НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Курсовая работа

«Сравнение быстродействия программ на Assembler и C++»

по дисциплине

Программирование на языках низкого уровня в задачах защиты информации

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дмитриев Д.В.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Голубев А.Д.

М18-ИСТ-1

Нижний Новгород

2019 г.

Содержание

[**Введение** 3](#_Toc10675186)

[**1.** **Программа с операциями над матрицами** 4](#_Toc10675187)

[**1.1.** **Листинг программ** 4](#_Toc10675188)

[**1.1.1.** **Assembler:** 4](#_Toc10675189)

[**1.1.2.** **C++:** 9](#_Toc10675190)

[**1.2.** **Сравнение программ** 11](#_Toc10675191)

[**1.2.1.** **Сравнение по размеру исполняемого файла** 11](#_Toc10675193)

[**1.2.2.** **Сравнение по быстродействию** 11](#_Toc10675194)

[**1.2.3.** **Сравнение по объему оперативной памяти** 12](#_Toc10675195)

[**Заключение** 13](#_Toc10675196)

# **Введение**

Assembler — язык программирования низкого уровня, представляющий собой формат записи машинных команд, удобный для восприятия человеком.

Команды языка ассемблера один в один соответствуют командам процессора и, фактически, представляют собой удобную символьную форму записи (мнемокод) команд и их аргументов. Также язык ассемблера обеспечивает базовые программные абстракции: связывание частей программы и данных через метки с символьными именами и директивы.

Директивы ассемблера позволяют включать в программу блоки данных (описанные явно или считанные из файла); повторить определённый фрагмент указанное число раз; компилировать фрагмент по условию; задавать адрес исполнения фрагмента, менять значения меток в процессе компиляции; использовать макроопределения с параметрами и др.

Каждая модель процессора, в принципе, имеет свой набор команд и соответствующий ему язык (или диалект) ассемблера.

Достоинства и недостатки:

* минимальное количество избыточного кода (использование меньшего количества команд и обращений в память). Как следствие — большая скорость и меньший размер программы
* большие объемы кода, большое число дополнительных мелких задач
* плохая читабельность кода, трудность поддержки
* трудность реализации парадигм программирования
* меньшее количество доступных библиотек, их малая совместимость по сравнению с другими языками
* непосредственный доступ к аппаратуре: портам ввода-вывода, особым регистрам процессора
* максимальная адаптация для нужной платформы (использование специальных инструкций, технических особенностей «железа»)
* непереносимость на другие платформы (кроме двоично совместимых).

C++ — компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C, — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

Достоинства и недостатки

* Высокая совместимость с языком Си: код на Си может быть с минимальными переделками скомпилирован компилятором C++.
* Вычислительная производительность. Язык спроектирован так, чтобы дать программисту максимальный контроль над всеми аспектами структуры и порядка исполнения программы.
* Поддержка различных стилей программирования.
* Перегрузка операторов.
* Шаблоны C++ дают возможность построения обобщённых контейнеров и алгоритмов для разных типов данных.
* Доступность.
* Унаследованные от Си опасные и провоцирующие ошибки возможности, такие как макроопределения #define, адресная арифметика и неявное приведение типов.
* Возможность прямого управления распределением памяти провоцирует ошибки, приводящие к внезапному краху программ из-за разрушения стека или обращения к невыделенной памяти.

Сложный синтаксис и объёмная спецификация языка затрудняют его изучение.

* Сложная и постоянно разрастающаяся стандартная библиотека, затрудняющая изучение и дополнительно увеличивающая объём программ.
* Отсутствие или ограниченность поддержки ряда полезных технологий и методик программирования.

1. **Программа с операциями над матрицами**

В числовом двумерном массиве вычислить среднее арифметическое максимальных элементов столбцов матрицы.

* 1. **Листинг программ**
     1. **Assembler:**

.686P

.model flat, stdcall

option casemap:none

include ..\irvine\Irvine32.inc

include ..\masm32\include\msvcrt.inc

includelib ..\irvine\User32.lib

includelib ..\irvine\Kernel32.lib

includelib ..\irvine\Irvine32.lib

includelib ..\masm32\lib\msvcrt.lib

; ограничение на размер массива

maxSize = 1000000

; инициализация массива

InitMat PROTO,

lpArray:PTR DWORD

; сумма максимальных элементов по столбцам

ArithmeticMeanOfColumns PROTO,

lpArray:PTR DWORD,

mCols:DWORD,

mRows:DWORD

; печать матрицы

PrintMat PROTO,

lpArray:PTR DWORD,

mRows:DWORD,

mCols:DWORD

.data?

rows DWORD ? ; число строк

cols DWORD ? ; число столбцов

matrixSize DWORD ? ; количество элементов

vSum DWORD ?

vAmountOfEl DWORD ?

vArithMeanInt DWORD ?

vArithMeanRem DWORD ?

shift DWORD ?

startTime SYSTEMTIME <>

endTime SYSTEMTIME <>

Array dword maxSize dup(?)

.const

maxRandNumb dword 100

sConsoleTitle byte "Lab 2 ASM", 13, 10, 0

sSize byte "Enter a Mat size: ", 0

sMat byte "Matrix [%d x %d]: ", 13, 10, 0

sFormat byte " %d",0

sTime BYTE "Time = ",0

sSpace byte " ", 0

sError byte "Err", 13, 10, 0

sSum byte "Sum: %d", 13, 10, 0

sAmountOfEl byte "Amount of elements: %d", 13, 10, 0

sArithMean byte "Arithmetic Mean Of Columns: %d.%d", 13, 10, 0

sExit byte "Press any key to Exit", 13, 10, 0

fm db '%d, ',0

.code

main PROC

; заголовок(title) консоли

;invoke SetConsoleTitleA, addr sConsoleTitle

invoke GetSystemTime, addr startTime

invoke InitMat, addr Array

;invoke crt\_printf, addr sMat, rows, cols

;invoke PrintMat, addr Array, rows, cols

invoke ArithmeticMeanOfColumns, addr Array, cols, rows

;invoke crt\_printf, addr sExit

invoke GetSystemTime, addr endTime

movzx eax, endTime.wMilliseconds

movzx ebx, startTime.wMilliseconds

sub eax, ebx

mov edx,OFFSET sTime

call WriteString

call WriteInt

; \_getch()

invoke crt\_\_getch

invoke ExitProcess, 0

main ENDP

InitMat PROC,

lpArray: PTR DWORD

pushad

;invoke crt\_printf, addr sSize

;invoke ReadInt

mov eax, 1000

; проверки ввода

jo error

cmp eax, 0

jle error ; меньше или равно нулю

push eax

mul eax

cmp eax, maxSize

jg error ; превышает maxSize

pop eax

mov rows, eax

mov cols, eax

mul eax

mov matrixSize, eax

; готовим регистры для цикла заполнения

mov ebx, lpArray

mov esi, 0

mov ecx, matrixSize

; ГПСЧ

invoke Randomize ; инициализурем генератор случайных чисел

InitVal:

mov eax, maxRandNumb

invoke RandomRange

mov [ebx + esi], eax ; сохраняет сгенерированное значение

add esi, TYPE lpArray

loop InitVal

popad

ret

; переход при ошибке

error:

invoke crt\_printf, addr sError

popad

invoke ExitProcess, 0

InitMat ENDP

ArithmeticMeanOfColumns PROC,

lpArray:PTR DWORD,

mCols:DWORD,

mRows:DWORD

pushad

mov eax, mCols

imul eax, TYPE lpArray

mov shift, eax

mov eax, 0

mov ebx, 0

mov ecx, mCols

cycleCols:

push ecx

mov ecx, mRows

mov edx, Array[ebx]

mov esi, shift ; смещение второго элемента столбца(cols\*TYPE arr)

cycleEl:

cmp edx, Array[ebx] + [esi]

jge next

mov edx, Array[ebx] + [esi]

next:

add esi, shift

loop cycleEl

add eax, edx

pop ecx

add ebx, TYPE lpArray

loop cycleCols

mov vSum, eax

mov edx, 0

mov ebx, mCols

idiv ebx

mov vArithMeanRem, edx

mov vArithMeanInt, eax

; вывод

;invoke crt\_printf, ADDR sSum, vSum

;invoke crt\_printf, ADDR sAmountOfEl, mCols

;invoke crt\_printf, ADDR sArithMean, vArithMeanInt, vArithMeanRem

popad

ret

ArithmeticMeanOfColumns ENDP

PrintMat PROC,

lpArray:PTR DWORD,

mRows:DWORD,

mCols:DWORD

pushad

invoke crt\_printf, addr sMat, mCols, mRows

mov ebx, lpArray

mov esi, 0 ; устанавливаем на 0

mov ecx, mRows

loopCols:

push ecx

mov ecx, mCols

LoopRows:

; выводит число

mov eax,[ebx + esi]

invoke WriteInt

; следующее число

add esi, TYPE lpArray

; печатает пробел между числами массива

mov edx, offset sSpace

invoke WriteString

loop LoopRows

invoke Crlf ; переход на новую строку

pop ecx

loop loopCols

popad

ret

PrintMat ENDP

END main

* + 1. **C++:**

#include <Windows.h>

#include <iostream>

#include <conio.h>

#include <ctime>

bool initMat(std::uint32\_t \*\*\*arr, const std::uint32\_t& size);

std::double\_t calcArithMean(std::uint32\_t \*\*\*arr, const std::uint32\_t& size);

void printMat(std::uint32\_t \*\*\*arr, const std::uint32\_t& size);

int main()

{

std::double\_t workTime = std::double\_t(clock()) / 1000;

std::uint32\_t size = 10000;

std::uint32\_t \*\*arr = nullptr;

srand(time(NULL));

//std::cout << "Enter a mat size: ";

//std::cin >> size;

initMat(&arr, size);

//printMat(&arr, size);

std::double\_t arithMean = calcArithMean(&arr, size);

//std::cout << "Arithmetic mean: " << arithMean << std::endl;

for (std::uint32\_t i = 0; i < size; i++)

{

delete[] arr[i];

}

delete arr;

arr = nullptr;

workTime = std::double\_t(clock()) / 1000 - workTime;

std::cout << "Working time " << ": " << workTime << std::endl;

\_getch();

return 0;

}}

bool initMat(std::uint32\_t \*\*\*arr, const std::uint32\_t& size)

{

if (size <= 0) return false;

(\*arr) = new std::uint32\_t \*[size];

for (std::uint32\_t i = 0; i < size; i++)

{

(\*arr)[i] = new std::uint32\_t[size];

for (std::uint32\_t j = 0; j < size; j++)

{

(\*arr)[i][j] = rand() % 100;

}

}

}

std::double\_t calcArithMean(std::uint32\_t \*\*\*arr, const std::uint32\_t& size)

{

std::uint32\_t sumEl = 0;

for (std::uint32\_t j = 0; j < size; j++)

{

std::uint32\_t maxEl = 0;

for (std::uint32\_t i = 0; i < size; i++)

{

if ((\*arr)[i][j] > maxEl)

{

maxEl = (\*arr)[i][j];

}

}

sumEl += maxEl;

maxEl = 0;

}

return std::double\_t(sumEl) / std::double\_t(size);

}

void printMat(std::uint32\_t \*\*\*arr, const std::uint32\_t& size)

{

std::cout << std::endl << "Print matrix: " << std::endl;

for (std::uint32\_t i = 0; i < size; i++)

{

std::cout << "[" << i << "] ";

for (std::uint32\_t j = 0; j < size; j++)

{

std::cout << (\*arr)[i][j] << " ";

}

std::cout << std::endl;

}

std::cout << std::endl;

}

* 1. **Сравнение программ**

Для сравнения программ были проведены эксперименты в количестве 100 шт. для пунктов «Сравнение по быстродействию» и «Сравнение по объему оперативной памяти» для повышения качества данных. Многократное повторение эксперимента может позволить говорить об объективности эксперимента в независимости от текущей загрузки вычислительной машины. В момент выполнения исследования никакие дополнительные программы не включались и не отключались.

* + 1. **Сравнение по размеру исполняемого файла**

Первым критерием сравнения является размер исполняемого файла:

|  |  |
| --- | --- |
| Assembler | C++ |
| 10 kB | 13 kB |

Таблица 1 - Сравнение размеров исполняемых файлов

Программа на языке низкого уровня занимает меньше места. Возможно, это связано с использованием компиляторов Visual Studio 2017. Наиболее вероятно, что они добавляют какие-то свои средства для оптимизации производительности.

* + 1. **Сравнение по быстродействию**

Вторым критерием оценки является быстродействие программы. Для проверки быстродействие работы программ используем профилировщик Visual Studio 2017. Для исследования использовались массивы разной размерности: «5», «20», «100». «1000». Для корректной оценки быстродействия программ было принято решение исключить операции вывода информации на экран, а также ввод чисел пользователем. Также стоит отметить, что использование одного и тоже инструмента для измерения быстродействия также повышает надежность данных.

|  |  |  |
| --- | --- | --- |
|  | Assembler | C++ |
| 5 | 0.00001 s | 0.00001 |
| 20 | 0,00001 s | 0,00002 s |
| 100 | 0,00009 s | 0,00032 s |
| 1000 | 0,00773 s | 0,02611 s |

Таблица 2 - Сравнение быстродействия программ(профилировщик)

|  |  |  |
| --- | --- | --- |
| 5 | Невычисляемо | Невычисляемо |
| 20 | Невычисляемо | 0,00001 s |
| 100 | 0,00012 s | 0,00027 s |
| 1000 | 0,00110 s | 0,02751 s |

Таблица 3. Сравнение быстродействия программ(средства языка)

Рисунок 1 - График быстродействия программ(профилировщик)

Результаты измерения показывают, что для малых и средних значений программа на языке высокого уровня имеет небольшое преимущество, однако для программ с большим количеством элементов программа, написанная на Assembler, показывает лучшие результаты по сравнению с аналогичной программой, написанной на С++.

* + 1. **Сравнение по объему оперативной памяти**

Третьим критерием оценки являются затраты оперативной памяти. Для определения затрат оперативной памяти использовался Диспетчер задач. Учитывались изначальные затраты ОП, т.к. при длительной работе (размер матрицы 10 000 для программы на С++), выделение памяти увеличивалось в процессе работы программы для ускорения выполнения операций.

|  |  |  |
| --- | --- | --- |
|  | Assembler | C++ |
| 5 | 0.98 mb | 0.744 mb |
| 20 | 0.98 mb | 0.824 mb |
| 100 | 0.98 mb | 0.832 mb |
| 1000 | 5.3 mb | 5 mb |

Таблица 4 - Сравнение выделяемой оперативной памяти(профилировщик)

|  |  |  |
| --- | --- | --- |
|  | Assembler | C++ |
| 5 | 6 mb | 5.9 mb |
| 20 | 6 mb | 5.9 mb |
| 100 | 6 mb | 6.1 mb |
| 1000 | 9.8 mb | 10 mb |

Таблица 4 - Сравнение выделяемой оперативной памяти(диспетчер задач)

Рисунок 2 - График выделения оперативной памяти для программ

Выполнение не так сильно отличается от программы на языке С++. Возможно, это связано с плохой оптимизацией кода на Assembler или особенностей компилятора Visual Studio 2017.

# **Заключение**

В ходе выполнения курсовой работы были изучены основы низкоуровнего языка программирования Assembler. Были проведены исследования по сравнению быстродействия и ресурсоемкости программ, написанных на языках низкого и высокого уровня. В результате работы можно сделать вывод, что программы, написанные на Assembler, показывают практически одинаковые результаты по этим показателям, однако скорость написания программ, а также читаемость кода остается за языками высокого уровня.