Глава 1

Векторная арифметика

Файл: Makefile

```
lab03.exe: lab03.obj
	ld lab03.obj -l kernel32 -o lab03.exe

lab03.obj: lab03.asm
	nasm lab03.asm -f win32
```

1.1 Система сборки CMake

Процесс сборки приложения из нескольких файлов может отличаться на разных компьютерах, например, могут использоваться разные пути для библиотек или использоваться разные компиляторы. Поэтому ручное написание Makefile является не желательным. Для создания Makefile обычно используют специальные утилиты, например, CMake, которую мы и рассмотрим в этой лабораторной работе.

1. Создайте рабочую директорию lab06 с репозиторием Mercurial.

2. Создайте файлы main.asm и CMakeLists.txt (обратите внимание на регистр символов в названии файлов, это будет важно при работе в ОС Linux):

Файл: main.asm

```
global  _main                   ; вход в программу
extern  _ExitProcess@4
extern  _printf

section .text   use32
_main:
        push    msg
        call    _printf
        add     esp, 4

        xor     eax, eax
        push    eax
        call    _ExitProcess@4
section .data
msg     db      "Hello World!!!", 13, 10, 0
```

Файл: CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.12)
project(lab06 C ASM_NASM)
add_executable(lab06 "main.asm")
```

```
set_target_properties(lab06 PROPERTIES LINKER_LANGUAGE C)
```

В первой строке файла CMakeLists.txt минимальная версия CMake, необходимая для сборки проекта. Во второй задаётся имя проекта и используемые языки программирования, хотя в нашей программе сейчас нет кода на языке Си, нам будет необходимы инструменты линковки. В третьей строке создаётся исполняемый файл lab06 и файлы с исходным кодом для его компиляции. В четвёртой строке мы указываем, что для линковки исполняемого файла надо использовать средства поставляемые с компилятором языка Си.

Далее скомпилируем нашу программу с использованием компиляторов MinGW и MS Visual Studio. Если раньше мы компилировали программы в той же директории, где хранились и исходные файлы, то теперь мы уйдём от этой вредной привычки.

3. Скомпилируйте приложение с использованием инструментов MinGW:

1. Создайте директорию build-mingw.
2. Откройте новый сеанс командной строки.
3. Перейдите в директорию lab06/build-mingw.
4. Внесите в начало переменой окружения PATH пути до NASM, CMake и MinGW.
5. Создайте Makefile для MinGW используя CMake:

```
.../lab06/build-mingw> cmake -G "MinGW Makefiles" ..
```

6. Посмотрите на созданный файл.
7. Соберите приложение (mingw32-make).
8. Проверьте работу программы.

4. В директории lab06 создайте файл .hgignore:

```
syntax: glob
build*
```

5. Зафиксируйте изменения с комментарием: «Init. Test compiling with MinGW tools.».

Теперь пришло время воспользоваться инструментами MS Visual Studio для сборки приложения. В дальнейшем мы будем их использовать для создания 64-разрядных приложений.

6. Попробуйте скомпилировать приложение с использованием MSVC:

1. В lab06 создайте директорию build-msvc.
2. Откройте новый сеанс командной строки.
3. Перейдите в директорию lab06/build-msvc.
4. Настройте окружения для работы с MSVC в x86-режиме:

```
...> call "{путь до MS VS}/VC/vcvarsall.bat" x86
```

5. Внесите в начало переменой окружения PATH пути до NASM и CMake.
6. Создайте Makefile для MSVC используя CMake:

```
.../lab06/build-msvc> cmake -G "NMake Makefiles" ..
```

7. Посмотрите на созданный файл.
8. Соберите приложение:

```
.../lab06/build-msvc> nmake
```

9. Что произошло? Можете ли вы самостоятельно сказать причину ошибки?

Как видно, линковщик не увидел реализацию двух функций. Эти функции находятся в среде выполнения языка Си, которая не была подключена к исполняемому файлу. Следовательно нашей задачей будет подключение необходимой библиотеки, но надо иметь в виду, что есть две реализации этой библиотеки: динамическая и статическая. По умолчанию используется динамическая библиотека, поэтому надо будет линковаться с библиотекой msvcrtd.

7. Определите правило линковки с библиотекой msvcrtd для исполняемого файла, при использовании инструментов MSVC в CMakeLists.txt:

```
cmake_minimum_required(VERSION 2.8.12)
project(lab06 C ASM_NASM)

add_executable(lab06 "main.asm")
set_target_properties(lab06 PROPERTIES LINKER_LANGUAGE C)

if(MSVC)
    target_link_libraries(lab06 msvcrtd)
endif()
```

8. Повторите сборку с помощью nmake.
9. Зафиксируйте изменения с комментарием «Fix for MSVC».

1.2 Библиотека функций

В этой части работы мы рассмотрим пример реализации вычислительной функции на Ассемблер и её вызов из программ написанных как на языке Ассемблер, так и на Си.

1.2.1 Соглашения о вызове cdecl

Наша задача написать 32-битную функцию для x86 процессоров, которая будет вызываться, в итоге, из программы написанной на языке Си. Для этого мы должны придерживаться правил передачи параметров и приёма результата определённых в этом языке. Такие правила называются «Соглашение о вызове», и в нашем случае нужно использовать соглашение cdecl.

Напомню общие правила cdecl. Параметры функции передаются через стек в направлении справа-налево. Выравнивание параметров: 4 байта. Результат функции передаются: в EAX для данных размером ≤ 4 байт; в паре EAX:EDX для данных размером ≤ 8 байт; в регистре ST0 математического сопроцессора для действительных чисел; иначе вызывающая функция выделяет память для результата и помещает адрес в вершину стека (так называемый «скрытый параметр»), ожидая что в EAX будет реальный адрес, где сохранился результат. Вызывающая функция должна очистить стек самостоятельно. Подпрограмма может изменить состояние регистров EAX, ECX и EDX, значения остальных должны быть восстановлены подпрограммой. Стек математического сопроцессора должен быть пустым перед и после вызова.

1.2.2 Подпрограмма Пи

Как вам уже известно, константа пи вшита в математический сопроцессор, воспользуемся ей.

10. Создайте файл mylib.asm:

```
global  _pi

section .text   use32
_pi:
        fldpi
        ret
```

11. Теперь добавим новый файл в систему сборки cmake. Исправьте файл CMakeLists.txt:

```
...
add_executable(lab06 "main.asm" "mylib.asm")
...
```

12. Исправьте файл main.asm:

```
global  _main
extern  _ExitProcess@4
extern  _printf
extern  _pi

section .text   use32
_main:
        finit
        call    _pi
        sub     esp, 8
        fstp    qword [esp]
        push    msg
        call    _printf
        add     esp, 4+8

        xor     eax, eax
        push    eax
        call    _ExitProcess@4
section .data
msg     db      "Pi = %.20f", 13, 10, 0
```

13. Скомпилируйте программу и проверьте её работу. Сравните результат с эталонным значением числа Пи.

14. Добавьте файл mylib.asm под контроль версий.

15. Зафиксируйте изменения с комментарием «Added function pi.».

Теперь сравним точность представления чисел в float и double. Для этого сохраним результат функции дважды в разных форматах и выведем результат на экран используя printf. Но на втором этапе нас ожидает небольшая загвоздка: мы не можем передать переменную типа float напрямую в printf, необходимо преобразовать его в double (пункт 6.5.2.2-6 стандарта C99 [http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1256.pdf]), т. к. она использует переменное количество параметров.

16. Добавьте вывод числа π в формате float.

```
...
        call    _pi
        sub     esp, 16
        fst     dword [esp]
        fstp    qword [esp+8]
        fld     dword [esp]
        fstp    qword [esp]
        push    msg
        call    _printf
        add     esp, 4+16
...
msg     db      "Pi(float)  = %.20f", 13, 10
        db      "Pi(double) = %.20f", 13, 10, 0
```

17. Скомпилируйте программу и проверьте её работу. Какими командами происходит преобразование числа из float в double?

18. Зафиксируйте изменения с комментарием «Print float pi.».

1.2.3 Вызов функции из программы на языке Си

Теперь пришло время задействовать язык Си. Мы перепишем основную функцию нашей программы. Для этого корректно удалим файл main.asm и создадим новый main.cpp. Кроме этого нам понадобиться заголовочный файл mylib.h для нашей библиотеки.

19. Средствами Mercurial удалите файл main.asm:

```
.../lab06> hg remove main.asm
```

20. Добавьте файлы main.c и mylib.h:

Файл: mylib.h

```
#ifndef _MYLIB_H_
#define _MYLIB_H_

double pi();

#endif // _MYLIB_H_
```

Файл: main.c

```
#include <stdio.h>

#include "mylib.h"

int main()
{
→double pi_d = pi();
→float pi_f = (float)pi_d;
→printf(
→→"Pi          ~ 3.14159265358979323846\n"
→→"Pi(float)   = %.20f\n"
→→"Pi(double) = %.20f\n", pi_f, pi_d);
→return 0;
}
```

21. Подправим файл CMakeLists.txt:

```
cmake_minimum_required(VERSION 2.8.12)
project(lab06 C ASM_NASM)

add_executable(lab06 "main.c" "mylib.h" "mylib.asm")
set_target_properties(lab06 PROPERTIES LINKER_LANGUAGE C)
```

Стоит отметить, что теперь нет необходимости в явном указании необходимости линковки со средой выполнения Си, т. к. в нашем проекте присутствует файл с исходным кодом на Си.

22. Скомпилируйте программу и проверьте её работу. Сравните результат.

23. Добавьте новые файлы под контроль версий.

24. Зафиксируйте изменения с комментарием «Call pi from C.».

Найдите способ генерации кода на языке Ассемблер из исходного кода на языке Си используя компиляторы mingw32-gcc (из пакета MinGW) и cl (из MS VS). Сравните результат сгенерированного кода с вашей реализацией.