«Генерация и анализ ассемблерного кода программы на Си под разные архитектуры»

Цель задания:

- Научиться генерировать и анализировать ассемблерный код на языке Си под разные архитектуры (x86-64, ARM64, RISC-V).
- Понять, как высокоуровневые конструкции языка Си реализуются на уровне машинных инструкций, и как компилятор адаптирует код под разные платформы.

Общая часть

Напишите программу на Си, соответствующую вашему варианту (см. ниже).
Программа должна быть минимальной по объёму, но полноценно демонстрировать указанную языковую конструкцию.
Программа не должна содержать ввода/вывода — только логика.
Скомпилируйте программу под указанные архитектуры и получите ассемблерный листинг командой:

<компилятор> -S -O0 program.c

В отчёте:

Приведите исходный код программы.
Опишите процесс получения ассемблерного листинга (если, нужно было установить компилятор, нужно описать процесс)
Приведите ассемблерные листинги для всех запрошенных архитектур. (семейство процессора/версия ОС/версия компилятора)
Кратко опишите различия в реализации конструкции (регистры, инструкции, структура кода).

Варианты программ

Вариант	Задание
0,5	Переменные и выражения: инициализация нескольких переменных разных типов (char, int, double), арифметическое выражение с ними (сохранение результата в одну из них) Пример: $F = G\frac{m_1 m_2}{r^2}$, Закон всемирного тяготения Ньютона, $c = \frac{1}{\sqrt{\mu_0 \epsilon_0}}$, Скорость распространения электромагнитных волн Максвелла
1,6	Условный оператор: написать условия отношений переменной к определенному диапазону значений и выбор конченого значения y в зависимости от x. Пример: если x от – 1000 до – 100, то y = x^2, если x от -99 до 99, то y = x^3, иначе y=x
2,7	Циклы (for или while, на выбор): написать функцию, которая вычисляет произведение чисел от 1 до N (где N может <= 19 и является аргументом функции).
3,8	Множественное ветвление (switch): написать функцию которая присваивает строке типа C-style соответствующего цвета: если char ch='r' – "red", если 'b' –

	“blue”, если ‘g’ = “green”, если ‘w’ = “white”, если любое другое то “black”.
4,9	Функции с указателями: написать свои собственные функции, которые выполняют инкремент и декремент с использованием указателей.

Критерии оценки

Уровень «Удовлетворительно»
Написана программа в соответствии с вариантом. Скомпилирован ассемблерный код под ***одну*** архитектуру (основанная ОС на которой компилируется программа). Представлены листинги и краткий анализ.

Уровень «Хорошо»

Выполнено всё для «3»
+ скомпилирован дополнительный листинг (***второй***) под Windows (x86-64) с помощью MinGW-w64/x86. В анализе — сравнение Linux vs Windows

Уровень «Отлично»

Выполнено всё для «4»
+самостоятельно установлен кросс-компилятор, например **riscv64-linux-gnu-gcc** (**gcc-arm** или прочие аналоги) и проведена компиляция вручную (***третьего листинга***). В отчёте — описание процесса установки и подтверждение корректности генерации кода. Анализ и сравнение с предыдущими листингами.

Требования к оформлению отчёта (формализовано)

Отчёт оформляется в электронной форме и должен содержать следующие разделы:

1. Титульный лист

- По шаблону СибГУТИ
- Название работы: «Генерация и анализ ассемблерного кода программы на Си под разные архитектуры» вариант.
- ФИО студента, группа
- Дата сдачи

2. Исходный код программы на С

Полный и корректный код, соответствующий варианту
Без #include, printf, scanf, main можно опустить, если функция вынесена отдельно, но должен быть воспроизводимый пример вызова

3. Процесс получения ассемблерных листингов

Указание ОС и версии (например, Альт Рабочая станция p11, x64)
Версии компиляторов: gcc --version, x86_64-w64-mingw32-gcc --version, riscv64-linux-gnu-gcc --version
Команды компиляции для каждой архитектуры
Для уровня «5» — описание установки кросс-компилятора RISC-V (например: sudo apt install gcc-riscv64-linux-gnu)

4. Ассемблерные листинги

Три листинга (для уровней 4 и 5):
x86-64 (Linux, gcc)
x86-64 (Windows, x86_64-w64-mingw32-gcc)
RISC-V 64 (с помощью riscv64-linux-gnu-gcc)
Каждый листинг должен быть подписан: архитектура, компилятор, флаги (-S -O0)

5. Анализ и сравнение

Различия в именах и количестве регистров (например, %rax vs x0 vs a0)
Соглашения о вызовах: передача аргументов (регистры vs стек), возврат значения
Структура кода: наличие/отсутствие пролога/эпилога, использование стека
Специфика инструкций: например, как реализовано условие или цикл
Для уровня 4: сравнение ABI Linux (System V) и Windows (Microsoft x64)
Для уровня 5: особенности RISC-V (load/store архитектура, регулярность инструкций)

6. Вывод

Что вы узнали о связи C и архитектуры?
Как компилятор адаптирует один и тот же C-код под разные платформы?
Почему знание ассемблера важно для системного программиста?