ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

МИКРОПРОЕКТ

Программа для проверки взаимной простоты пяти чисел на языке ассемблера

Пояснительная записка

Работу выполнил
студент группы БПИ192
Ахметьянов Арслан Рашидович
31.10.2020

Москва 2020

Содержание

Постановка задачи

Согласно выданному заданию было необходимо реализовать следующий проект:

Разработать программу, решающую вопрос - являются ли пять заданных чисел взаимно простыми (числа задать машинными словами без знака).

Область допустимых входных данных

Программа принимает на вход 5 целых чисел.

По причине того, что вводимые числа сохраняются в машинные слова без знака, максимальное значение ограничено 65535 (верхняя граница ushort). Таким образом, если введено иное значение, то сохраняется его остаток от деления на 65536.

При вводе нулевого значения по модулю 65536 пользователю предлагается повторить ввод, поскольку взаимная простота не определена для пары чисел, если хотя бы одно из них нулевое.

Выходные данные

В зависимости от результата выполнения программы возможен один из двух выводов:

1. `Numbers are not coprime: two elements are %num1 and %num2; common divider = %divider.`
   - num1 – первое число рассматриваемой пары среду пяти чисел
   - num2 – второе число рассматриваемой пары среду пяти чисел
   - divider – их общий делитель, больший единицы

   Таким образом, данный вывод соответствует случаю, когда нашлась хотя бы одна пара не взаимно простых чисел среди пяти введенных.

2. `Every pair of numbers is coprime!`
   Данный вывод соответствует случаю, когда все пять введенных чисел попарно взаимно просты.

Алгоритм реализации

Проверка чисел на взаимную простоту реализована следующим образом:

1. Внешний цикл обходит элементы с первого по четвертый включительно (подпрограмма `CoprimeCheck`).
2. Вложенный цикл обходит все элементы, начиная со следующего за рассматриваемым во внешнем цикле, и заканчивает рассмотрением последнего (подпрограмма `InnerLoop`).

При таком обходе элементов массива будут рассмотрены все возможные пары различных введенных чисел.

Рассматриваемые числа сохраняются в num1, num2. При вызове подпрограммы `TwoIfCoprime` выполняется следующее:

1. Вызывается подпрограмма `DefineUpperDividerLimit`, устанавливающая в память upperLimit значение меньшего из чисел в num1 и num2. Это максимально возможное значение, которое может принимать общий делитель чисел в num1 и num2.
2. Проверяются все возможные значения divider от 2 до upperLimit включительно. Если на него делятся оба рассматриваемых числа,
    - производится вывод о том, что нашлась пара не взаимно простых чисел
    - в память coprime записывается значение 0 (по умолчанию в нем 1)
    - происходит выход из циклов

   Иначе рассматривается следующая пара чисел.

Если были рассмотрены все возможные пары чисел, и среди них не нашлось хотя бы одной пары взаимно простых (то есть значение coprime равно 1), то выводится информация о том, что все введенные числа попарно взаимно просты, и программа завершает выполнение.

Иначе программа завершает выполнение без дополнительного вывода, т.к. он уже был осуществлен.

Подпрограммы `ArrayInput` и `ArrayOutput` производят ввод и вывод 5 чисел соответственно.

Примеры входных данных

Далее в ожидаемом выводе я опущу печать введенных чисел, которая реализована в программе, чтобы не усложнять приведенный здесь вывод. Полностью вывод можно будет увидеть на скриншотах, размещенных на Github.

1. **Ввод:** 2, 3, 5, 7, 11
   Ожидаемый вывод: Every pair of numbers is coprime!

2. **Ввод:** 2, 5, 7, 60000, 24
   Ожидаемый вывод: Numbers are not coprime: two elements are 2 and 60000; common divider = 2!

3. **Ввод:** 2, 5, 7, -60, 12
   Примечание: число ‘-60’ будет интерпретировано как ‘65476’ – что будет отображено при выводе всех чисел в консоль.
   Ожидаемый вывод: Numbers are not coprime: two elements are 2 and 65476; common divider = 2!

4. **Ввод:** 0, 12, 5, 65536, 7, 33, 24
   Примечание: числа ‘0’ и ‘65536’ приведут к повтору вывода – поэтому во входных данных представлено 7 чисел.
   Ожидаемый вывод: Numbers are not coprime: two elements are 12 and 33; common divider = 3!

5. **Ввод:** 123, 32454, 577, 899, 2227
   Ожидаемый вывод: Numbers are not coprime: two elements are 123 and 32454; common divider = 3!

Список источников

1. http://av-assembler.ru/instructions/
2. http://flatassembler.narod.ru/fasm.htm
3. https://flatassembler.net/docs.php?article=manual#2.1.9
4. http://softcraft.ru/edu/comparch/

Приложение
(исходный код программы)

```
; Вариант 3 (Ахметьянов Арслан Рашидович, БПИ 192)
;         Разработать программу, решающую вопрос - являются ли пять заданных
чисел взаимно
;         простыми (числа задать машинными словами без знака).
format PE console

entry Start

        include 'win32a.inc'

section '.data' data readable writable

        ; Хранение чисел
        array rw 5

        ; Нахождение общего делителя
        num1 du 0
        num2 du 0
        divider du 1

        ; Результат обнаружения общего делителя - 0, если нашелся делитель,
отличный от единичного
        coprime dw 1

        ; Промежуточное хранение данных
        temp dd ?
        temp2 dd ?

        ; Хранение значения ecx для циклов
        innerCycle dd 1
        ecxStorage dd ?

        ; Хранение состояния стека
        espStorage dd ?
        coprimeEspStorage dd ?
        innerEspStorage dd ?

        ; Ввод чисел
        numIn db 'Enter positive number %d: ', 0
        elemOutputUshort db 'Elem [%d] = %hu', 10, 0
        ushort db '%hu', 0
        failureMessage db 'Entered number is equivalent to 0!', 10, 0

        ; Вывод информации о вводе
        arrayInfoString db 'Entered %d elements:', 10, 0


        ; Результат работы программы
        notCoprime db 'Numbers are not coprime: two elements are %hu and %hu;
common divider = %hu!', 10, 0
        successStr db 'Every pair of numbers is coprime!', 10, 0
```

```
        ; Ограничения значения делителя меньшим из рассматриваемых чисел
        upperLimit du ?

        NULL = 0

section '.code' code readable executable

        ; Подпрограмма ввода массива с клавиатуры
        ArrayInput:
                mov [espStorage], esp
                mov ebx, array          ; ebx = &array
                mov ecx, 5

                getVecLoop:
                        mov [temp], ebx
                        mov [ecxStorage], ecx

                        ; Подсчет индекса вводимого элемента
                        mov eax, 5
                        sub eax, [ecxStorage]
                        mov [temp2], eax

                elemInput:
                        push [temp2]
                        push numIn
                        call [printf]

                        ; Ввод элемента
                        push ebx
                        push ushort
                        call [scanf]

                        ; Проверка на положительность введенного числа - если
положительно, читаем следующее
                        cmp dword [ebx], 0
                        jg correctInput

                failedInput:
                        push failureMessage
                        call [printf]
                        jmp elemInput

                correctInput:
                        mov ecx, [ecxStorage]

                        ; Переход к следующему элементу
                        mov ebx, [temp]
                        add ebx, 2
                        loop getVecLoop

                inputEnd:
                        mov esp, [espStorage]
                        ret

        ; Подпрограмма вывода введенных чисел массива
```

```
        ArrayOutput:
                mov [espStorage], esp

                push 5
                push arrayInfoString
                call [printf]

                mov ecx, 5
                mov ebx, array ;ebx = &array
                printLoop:

                        mov [temp], ebx
                        mov [ecxStorage], ecx

                        ; Подсчет индекса выводимого элемента
                        mov eax, 5
                        sub eax, [ecxStorage]
                        mov [temp2], eax

                        ; Вывод элемента
                        push word [ebx]
                        push [temp2]
                        push elemOutputUshort
                        call [printf]

                        ; Переход к следующему элементу
                        mov ebx, [temp]
                        add ebx, 2

                        mov ecx, [ecxStorage]
                        loop printLoop

                outputEnd:
                        mov esp, [espStorage]
                        ret

        ; Подпрограмма, ограничивающая возможный делитель меньшим из чисел
        DefineUpperDividerLimit:
                xor ebx, ebx
                xor eax, eax

                mov bx, [num1]
                mov ax, [num2]
                cmp bx, ax
                ja num2Limit

                ; Если [num1] меньше или равен [num2], то в upperLimit
записывается [num1]
                xor ebx, ebx
                mov bx, [num1]
                jmp limitFinish

                num2Limit:
                        ; Если [num2] меньше [num1], то в upperLimit
записывается [num2]
                        xor ebx, ebx
```

```
                        mov bx, [num2]

                limitFinish:
                        mov [upperLimit], bx
                        ;mov esp, [eaxDebug]
                        ret


        ; Подпрограмма, проверяющая, являются ли два числа взаимно простыми
        TwoIfCoprime:
                mov [coprimeEspStorage], esp
                mov [divider], 1

                        ; Расчет ограничения делителя
                        call DefineUpperDividerLimit

                dividerLoop:
                        inc [divider]

                        ; Если потенциальный делитель больше максимально
возможного значения общего делителя -
                        ; числа взаимно просты
                        xor ebx, ebx
                        mov bx, [divider]
                        cmp bx, [upperLimit]
                        ja dividerExit

                        xor eax, eax
                        xor edx, edx

                        mov ax, [num1]
                        div [divider]

                        ; Если первое число не делится на делитель -
переходим к следующему делителю
                        cmp dx, 0
                        jg dividerLoop

                        xor eax, eax
                        xor edx, edx

                        mov ax, [num2]
                        div [divider]

                        ;  Если первое число делится на делитель, но не
второе - переходим к следующему делителю
                        cmp dx, 0
                        jg dividerLoop

                        ; Иначе записываем в [coprime], что нашлась пара не
взаимно простых чисел
                        mov [coprime], 0

                        ; Выводим информацию об этой паре и выходим
                        push dword [divider]
                        push dword [num2]
```

```
                        push dword [num1]
                        push notCoprime
                        call [printf]

                        jmp dividerExit

                dividerExit:
                        mov esp, [coprimeEspStorage]
                        ret

        ; Подпрограмма, проверяющая, есть ли среди чисел пара не взаимно
простых (внешний цикл)
        CoprimeCheck:
                mov [espStorage], esp

                mov ebx, array    ; ebx = &array
                mov [temp], ebx

                ; Итерируемся по элементам с 0 по 4 - сохраняем количество
итераций в ecx
                mov ecx, 4
                mov [ecxStorage], ecx

                goPrimeCheck:
                        ; Сохраняем в num1 первый рассматриваемый элемент
                        mov ebx, [temp]
                        mov dx, [ebx]
                        mov [num1], dx

                        ; Переводим ebx - теперь он указывает на следующий
элемент массива
                        add ebx, 2
                        mov [temp], ebx
                        mov [temp2], ebx

                        ; Сохраняем значение ecx для внешнего цикла
                        mov [ecxStorage], ecx

                        ; Запускаем внутренний цикл - начиная со следующего
после [num1] элемента
                        ; Заметим, что потребуется ровно ecx итераций: если в
ecx лежит x, то рассмотрено 5 - x элементов (считая текущий рассмотренным)
                        ; Значит, останется x элементов, которые нуно
поставить в пару к текущему
                        call InnerLoop

                        ; Если в InnerLoop не нашлось пары не взаимно простых
чисел, то продолжаем цикл - переходим к следющему num1
                        mov ecx, [ecxStorage]
                        loop goPrimeCheck

                fiveCoprimeCheckFinish:

                          mov esp, [espStorage]
                          ret
```

```
         ; Подпрограмма, проверяющая, есть ли среди чисел пара не взаимно
простых (внутренний цикл)
        InnerLoop:
                mov [innerEspStorage], esp

                ; Сохраняем значение ecx для внутреннего цикла
                mov [innerCycle], ecx

                inner:

                        ; Сохраняем в num2 второй рассматриваемый элемент
                        mov ebx, [temp2]
                        mov dx, [ebx]
                        mov [num2], dx

                        ; Проверям, являются ли они взаимно простыми
                        call TwoIfCoprime

                        ; Если значение 0 - нашлась пара не взаимно простых,
завершаем работу программы
                        cmp [coprime], 0
                        je fiveCoprimeCheckFinish

                        ; Переходим к следующему рассматриваемому элементу,
пишем его в num2
                        mov ebx, [temp2]
                        add ebx, 2

                        mov [temp2], ebx
                        mov dx, [ebx]
                        mov [num2], dx

                        ; Продолжаем, пока не переберем все num2, что не
рассматривались в паре с текущим num1 (зафиксирован во внешнем цикле)
                        loop inner

                ; Если взаимно не простая пара не нашлась, возвращаемся во
внешний цикт
                mov esp, [innerEspStorage]
                ret

        Start:

                ; Считали массив
                call ArrayInput

                ; Вывели массив
                call ArrayOutput

                ; Проверили, есть ли среди 5 чисел пара не взаимно простых
                call CoprimeCheck

                ; Если [coprime] = 0, то пара не взаимно простых нашлась =>
была выведена, завершаем работу
                cmp [coprime], 0
                je finish
```

```
                ; Иначе выводим сообщение, что все числа попарно взаимно
просты
                successFinish:
                        push successStr
                        call [printf]

                finish:

                        call [getch]

                        push NULL
                        call [ExitProcess]

section '.import' import data readable

        library kernel, 'kernel32.dll',\
                msvcrt, 'msvcrt.dll'

        import kernel,\
               ExitProcess, 'ExitProcess'

        import msvcrt,\
               printf, 'printf',\
               scanf, 'scanf',\
               getch, '_getch'
```