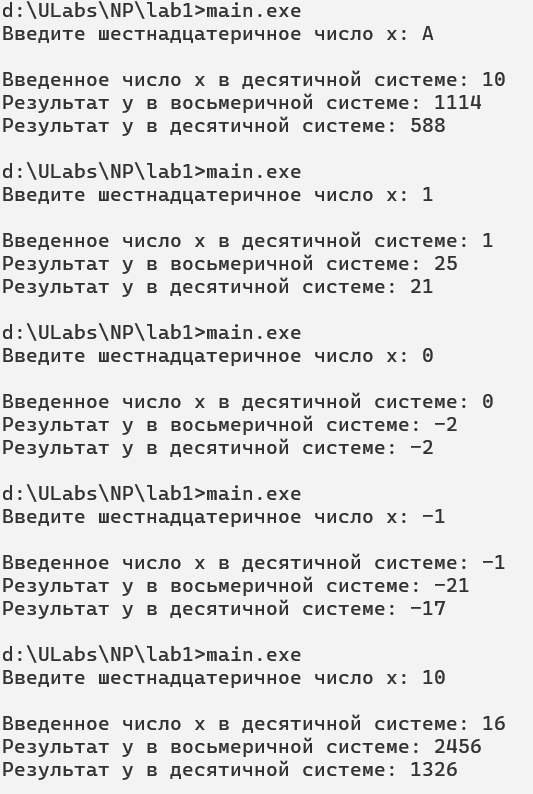


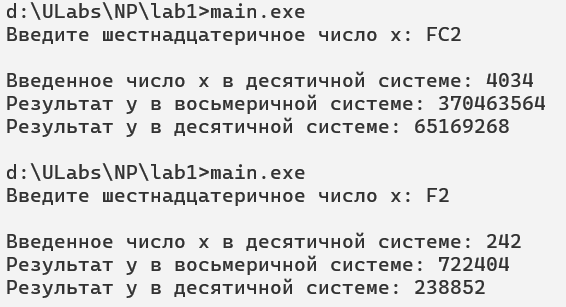
**Отладка**

Отладка производилась с помощью бесплатной утилиты x64dbg (<https://x64dbg.com/>). Она позволяет выполнять программу построчно, при этом смотреть ассемблер код исполняемой строки, смотреть сообщения об ошибках и исключениях от windows и видеть значения используемых регистров.

**Тестирование**

Результаты тестирования:





**Вывод**

Мы изучили и приобрели практические навыки работы с основными командами языка Ассемблера, функциями ввода-вывода, регистрами и символьными данными.