; Директива: указываем процессор (Pentium Pro и выше)

.686

; Директива: используем плоскую модель памяти и соглашение stdcall

.model flat, stdcall

; Директива: выделяем 256 байт для стека

.stack 100h

; Секция данных (переменные)

.data

X dw 5429h ; Переменная X (16 бит, значение 5429h)

Y dw 7844h ; Переменная Y (16 бит, значение 7844h)

Z dw 0AD43h ; Переменная Z (16 бит, значение AD43h)

Q dw 5622h ; Переменная Q (16 бит, значение 5622h)

R dw 0 ; Результат R (16 бит, инициализирован 0)

; Секция кода

.code

; Объявление функции ExitProcess из Windows API

ExitProcess PROTO STDCALL :DWORD

; Точка входа в программу

Start:

; Шаг 1: Увеличиваем X,Y,Z,Q на 1 и суммируем

mov esi, offset X ; ESI = адрес переменной X в памяти

mov ecx, 4 ; ECX = 4 (счетчик для цикла по 4 переменным)

xor ebx, ebx ; EBX = 0 (здесь будет сумма L)

loop\_inc:

mov ax, [esi] ; AX = текущее значение переменной (X, Y, Z, Q)

inc ax ; Увеличиваем AX на 1 (X+1, Y+1 и т.д.)

mov [esi], ax ; Сохраняем новое значение обратно в память

add bx, ax ; Добавляем AX к сумме L (BX = BX + AX)

add esi, 2 ; Переходим к следующей переменной (+2 байта)

loop loop\_inc ; Повторяем цикл, пока ECX > 0

; Шаг 2: Вычисление M = (L & X') - (L & Y')

mov ax, bx ; AX = L (сумма из BX)

and ax, [X] ; AX = L & X' (побитовое И)

mov dx, ax ; Сохраняем результат в DX

mov ax, bx ; Снова загружаем L в AX

and ax, [Y] ; AX = L & Y'

sub dx, ax ; DX = (L & X') - (L & Y') → M

; Выбор подпрограммы в зависимости от M

cmp dx, 921Bh ; Сравниваем M с 921Bh

jae call\_sp1 ; Если M >= 921Bh, переходим к call\_sp1

call subprog2 ; Если M < 921Bh, вызываем subprog2

jmp check\_even ; Переход к проверке четности R

call\_sp1:

call subprog1 ; Вызываем subprog1

check\_even:

; Шаг 3: Проверка четности R

mov ax, [R] ; Загружаем R в AX

test ax, 1 ; Проверяем младший бит (0 – четное, 1 – нечетное)

jz even\_r ; Если четное (ZF=1), переходим к even\_r

jmp odd\_r ; Если нечетное (ZF=0), переходим к odd\_r

even\_r:

or ax, 009Fh ; Побитовое ИЛИ: R = R | 009Fh

mov [R], ax ; Сохраняем результат

jmp ADDR1 ; Переход к метке ADDR1

odd\_r:

dec ax ; Уменьшаем R на 1: R = R - 1

mov [R], ax ; Сохраняем результат

jmp ADDR2 ; Переход к метке ADDR2

; Подпрограмма 1: R = M/2 - 12B9h

subprog1 PROC

mov ax, dx ; AX = M

shr ax, 1 ; Делим M на 2 (логический сдвиг вправо)

sub ax, 12B9h ; Вычитаем 12B9h: AX = M/2 - 12B9h

mov [R], ax ; Сохраняем результат в R

ret ; Возврат из подпрограммы

subprog1 ENDP

; Подпрограмма 2: R = M - Q'/2

subprog2 PROC

mov ax, [Q] ; AX = Q'

shr ax, 1 ; Делим Q' на 2: AX = Q'/2

mov bx, dx ; BX = M

sub bx, ax ; BX = M - Q'/2

mov [R], bx ; Сохраняем результат в R

ret ; Возврат из подпрограммы

subprog2 ENDP

; Метки для переходов (заглушки)

ADDR1: ; Код для четного R

jmp exit ; Переход к завершению

ADDR2: ; Код для нечетного R

jmp exit ; Переход к завершению

exit:

; Завершение программы с кодом 0

Invoke ExitProcess, 0

End Start ; Конец программы