Приложение Б. Лабораторный практикум по организации ЭВМ и GNU Assembler

Регламент курса в целом описан в приложении А.

РЛ. Регламент лабораторных работ

Баллы (базовые — график снижения в таблице РЛ.1 — и бонусные) за лабо-

График снижения базовых баллов за лабораторные работы

Таблица РЛ.1

max (Лi) для ОЭВМ и Асм (12 занятий): группы ПИН-21–24													
Неделя	Л1	Л2	Л3	Л4	Л5	Л6	Л7	Л8	Л9	ЛА	ЛВ	3–4	$\geqslant 5$
1, 2	9											3	1
3, 4	9	9	9									3	1
5, 6	8	9	9	9								3	1
7, 8	7	8	8	9	9	9						3	1
9, 10	6	7	7	8	9	9	9					3	1
11, 12	5	6	6	7	8	8	9	9	9			3	1
13, 14	—	5	5	6	7	7	8	9	9	9		1	1
15, 16	—	—	—	5	6	6	7	8	8	9	9	1	1
15, 16 (с нуля) 17, 18, сессия	л/р не принимаются — пишите КрЭВМ или выполняйте индивидуальное задание												

max (Лi) для ОЭВМ (8 занятий): группы НБ-3*, ПМ-31–32, ПИН-35–36										
НБ-3*	Л1	Л2	Л7 на C++	ЛВ на C++	Л3	Л4	Л5	Л6	3–4 шт.	$\geqslant 5$ шт.
ПМ-31–32	ЛС	Л1	Л2	Л3	Л4	Л5	Л6	Л7	3–4 шт.	$\geqslant 5$ шт.
ПИН-35–36	Л1	Л2	Л3	Л4	Л5	Л6	Л7	Л8	3–4 шт.	$\geqslant 5$ шт.
1, 2	12								3	1
3, 4	12	12							3	1
5, 6	11	12	12						3	1
7, 8	10	11	12	12					3	1
9, 10	9	10	11	12	12				3	1
11, 12	8	9	10	11	12	12			3	1
13, 14	—	8	9	10	11	12	12		1	1
15, 16	—	—	8	9	10	11	12	12	1	1
15, 16 (с нуля) 17, 18, сессия	л/р не принимаются — пишите КрЭВМ или выполняйте индивидуальное задание									

раторные работы Лi характеризуют *процесс* изучения ОЭВМ [и Асм] в течение семестра, поэтому их можно получить, только сдавая Лi **по графику:**

— начиная с первого или второго лабораторного занятия (Л1 и Л2 могут выполняться даже до первой лекции, они не требует специфических знаний);
— по одной работе на каждом занятии, максимум — две на одном занятии;
— с минимальным опозданием.

По результатам защиты лабораторной работы Лi выставляются:

а) на 1–16 неделях по графику (раздел РЛ.4):
   — базовые баллы $0 \leqslant s \leqslant s_{\max}$ в графу «Лi»;
   — бонусные баллы $0 \leqslant s^{\text{bonus}} \leqslant s^{\text{bonus}}_{\max}$ в «Бонус (л/р)» (раздел РЛ.3);
   в графу «Лi» выставляется сумма $s + s^{\text{bonus}}$;
б) на 1–16 неделях в составе комплекта из трёх и более работ (раздел РЛ.6):
   — баллы $0 \leqslant s \leqslant 1$ в графе «Лi»;
   — нет бонусных баллов: $s^{\text{bonus}} = 0$;
в) 17–18 недели, сессия — лабораторные работы не принимаются (раздел РЛ.7).

Базовые s и бонусные s^{bonus} баллы выставляются преподавателем:
— не за программу или отчёт, а за **защиту** лабораторной работы;
— на его усмотрение: если описанных явно бонусов/штрафов недостаточно для адекватного оценивания конкретной защиты — вводятся дополнительные;
— сообразно **качеству** защиты и работы: s при $0 \leqslant s \leqslant s_{\max}$ может быть и 1, и 0.

Из баллов Лi, просроченной более чем на две недели без уважительной причины, вычитается величина опоздания (таблица РЛ.1). Просроченная на $\Delta t \geqslant 12$ недель даже по уважительной причине — не может быть сдана вообще.

РЛ.1. Командная работа

Состав команды: либо один студент, либо 2–3 сидящих рядом студента. Задания для $n \in \{1, 2\}$ одинаковы; для $n \geqslant 3$ есть дополнительные (для троек). Независимо от количества участников n, команда выполняет **один вариант** работы — по номеру команды (№), один отчёт и **одну защиту,** получает **одну оценку.**

Распределение номеров команд (№) обычно происходит на первом лабораторном занятии. Группа может поделиться на команды и распределить уникальные (в пределах группы) номера заранее — тогда преподавателю лабораторных работ на первом занятии необходимо предоставить список команд, отсортированный по возрастанию №, а внутри каждой команды — по алфавиту. Отдельно взятая команда не может сама выбрать себе №, так как это не гарантирует уникальность — необходимо утверждение у преподавателя (таблица РЛ.2).

Если студент один в команде — он может использовать №, равный номеру пропуска. За самовольное присвоение №1 без согласования с группой и преподавателем — штраф -2 балла к первой по графику лабораторной работе.

Каждый из соавторов должен уметь **объяснить все результаты** лабораторной работы и модифицировать свою часть кода. Если студенты выполняют задания

Командная работа и сложные ситуации

Таблица РЛ.2

	Нормально	**Недопустимо**
Изменение состава команды (постоянное)	Один-два раза (не сработались) — баллы за сданные Лi сохраняются. № команд д. б. уникальным — утвердите № у преподавателя, прежде чем делать новый вариант	Постоянно делиться/объединяться синхронно с наличием/отсутствием заданий для троек — штраф ко всем Лi по -2 балла к каждой
Защита Лi в неполном составе (подготовка Лi — всегда в полном)	Отсутствовать на защите некоторых Лi — защищаться отдельно не нужно. Тройка выполняет задания для троек, даже если на защите один студент	Отсутствовать систематически (и не сообщать о сложностях): у пропавшего на полсеместра без предупреждения — аннулируются баллы за все сданные без него Лi
Исключение из команды на одну Лi (временное)	Делать и защищать с командой Л$(i+1)$ и следующие. На тему Лi — получить и выполнить доп. задание или смириться с $s(\text{Л}i) = 0$	Защищать ту же самую Лi, которую уже защитила команда — в лучшем случае будет $s(\text{Л}i) = 1$
Постоянное исключение из команды — изменение состава; исключённый меняет №		
Разные мнения на защите Лi	Придти к согласию, затем отвечать. Не удалось договориться — скажите об этом и изложите все мнения последовательно	Излагать все мнения одновременно — ни один ответ не засчитывается; озлобленность преподавателя растёт
Разные мнения при подготовке Лi	Спросить у преподавателя и только потом защищаться	Всё остальное — штрафуется либо как недоделка, либо как жульничество

не совместно, а распределяют их между собой, то после того, как решения будут начерно готовы, каждый студент должен:

а) изучить все решения, выполненные другими соавторами;

б) спросить у автора каждого решения всё то, что ему непонятно в его решении — а автор должен объяснить;

в) исправить ошибки в этом решении, если заметит их.

Для лабораторных работ, где можно использовать несколько равнодопустимых альтернатив (`printf()/cout`, AVX/SSE и т. п.) — для всех заданий одной команды должна использоваться одна и та же (или везде AVX, или везде SSE).

Студента, который не только не пришёл на защиту Лi, но и не помогал её делать, присутствующие соавторы могут временно (на Лi) или постоянно исключить из команды (таблица РЛ.2).

Команды из $n \geqslant 4$ студентов запрещены на ВЦ МИЭТ; в не-ВЦ классах (и задания для $n \geqslant 4$ сверх заданий для троек) — на усмотрение преподавателя.

РЛ.2. Требования к выполнению лабораторных работ

Лабораторные работы могут быть подготовлены дома и защищаться на ВЦ МИЭТ. Перед защитой на лабораторном занятии необходимо разрешить все вопросы, возникшие в процессе подготовки.

Язык программирования

Задания всех лабораторных работ, **если не указан язык, выполняются на ассемблере,** в виде вставок в программу на языке C/C++ либо отдельных модулей.

Компилятор, отладчик, IDE, ОС

Группе НБ-31 можно использовать любые средства разработки на C/C++, так как у них нет обязательных лабораторных работ с элементами ассемблера.

Всем остальным необходимо использовать **компилятор из коллекции GCC,** отладчик GDB и любую IDE, которая их поддерживает: Qt Creator, TheIDE (Ultimate++). Можно работать без IDE, выполняя сборку и запуск из консоли.

Допустимые платформы указаны в таблице РЛ.3. Другие средства, в частности,

Доступные для выполнения лабораторных работ платформы

Таблица РЛ.3

Платформа	Процессор	ОС	Разр-ть	Коллекция компиляторов	IDE
godbolt.org, onlinegdb.com, jdoodle.com	x86/amd64	GNU/Linux	64 бита	GCC	встроены в сайты
ВЦ МИЭТ	x86/amd64	MS Windows	64 бита	порт GCC под ОС MS Windows (MinGW)	Qt Creator или без IDE
личный ПК студента	x86/amd64	известные студенту		GCC	любая или без IDE

IDE Microsoft Visual Studio, не могут быть использованы там, где иное не сказано явно. Экзотические процессоры, ОС и компиляторы — обсуждаемы, но обычно это курсовая, а не лабораторные работы.

Недопустимо выполнять лабораторные работы на платформе, для которой студент не может назвать процессор/ОС/разрядность/компилятор.

Отчёт

Дистанционные группы оформляют отчёт в любом случае. Для очных групп:

— если Лi сделана на занятии — защищайте её сразу, **не оформляя отчёта;**

— если Лi готовится дома заранее — параллельно оформляйте отчёт.

Формат — OpenDocument или PDF. Заголовок отчёта должен включать имя группы и ФИО авторов, тему работы, а также — для каждого задания:

— номер и текст задания;
— номер и текст варианта (если есть);
— **платформа,** на которой делается задание, в частности — ОС и разрядность;
— ключевые фрагменты программного кода;
— ключевые пояснения, которые вы боитесь забыть;
— ссылки на использованные справочные материалы и пояснения, что и для чего вы там искали, где применили найденное;
— результат выполнения задания: результаты измерений с комментариями.

Вместе с отчётом должен предоставляться полный текст программ, готовый к сборке и запуску, поэтому копировать их ещё и в отчёт не нужно.

РЛ.3. Необязательные (бонусные) задания (1–16 недели)

Задания, отмеченные как **«Бонус»**, *необязательны.* На 1–16 неделях за их выполнение начисляются дополнительные баллы $0 \leqslant s^{\text{bonus}} \leqslant s^{\text{bonus}}_{\max}$ (максимальное количество $s^{\text{bonus}}_{\max}$ указано в тексте задания и не уменьшается со временем: s^{bonus} зависит только от качества исполнения и защиты, но не от времени сдачи).

Баллы s^{bonus} добавляются к графе «Бонус (л/р)» «Лi». Бонусные задания могут быть сданы либо одновременно с соответствующей лабораторной работой, либо (при условии, что это не помешает преподавателю принимать у других студентов обязательные задания) после неё.

При защите нескольких лабораторных работ сразу (раздел РЛ.6) бонусные баллы не начисляются ($s^{\text{bonus}} = 0$), бонусные задания формируют базовые баллы s наравне с обязательными.

РЛ.4. Защита лабораторной работы (1–16 недели)

До начала защиты лабораторной работы Лi команда должна задать преподавателю подготовленные во время выполнения Лi вопросы.

Если у вас вопрос — говорите «у меня вопрос», если планируете защищаться — «хочу сдать»/«хочу защитить работу». Запрос «посмотрите лабораторную работу» будет по возможности игнорироваться, так как его цель не ясна.

Дополнительные вопросы на защите задаются всем. **Отчёта для оценивания недостаточно.** По итогам защиты ставятся баллы $s + s^{\text{bonus}}$.

Защищать не по порядку — можно, $\max(s_{\max})$ **корректируются**

Если какую-то Лi не удалось выполнить даже частично:

— сформулируйте вопросы к преподавателю о том, что вызвало проблемы в Лi;
— если Л$(i+1)$ сделать получается — делайте и защищайте;

— на том же занятии задавайте вопросы про Лi;
$\max(s_{\max})$ считается по фактическому порядку защит, а не по номерам $i/(i+1)$.

Не выполненная, но проработанная Лi + дополнительное задание = «удовл»

Если ни одно задание лабораторной работы Лi не выполнено, но:
— по всем обязательным заданиям Лi подготовлены вопросы к преподавателю;
— сохранены все неудачные попытки выполнения заданий с комментариями, чем именно они плохи (не собирается — с какими сообщениями компилятора? Не приводит к нужному результату — а к какому приводит?);
— после разрешения вопросов команда выполнила по указанию преподавателя либо задание Лi, либо дополнительное задание, не покидая аудиторию;
то Лi засчитывается на $1 \leqslant s \leqslant \frac{1}{2}\max(s_{\max})$, без бонусных баллов ($s^{\text{bonus}} = 0$).

Чего делать не стоит («неуд», то есть 0 баллов)

Однозначно на «неуд» (на $s = s^{\text{bonus}} = 0$ баллов) оценивается работа Лi, если вопросов перед защитой не задавалось, зато на самой защите:
— звучат ответы «делал давно, не помню», «взял в Интернете, не помню где»;
— есть хотя бы одна программа (в том числе отлично выполненная), в которой студенты ничего не могут объяснить и/или изменить;
— в отчёте есть хотя бы одно место, которое студенты не могут разъяснить и пересказать своими словами;
— хотя бы в одном задании Лi (основном или бонусном) есть признаки плагиата (несоответствие варианту, несоответствие программы заявленным ОС и разрядности, использование заданий предыдущего года и ранее, и т. п.);
как только хоть что-то из этого проявилось — защита завершается. Преподаватель может как дать команде дополнительное задание на $1 \leqslant s \leqslant 2$ (но $s^{\text{bonus}} = 0$), так и сразу отправить на пересдачу (на следующем занятии, тому же преподавателю).

РЛ.5. Дистанционная защита для дистанционных групп

Дистанционные группы защищают лабораторные работы, используя:
— синхронную связь с демонстрацией экрана (телемост=видеоконференция) — регламент и требования к отчёту полностью аналогичны очной защите;
— или асинхронную (почта, домашние задания ОРИОКС) — отчёт необходим, но оценивается защита; **«прислать лабы на почту»** без защиты **= «неуд».**
«На одном занятии» дистанционным группам читать как «за одну неделю из 1–16», «на следующем занятии» — как «через две недели».

Студенты очных групп, согласно требованиям МИЭТ, должны учиться очно. В виде исключения преподаватель лабораторных работ может принять у кого-то защиту дистанционно, но не обязан это делать.

РЛ.6. Защита двух, трёх или более лабораторных работ (1–16 недели)

Две лабораторные работы на одном занятии могут быть оценены раздельно (как описано в разделе РЛ.4); если есть очередь на защиту — после защиты первой команда отправляется в конец очереди.

Комплект (три и более работы сразу) — оценивается выборочно, без бонусов

Если команда приносит на занятие три и более лабораторных работы сразу — защита происходит выборочно: преподаватель проверяет несколько произвольно выбранных (из всех выполненных, в том числе бонусных) заданий разных работ, задаёт вопросы и даёт задания по произвольным темам.

В каждую графу «Лi» выставляются одинаковые баллы:

$$\begin{cases} 0 \leqslant s \leqslant 3, & \text{если в комплекте 3–4 работы (до 12 недели),} \\ 0 \leqslant s \leqslant 1, & \text{если работ} \geqslant 5 \text{ или неделя} \geqslant 13. \end{cases} \qquad \text{(Л.1)}$$

Бонусные баллы, которые рассчитываются для одной лабораторной работы, при защите комплекта не рассчитываются и не добавляются: $s^{\text{bonus}} = 0$.

Третья лабораторная работа без предупреждения — ждёт две недели

Если команда уже получила баллы за две работы (или за комплект из 3–4) на занятии — защита следующей не ранее следующего занятия или через две недели; регламент — по фактической дате защиты, а не по первой заявке.

РЛ.7. Лабораторные работы на 17–18 неделях и в сессию

Раздельная защита лабораторных работ (с вычислением баллов, бонусов и штрафов к каждой работе отдельно) на 17–18 неделях и в сессию **невозможна.**

Если, вопреки требованиям регламента, команда принесёт на 17 неделе и позже любое количество лабораторных работ (от одной и более), преподаватель может:

— либо оценить их как один комплект: выборочно, на $0 \leqslant s \leqslant 1$ и $s^{\text{bonus}} = 0$;

— либо сразу послать такую команду писать Кр0 (ЭВМ).

Исключение — если команда сдавала лабораторные работы весь семестр, начиная с первого занятия, по графику и досдаёт тому же преподавателю *одну* последнюю работу — она оценивается как сделанная на 16 неделе.

Лабораторная работа 1 (0001 = 1)
Ввод-вывод при помощи libc

Засчитывается только весной 2025 г.; актуальная версия в https://gitlab.com/illinc/gnu-asm

Цель работы: изучить размеры стандартных типов C/C++ на выбранной платформе; научиться использовать функции ввода-вывода libc.

Все задания Л1 выполняются на чистом C/C++, без использования ассемблера. Как и для всего курса — если иное не указано явно, компилятор должен быть из коллекции GCC (среда Microsoft Visual Studio недопустима), подробнее в разделе «Компилятор, IDE, отладчик» регламента, РЛ.2.

Штраф за одно пропущенное обязательное задание -2 балла. Дополнительный бонус за сдачу на первом занятии $+4$ балла.

Л1.1. Задание на лабораторную работу

Очистка буфера после некорректного ввода не требуется ни в каком задании Л1

Задание Л1.№1. Напечатайте (выведите на стандартный вывод) группу, номер и состав команды при помощи функции $puts()$ библиотеки libc.

Что должна содержать выводимая строка, чтобы задание Л1.№1 было выполнено *одним* вызовом $puts()$; и при этом группа, номер и ФИО участников печатались на *разных* строках?

При работе в ОС MS Windows возможны проблемы с кодировкой русского языка. Если они возникли — используйте транслит или любые доступные вам способы настройки.

Задание Л1.№2. Укажите для платформы, где выполняется работа:
- ОС и разрядность ОС;
- компилятор (должен относиться к коллекции GCC/MinGW) и его версию;
- разрядность сборки (собираемая программа может работать в 32-битном режиме даже под 64-битной ОС — в режиме совместимости);
- архитектуру процессора, назначение платформы.

Компьютер с процессором x86/amd64 под управлением GNU/Linux, BSD (в том числе Mac OS X) или MS Windows — платформа общего назначения.

При помощи оператора `sizeof` языка C/C++ выясните, сколько байтов занимают на выбранной платформе переменные следующих типов: `char, char*, bool, wchar_t, short, unsigned short, int, long, long long, float, double, double*, long double, size_t, ptrdiff_t, void*`.

Штраф -2 балла, если выводятся только числа, без пояснений, и непонятно, где размер какого типа. **Бонус $+2$ балла,** если при помощи макроса пояснения выводятся так, что в коде каждое имя типа в Л1.№2 встречается единожды.

Соответствуют ли размеры заявленным разрядности, ОС и компилятору?

Для НБ и индивидуальщиков, которые уже использовали sizeof в курсе ОТИК: обратите внимание, что здесь другой набор типов и другое задание. Вы можете использовать разработанные ранее макросы и шаблоны, но измерения необходимо выполнить заново и заново ответить на вопросы.

Задание Л1.№3. Создайте и инициализируйте заданными значениями x шесть массивов $M*$, каждый из $N = 5$ чисел (типы указаны для 32/64-битных режимов x86/amd64, для иных платформ выбирайте по результатам задания Л1.№2):

— Mb из 8-битных целых чисел $char/unsigned\ char$ ($x =$ `0xED`);
— Ms из 16-битных целых чисел $short/unsigned\ short$ ($x =$ `0xFADE`);
— Ml из 32-битных целых чисел $int/unsigned\ int$ ($x =$ `0x ADE1 A1DA`);
— Mq из 64-битных целых чисел $long\ long/unsigned\ long\ long$ ($x =$ `0x C1A5 51F1 AB1E`);
— Mfs из 32-битных чисел с плавающей запятой $float$ (x из таблицы Л1.1);
— Mfl из 64-битных чисел с плавающей запятой $double$ (x из таблицы Л1.1).

Варианты начальных значений элементов с плавающей запятой

Таблица Л1.1

$(№ - 1)\%2 + 1$	**Вариант**
1	$x = \frac{8}{3}$
2	$x = -\frac{2}{7}$

Не используйте типы фиксированной разрядности $int*\_t/uint*\_t$, так как для них не существует модификаторов размера форматных полей $printf()/scanf()$.

Напечатайте все массивы $M*$ при помощи функции libc $printf()$ (раздел **??**).

1. Целочисленные Mb, Ms, Ml, Mq четырежды (каждый — четырежды):
   а) в шестнадцатеричном представлении (формат X);
   б) в двоичном представлении (формат b);
   в) в десятичном беззнаковом представлении (формат u);
   г) в десятичном знаковом представлении (формат d).
2. С плавающей запятой — Mfs и Mfl— трижды:
   а) в шестнадцатеричном экспоненциальном представлении (формат A);
   б) в десятичном экспоненциальном представлении (формат e);
   в) в представлении с десятичной запятой (формат f).

Элементы массива разделяйте пробелами.

Штраф -1 балл за пару тип+формат, где не указан **модификатор размера** (раздел **??**) для типов, отличных от $int/unsigned/float$.

Если компилятор устарел и не поддерживает форматов b, A или модификатора размера hh, отметьте это в отчёте и реализуйте то, что поддерживается.

Напечатайте 8-битный Mb ещё пятый раз: в символьной форме (формат c). Так как тип по умолчанию для формата c — это именно $char/unsigned\ char$, модификатор размера не нужен.

Укажите для какой либо пары массив+формат **ширину поля вывода,** добавив некоторое число w между символом $\%$ и модификатором размера (если модификатора размера нет — между $\%$ и форматом). Пусть исходная ширина выводимого числа w_0 знакомест: как изменится вывод при $w \leqslant w_0$, как при $w > w_0$?

Установите $w > w_0$. Добавьте между $\%$ и w знак «$-$» (минус). Что изменилось?

Дополните шестнадцатеричное представление всех целочисленных массивов (формат X) ведущими нулями до необходимого количества цифр. Для этого:

а) рассчитайте необходимое количество шестнадцатеричных цифр w — по две на байт, так, для $short/unsigned\,short$ цифр $w = 2{\cdot}\mathrm{sizeof}(short) = 2{\cdot}2 = 4$;

б) укажите ширину поля вывода w и перед ней, но после $\%$ — символ 0, так, для $short/unsigned\ short$ форматное поле $\%hX$ замените на $\%04hX$.

Как изменился вывод? Дополните двоичное представление (формат b) аналогично ведущими нулями до количества бит $w = 8 \cdot \mathrm{sizeof}(M[i])$.

Дополните знаковое десятичное представление целочисленных массивов (формат d) знаком «$+$» перед положительными числами, для чего вставьте «$+$» между $\%$ и шириной поля вывода.

Для одного из массивов с плавающей запятой и форматов A, e, f и всех форматов его вывода задайте **точность** в две цифры после запятой, для чего добавьте .2 между шириной поля вывода и модификатором размера. Что изменилось?

Штраф -1 балл, если вместо именованной константы N здесь и/или позже используется литерал 5.

Бонус $+1$ балл, если вывод массива во всех формах описан как функция и в последующих заданиях используется вызов этой функции, а не копирование и вставка; **$+2$ балла,** если эта функция описана как единый для всех массивов шаблон и принимает тип как параметр шаблона, а адрес начала M, длину N и форматы с модификатором размера как параметры функции; **$+3$ балла,** если вывод описан как единый для всех массивов макрос с соответствующими параметрами.

Задание Л1.№4. Для каждого массива M из всех созданных введите с клавиатуры новое значение элемента $M[i]$, $i = 2$ при помощи функции libc $scanf()$.

Проанализировав возвращённое $scanf()$ значение, определите корректность ввода; при необходимости отобразите сообщение об ошибке при помощи функции libc $puts()$.

Выведите массивы на экран до и после ввода, каждый раз — во всех форматах, описанных в Л1.№3; убедитесь, что элемент $M[i]$ приобрёл ожидаемое значение, а другие элементы массива не изменились (если изменились — проверьте, верно ли вы указали модификатор размера).

Штраф -1 балл, если массив после ввода нового значения $M[i]$ выводится только в одном формате (а если для целочисленных это ещё и не X, то $-$**2 балла).**

В данном задании необходимо передать функции $scanf()$ адрес $M[i]$, а не промежуточной переменной — иначе нет смысла контролировать значение соседних элементов массива. **Штраф -2 балла,** если используется промежуточная переменная для ввода-вывода.

Задание Л1.№5. Для одного из массивов M (по варианту согласно таблице Л1.2) напечатайте *адреса* (формат p — указатели):

Варианты массива M

Таблица Л1.2

$(№ - 1)\%5 + 1$	**Вариант**
1	Ms
2	Ml
3	Mq
4	Mfs
5	Mfl

— начала массива — M;
— $M[0]$, начального (нулевого) элемента массива — $\&(M[0])$;
— $M[1]$, следующего за $M[0]$ элемента массива — $\&(M[1])$.

Сравните полученные значения между собой и с размером элемента массива M. Как расположены в памяти элементы массива?

Создайте статическую матрицу MM из R строк и N столбцов; элементы MM того же типа, что и элементы M. Напечатайте адреса элементов $MM[0][0]$, $MM[0][1]$, $MM[1][0]$, $MM[1][1]$. Как расположены в памяти элементы матрицы?

Как можно воспроизвести эту структуру на динамическом массиве M? Сколько памяти необходимо выделить? Как рассчитать индекс idx в M по номерам строки i и столбца j в MM? Требуется именно воспроизвести структуру матрицы в памяти, а не сымитировать синтаксис обращения $MM[i][j]$.

Задание Л1.№6. Для одного из массивов M (по варианту согласно таблице Л1.2) введите с клавиатуры новое значение всех пяти элементов при помощи одного вызова функции libc $scanf()$.

Проанализировав возвращённое $scanf()$ значение, определите корректность ввода; при необходимости отобразите сообщение о количестве введённых и не введённых элементов.

Выведите массив на экран до и после ввода; убедитесь, что количество изменившихся элементов соответствует ожиданиям.

Задание Л1.№7. Бонус +2 балла для пар, обязательное для троек. Введите с клавиатуры (каждую строку — одним вызовом $scanf()$):

а) слово (строку без пробелов) $s1$ (формат s без модификаторов);

б) слово $s2$ таким образом, чтобы принимающий его буфер гарантированно не переполнился: если буфер длины k — вводить не более $k-1$ символов (ширина поля ввода задаётся аналогично ширине поля вывода);

в) строку, возможно, содержащую пробелы $s3$ (формат `[]` — регулярное выражение Perl).

Выведите на экран при помощи функций libc строки «***$s1$***», «***$s2$***», «***$s3$***» (между звёздочками должна быть введённые строки, а не литералы s1-s3) и убедитесь, что ввод корректен.

Л1.2. Дополнительные бонусные и штрафные баллы

—2 балла за каждое задание, где не печатаются исходные данные.
—3 балла за утечку памяти (выделенные, но не освобождённые блоки).

Лабораторная работа 2 (0010 = 2)

Представление данных в ЭВМ

Засчитывается только весной 2025 г.; актуальная версия в https://gitlab.com/illinc/gnu-asm

Цель работы: сопоставить размеры стандартных типов C/C++ на различных платформах; изучить форматы представления чисел на примере выбранной платформы.

Все задания Л2 выполняются на чистом C/C++, без использования ассемблера.

Штраф за одно пропущенное обязательное задание -2 балла.

Л2.1. Задание на лабораторную работу

Рекомендации
Во всех заданиях Л2 рекомендуется вывод с помощью $printf()$:
а) библиотека libc, в том числе $printf()$, доступна и в GNU Assembler;
б) формат вывода $printf()$ компактен и влияет только на одно значение.
Не рекомендуется вывод в потоки (доступен только в C++), так как:
а) вывод в заданном формате (а не по умолчанию) объёмен и запутан;
б) большинство манипуляторов не прекращают своё действие до отмены, а в комбинации с другими дают неочевидные эффекты;
в) в заданиях Л2.№2 и Л2.№3 придётся выполнять лишние преобразования, так как по умолчанию в поток и $char$ / $unsigned\ char$, и $int8\_t$ / $uint8\_t$, и указатели на них выводятся как символы/строки.
Тем не менее, вывод в потоки не штрафуется. Часть заданий различается для $printf()$ и для потоков из-за принципиально разного подхода к выводу.

Задание Л2.№1. Бонус +2 балла. Выполните измерения согласно заданию Л1.№2 на платформах, доступных на ВЦ (таблица Л2.1).

Связка GNU/Linux 64 + GCC 64 широко используется в онлайн-компиляторах. На `godbolt.org` (ОС GNU/Linux 64) доступны сборка компиляторами GCC 64, clang 64 и ICC (Intel C++ Compiler) 64 с возможностью запуска; а также сборка без запуска для множества других компиляторов, в том числе для не-x86 процессоров.

ОС MS Windows 64 и компиляторы GCC и Microsoft доступны на ВЦ локально (для дистанционных занятий — на терминале ВЦ).

Не возбраняется использование инструментов, установленных дома.

Размеры каких типов одинаковы для наиболее популярных платформ? Какие характеристики типов приведены в стандарте C, а какие — могут различаться?

Платформы для измерения

Таблица Л2.1

Процессор	ОС	Компилятор	разрядность сборки
x86/amd64	GNU/Linux 64	GCC	64
x86/amd64	GNU/Linux 64	clang	64
x86/amd64	GNU/Linux 64	Intel	64
x86/amd64	MS Windows 64	GCC (MinGW)	64
x86/amd64	MS Windows 64	Microsoft	64
x86/amd64	MS Windows 64	Microsoft	32

Штраф −1 балл за платформу таблицы Л2.1, если в аудитории она доступна[1], а данных по ней нет.

Бонус +2 балла за платформу. При подготовке к работе выполните измерения на платформе, отсутствующей в таблице Л2.1. Укажите ОС, компилятор, режим (разрядность) сборки, архитектуру процессора, назначение платформы — без этих сведений баллы не начисляются.

Задание Л2.№2. Разработайте функцию void $viewPointer(\text{void } *p)$, которая принимает нетипизированный указатель p, преобразует его в типизированные:

а) `char *p1 = reinterpret_cast<char *>(p);`

б) `unsigned short *p2 = reinterpret_cast<unsigned short *>(p);`

в) `double *p3 = reinterpret_cast<double *>(p);`

и печатает *адреса* $p, p1, p2, p3$. Убедитесь, что $p, p1, p2, p3$ — один и тот же адрес, то есть что `reinterpret_cast` не меняет преобразуемого указателя и, следовательно, может быть использован для интерпретации одной и той же области памяти как значений различных типов.

Дополните $viewPointer()$ печатью смежных с p адресов: $p1+1$, $p2+1$, $p3+1$. Сопоставьте разницу между p_i и p_i+1 в байтах для типизированного указателя $T *p_i$ с размером типа T. Проверьте, позволяют ли текущие настройки компилятора рассчитать $p+1$. Если да — какова разница между p и $p+1$ в байтах?

Разработайте функцию void $printPointer(\text{void } *p)$, которая преобразует p в типизированные $p1, p2, p3$ аналогично $viewPointer()$ и печатает *значения* соответствующих типов по адресу p: $*p1, *p2, *p3$.

Дополните $printPointer()$ печатью значений по смежным с p адресам: $*(p1+1)$, $*(p2+1), *(p3+1)$. Все целые числа выводите в шестнадцатеричном виде.

Проверьте работу функций $viewPointer()$ и $printPointer()$ на значениях `0x1122334455667788` (*long long*), `"0123456789abcdef"` (*char*[]).

[1] если платформа недоступна в лабораторной аудитории либо её убрали с ВЦ вообще (в 2021 году с ВЦ исчезла связка MS Windows 64 + GCC 32), штраф не начисляется

Можно ли в C/C++ разыменовать нетипизированный p (получить $*p$)?

Задание Л2.№3. Разработайте функцию void $printDump$(void $*p$, size_t N), которая преобразует нетипизированный указатель p в указатель $p1$ на байт ($char/unsigned\ char$) и печатает шестнадцатеричные значения N байтов, начиная с этого адреса: $*p1$, $*(p1+1)$, ... $*\Big(p1+(N-1)\Big)$ — шестнадцатеричный дамп памяти. Каждый байт должен выводиться в виде двух шестнадцатеричных цифр, байты разделяются пробелом (спецификатор «$\%02hhX$ »).

Исследуйте при помощи $printDump()$, как хранятся в памяти компьютера:

а) целое число x (типа int; таблица Л2.2); по результату исследования определите порядок следования байтов в словах для вашего процессора:
   — прямой (младший байт по младшему адресу, порядок Intel, Little-Endian, от младшего к старшему);
   — обратный (младший байт по старшему адресу, порядок Motorola, Big-Endian, от старшего к младшему);

б) число с плавающей запятой x (типа $double$; таблица Л2.2).

в) строки `"abcdef"` и `"абвгде"` (массив из $char$; при выборе N учитывайте всю длину строки, а не только видимые буквы);

г) «широкие» строки `L"abcdef"` и `L"абвгде"` (массив из $wchar\_t$; при выборе N учитывайте всю длину строки).

Длину строки *в элементах* $char/wchar\_t$ без завершающего нуля можно получить при помощи функций $strlen()/wcslen()$; после чего, зная размер элемента и то, что завершающий ноль занимает один элемент, можно вычислить размер строки в байтах. Если строки хранятся в статическом массиве без явного указания размера, инициализированном строкой при создании — размер строки равен размеру массива (можно определить sizeof).

На MS Windows возможна (если файл исходного кода сохранён в однобайтовой кодировке `windows-1251`) ситуация, когда литерал `L"абвгде"` не воспринимается компилятором как корректная широкая строка.

Бонус +2 балла. Выполните в) и г) на всех платформах таблицы Л2.1.

Бонус +2 балла за платформу. Выполните измерения на платформе, где архитектура процессора отлична от x86/amd64.

Задание Л2.№4. Напечатайте, используя <climits> (limits.h) C или <limits> C++, минимальные и максимальные значения 8-битных целых типов $char$ и $unsigned\ char$, 16-битных $short$ и $unsigned\ short$, 32-битных int и $unsigned$, 64-битных $long\ long$ и $unsigned\ long\ long$.

1. Сколько различных значений может принимать переменная беззнакового N-битного типа? Знакового N-битного?
   Связано ли это как-то со значением N и как именно?

2. Каждое ли целое число $x \in [0, 2^N)$ имеет своё N-битное представление? Всякая ли последовательность ξ из N битов может быть рассмотрена как целое $x \in [0, 2^N)$? Взаимно однозначно ли это соответствие?
3. Каждое ли целое число $x \in [-2^{N-1}, 2^{N-1})$ имеет своё N-битное представление? Всякая ли последовательность ξ из N битов может быть рассмотрена как целое $x \in [-2^{N-1}, 2^{N-1})$? Взаимно однозначно ли это соответствие?

Напечатайте минимальные и максимальные значения $\min$ и $\max$ 32-битного типа с плавающей запятой $float$, 64-битного $double$.

1. Каждое ли вещественное число $x \in [\min, \max]$ имеет своё представление с плавающей запятой стандарта IEEE 754 ($float/double$ ЭВМ)?
2. Всякая ли последовательность ξ из N битов $\Big(N \in \{32, 64\}\Big)$ может быть рассмотрена как N-битное *значение с плавающей запятой*? Всегда ли это значение — *число?*
3. Каких чисел больше: 32-битных целых ($int/unsigned$) или 32-битных с плавающей запятой ($float$)? 64-битных целых ($long\ long/unsigned\ long\ long$) или 64-битных с плавающей запятой ($double$)? Значения $\pm\infty$ числами не являются, нечисла nan — тем более.

Для НБ и индивидуальщиков, у которых уже есть $\min$ и $\max$, а также $print16()$, $print32()$ и $print64()$ из курса ОТИК: вы можете использовать разработанные ранее программы, но на вопросы необходимо ответить заново.
«Просто перезачесть» задания Л2.№4–Л2.№7 нельзя.

Задание Л2.№5. Исследуйте внутреннее представление 16-битных чисел. Для этого на C/C++ разработайте $\mathrm{void}\ print16(\mathrm{void}\ *p)$ (одну: С или П).

С-Л2.№5 для вывода при помощи стандартной библиотеки C (рекомендуется): функция $print16()$ интерпретирует p как адрес 16-битного целого числа x типа $short/unsigned\ short$ и печатает x при помощи $printf()$:

С-а) в шестнадцатеричном представлении;
С-б) в двоичном представлении: если не поддерживается формат b — напечатать биты `(x & (1 << i)) != 0`, от старшего $i = 15$ и до $i = 0$;
С-в) в десятичном беззнаковом представлении;
С-г) в десятичном знаковом представлении.

Необходимый вид вывода $print16()$ для значений `13` и `0x8000` по адресу p:

```
000D 0000000000001101     13     +13
8000 1000000000000000 32768 -32768
```

Для $print16()$, а также последующих $print32()$, $print64()$, и любых x и y младшая цифра любого представления y всегда должна печататься под младшей цифрой соответствующего представления x.

П-Л2.№5 для вывода в потоки (крайне не рекомендуется, но и не штрафуется). Так как вывод в поток различает $short$ и $unsigned\ short$, функция $print16()$ интерпретирует адрес p дважды:

— как адрес 16-битного беззнакового целого ux типа $unsigned\ short$;

— и как адрес 16-битного знакового целого sx типа $short$;

и печатает оба ux и sx во всех доступных представлениях:

П-а) ux в шестнадцатеричном представлении;

П-б) ux в двоичном представлении (шаблон `std::bitset<N>`);

П-в) ux в десятичном представлении;

П-г) sx в шестнадцатеричном представлении;

П-д) sx в двоичном представлении (шаблон `std::bitset<N>`);

П-е) sx в десятичном представлении.

Не забывайте, что манипуляторы dec/hex действуют на все числа до отмены, а $setw(\text{int } w)$ — только на одно следующее.

Убедитесь в процессе исследования, что (П-б) и (П-д) — одно и то же двоичное представление; (П-а) и (П-г) — одно и то же шестнадцатеричное представление. Если у вас (П-б)≠(П-д) или (П-а)≠(П-г) — ищите ошибку.

Сократите вывод П-Л2.№5 до вида, аналогичного С-Л2.№5.

Штраф −1 балл, если версия задания С-Л2.№5 реализована при помощи потоков или П-Л2.№5 при помощи стандартной библиотеки С.

Штраф −1 балл, если вывод $print16()$ занимает более одной строки.

Исследуйте при помощи $print16()$ 16-битные целочисленные переменные типа $short/unsigned\ short$, принимающие значения:

— минимальное и максимальное целое беззнаковое 16-битные значения;

— минимальное и максимальное целое знаковое 16-битные значения;

— целочисленные x, y, a, b, соответствующие варианту (таблица Л2.2).

Варианты значений

Таблица Л2.2

$(№ - 1)\%2 + 1$	**Вариант**
1	$x = 9, y = -9, a = 1, b = 2, c = 12345678, d = 123456789$
2	$x = 5, y = -5, a = 1, b = 2, c = 12345689, d = 123456891$

Как представляются в памяти ЭВМ знаковые целые значения? Как различаются целочисленные x и $y = -x$?

Как связаны двоичное представление (С-б) и шестнадцатеричное (С-а)? Как по шестнадцатеричному (С-а) записать двоичное для любого выбранного числа?

Задание Л2.№6. Исследуйте внутреннее представление 32-битных чисел — целых $int/unsigned$ и с плавающей запятой $float$. Для этого на C/C++ разработайте void $print32$(void $*p$) (также одну— C или П — соответственно Л2.№5).

C-Л2.№6 для стандартной библиотеки C: $print32()$ интерпретирует p дважды:
— как адрес 32-битного целого числа x типа $int/unsigned$;
— как адрес 32-битного числа с плавающей запятой fx типа $float$;
и печатает x в представлениях (C-а)–(C-г), а fx:
C-д) в шестнадцатеричном экспоненциальном представлении;
C-е) в десятичном экспоненциальном представлении;
C-ж) в представлении с десятичной запятой.
Необходимый вид вывода $print32()$ для значений `13` и `0x80000000` по адресу p:

```
0000000D 00000000000000000000000000001101          13          +13
    +0X1.A0P-146     +1.82e-44          +0.00
80000000 10000000000000000000000000000000 2147483648 -2147483648
      -0X0.00P+0     -0.00e+00          -0.00
```

Если ширина терминала позволяет вывести (C-а)–(C-ж) в одну строку — это необходимо сделать; иначе вторая строка должна иметь отступ.

П-Л2.№6 для вывода в потоки: $print32()$ интерпретирует адрес p трижды:
— как адрес 32-битного беззнакового целого ux типа $unsigned$;
— как адрес 32-битного знакового целого sx типа int;
— как адрес 32-битного числа с плавающей запятой fx типа $float$;
и печатает ux и sx в представлениях, аналогичных (C-а)–(C-г) (дублирующиеся — в одном экземпляре), а fx — в представлениях, аналогичных (C-д)–(C-ж). Обратите внимание, что манипулятор $setprecision$(int n) действует до отмены; $fixed/scientific/hexfloat$ — также до отмены, а также проявляют себя по-разному в зависимости от dec/hex.

Исследуйте при помощи $print32()$ 32-битные переменные — целочисленные типа $int/unsigned$ и с плавающей запятой типа $float$:
— минимальное и максимальное целое беззнаковое 32-битные значения;
— минимальное и максимальное целое знаковое 32-битные значения;
— целочисленные x, y, a, b, c, d, соответствующие варианту (таблица Л2.2);
— $float$-значения x, y, a, b, c, d, соответствующие варианту (таблица Л2.2);

Чем различается одно и то же значение (x, y, a, b), записанное в переменную типа $short/unsigned\ short$ и $int/unsigned$? Как различаются граничные (минимальные и максимальные) значения?

Похожа ли структура $int/unsigned$ и $float$? Как представляются значения с плавающей запятой? Как различаются x и $y = -x$ с плавающей запятой?

Получилось ли точно представить d как $float$? Почему? Существует ли такое $float$-значение z, что $d - z = 1$?

Есть ли среди исследованных переменных $float$-числа: нули -0 и $+0$, минимальное денормализованное положительное? Нечисла? Как они записываются?

Как записываются $float$-значения:
— минимальное нормализованное положительное число;
— предыдущее перед $+2$ (максимальное $float$-число x такое, что $x < +2$);
— максимальное нормализованное положительное число;
— $+\infty$?

Запишите предполагаемое битовое представление на бумаге, представьте его шестнадцатерично и инициализируйте им в программе переменную $int/unsigned$. Проверьте $print32()$, что это действительно нужное $float$-значение.

В каком интервале $float$-чисел больше: $[+0, +2)$ или $[+2, +\infty)$?

Задание Л2.№7. Аналогично Л2.№6, разработайте void $print64$(void $*p$), аналогичную $print32()$ для размера 64 бита.

Необходимый вид вывода $print64()$ для значений `13` и `0x8000000000000000`:

```
000000000000000D 0000000000000000000000000000000000000000000000000000000000001101
                                          13                  +13
          +0X0.00P-1022           +6.42e-323                +0.00
8000000000000000 1000000000000000000000000000000000000000000000000000000000000000
                          9223372036854775808 -9223372036854775808
             -0X0.00P+0             -0.00e+00                 -0.00
```

Если ширина терминала позволяет вывести данные в одну или хотя бы в две строки — это необходимо сделать; иначе вторая и третья должна иметь отступ.

Если двоичное представление (С-б) выводится при помощи битовых операций и оно явно некорректно, а в $print32()$ и $print16()$ всё было хорошо — вспомните, что литерал `1` имеет тип int. Единица-литерал типа $long\ long$ — `1ll` или `1LL`.

Исследуйте $print64()$ переменные $long\ long/unsigned\ long\ long$ и $double$:
— целочисленные x, a, соответствующие варианту (таблица Л2.2);
— $double$-значения x, a, соответствующие варианту (таблица Л2.2).

Чем различается одно и то же значение (x или a), записанное в $float$ и $double$? В каком интервале $double$-чисел больше: $[+0, +2)$ или $[+2, +\infty)$?

Штраф -4 балла, если для любой $printWW()$, где $WW \in \{16, 32, 64\}$:
— порядок представлений отличается от указанного;
— представления не соответствует друг другу (а команда, вместо того, чтобы задать вопрос преподавателю — так и сдаёт);
— в частности, двоичное (С-б) явно несообразно шестнадцатеричному (С-а);

или формат вывода отличается от указанного в худшую сторону, в частности:
— количество выводимых цифр двоичного представления (С-б) отлично от количества бит WW в числе;
— а выводимых цифр шестнадцатеричного (С-а) — от количества тетрад $\frac{WW}{4}$;

— младшая цифра не под младшей предыдущего вызова $printWW()$;
— вывод менее компактен, чем указано (делать компактнее при сохранении информативности — можно и нужно).

Далее во всех заданиях **всех лабораторных работ,** если явно не указано иное, все числа (как целые, так и с плавающей запятой) разрядности WW — как **результат,** так и **исходные данные** — должны выводиться соответствующей $printWW \in \{print16(), print32(), print64()\}$.
Штраф −2 балла за каждое число, выведенное только в одном представлении (**−4 балла за число,** если это единственное представление — десятичное).

Задание Л2.№8. Разработайте функцию $c16to32(\text{void } *p)$, которая принимает адрес 16-битной целочисленной переменной, печатает её значение $print16()$, расширяет это значение до 32 бит двумя способами:
— как знаковое ($short \to int$);
— как беззнаковое ($unsigned\ short \to unsigned\ int$).
и для каждого способа печатает результат $print32()$.

Явное расширение в C++ выполняется `static_cast`; неявное — в частности, при присваивании. Если источник и приёмник различаются не только размером, но и знаковостью — неопределённое поведение C/C++.

Проверьте её работу на значениях m и n (таблица Л2.3).

Варианты целочисленных значений

Таблица Л2.3

$(№ - 1)\%3 + 1$	**Вариант**
1	$m = 57, n = -21$
2	$m = 21, n = -37$
3	$m = 37, n = -33$

Задание Л2.№9. Разработайте функцию $ab16(\text{void } *p)$, которая принимает адрес 16-битной целочисленной переменной x и выполняет над её копиями операции (а1)–(б6). Оригинал, лежащий по адресу p, должен оставаться неизменным; для каждой операции исходным является значение x, а не результат предыдущей. Исходное значение x и каждый результат печатается $print16()$.

Сопоставьте результаты (аi) и (бi) — вначале на значении $x = m$, затем $x = n$ (таблица Л2.3). Сколько всего различных операций описано в задании Л2.№9?

а1) беззнаковое умножение на 2;
а2) знаковое умножение на 2;
а3) беззнаковое деление на 2;
а4) знаковое деление на 2;
а5) расчёт остатка от беззнакового деления на 16;
а6) округление вниз до числа, кратного 16 (беззнаковое);
б1) беззнаковый сдвиг влево на 1 бит;
б2) знаковый сдвиг влево на 1 бит;
б3) беззнаковый сдвиг вправо на 1 бит;
б4) знаковый сдвиг вправо на 1 бит;
б5) расчёт $x \ \& \ 15$;
б6) расчёт $x \ \& \ -16$.

Если Л2.№9 реализуется на чистом C/C++, то беззнаковые и знаковые операции (умножение/деление/сдвиги) записываются одним и тем же оператором. Таким образом, адрес p необходимо интерпретировать дважды:

— как адрес 16-битного беззнакового целого ux типа $unsigned\ short$;
— и как адрес 16-битного знакового целого sx типа $short$.

Одни и те же знаки операций C/C++:

— для ux обозначают беззнаковые операции (если операция бинарная, типа $*$ или $/$, то и второй операнд должен быть того же типа $unsigned\ short$);
— для sx — знаковые (бинарные — для sx и другого $short$).

Также можно забежать вперёд и реализовать операции как *ассемблерные вставки* — на уровне ассемблера беззнаковые и знаковые операции выполняются разными командами: $mul/imul$, shl/sal, $div/idiv$, shr/sar.

Л2.2. Дополнительные бонусные и штрафные баллы

−2 балла за каждое задание, где не печатаются исходные данные.
−3 балла за утечку памяти (выделенные, но не освобождённые блоки).

Лабораторная работа 3 (0011 = 3)

Использование ассемблерных вставок в программах на C++. Команды пересылки

Засчитывается только весной 2025 г.; актуальная версия в https://gitlab.com/illinc/gnu-asm

Цель работы: научиться вставлять в программы на языке высокого уровня ассемблерные фрагменты. Ознакомиться с командами пересылки данных.

Все задания Л3 — в виде ассемблерных вставок в программу на C/C++ (и для НБ-3* тоже). Как и для всего курса — если иное не указано явно, компилятор должен быть из коллекции GCC (среда Microsoft Visual Studio недопустима), подробнее в разделе «Компилятор, IDE, отладчик» регламента, РЛ.2.

Штраф за одно пропущенное обязательное задание -1 балл. Штраф за задание с некорректной секцией перезаписываемых элементов (clobbers) $-\frac{1}{2}$ балла.

Л3.1. Задание на лабораторную работу

Для НБ-3* Л3 — пятая по счёту. Перед ней выполняются Л7, ЛВ на языке C++.

Все задания одной лабораторной работы (Л3, а также Л5 и последующих) должны быть выполнены на одной платформе.
«Команды AVX или их SSE-аналоги» везде следует понимать как «команды AVX, если они доступны, и их SSE-аналоги, если расширение AVX на используемой платформе недоступно».

Задание Л3.№1. Как в задании Л1.№3, создайте массивы:
— Ms из 16-битных целых чисел $short/unsigned\ short$;
— Ml из 32-битных целых чисел $int/unsigned\ int$;
— Mq из 64-битных целых чисел $long\ long/unsigned\ long\ long$;
длина N и начальные значения аналогичны Л1.№3.

Реализуйте для каждого массива M вставку, записывающую непосредственное значение 18 в $M[i]$ для заданного $i \in [0, N-2]$ с использованием команды mov, где выражение $M[i]$ является выходным параметром вставки *в памяти*. Так как оба операнда mov здесь не имеют определённого размера (непосредственное значение и память), необходимо указывать для mov суффикс размера: $movw$, $movl$, $movq$.

Каждый M напечатайте до и после изменения в шестнадцатеричном представлении, элементы $M[i]$ разделяются пробелами, разные M — переводом строки.

Задание Л3.№2. Реализуйте для одного из массивов M (по варианту согласно таблице Л3.1) вставку, записывающую непосредственное (-1) в $M[i]$, где адрес начала массива M и индекс i передаются как входные параметры *в РОН*.

Варианты целочисленного массива M

Таблица Л3.1

$(№ - 1)\%3 + 1$	**Вариант**
1	Ms
2	Ml
3	Mq

Используйте компоненты эффективного адреса $(Base, Index, 2^{Scale})$. Разрядность компонент $Base$ и $Index$ должна быть одинаковой, поэтому для переносимости вставки необходимо объявить переменную i не как int (4 байта как для 32-, так и для 64-битного режимов), а как $size\_t$ (размер равен размеру указателя).

Заданный M напечатайте до и после изменения аналогично Л3.№1.

Задание Л3.№3. Реализуйте вставку, записывающую непосредственное значение `0x55` в заданный байт $Mq[i]$ (по варианту согласно таблице Л3.2; младший байт считайте нулевым) с использованием одной команды mov ($movb$) и всех компонент эффективного адреса $Disp(Base, Index, 2^{Scale})$; адрес начала массива Mq и индекс i передаются как входные параметры *в РОН*.

Варианты перезаписываемого байта $Mq[i]$ для Л3.№3

Таблица Л3.2

$(№ - 1)\%5 + 1$	**Вариант**
1	Первый байт после младшего
2	Третий байт
3	Пятый байт
4	Шестой байт
5	Седьмой байт (старший байт 64-битного $Mq[i]$)

Mq напечатайте до и после изменения аналогично Л3.№1.

Задание Л3.№4. Реализуйте вставку, записывающую в $M[i]$ (M по варианту согласно таблице Л3.1) значение переменной x; размер x равен размеру элемента M. Значение x передаётся как входной параметр *в памяти,* адрес M и индекс i — как входные параметры *в РОН*.

Так как команда x86/amd64 не может адресовать два операнда в памяти, прямая пересылка $x \to M[i]$ невозможна; используйте промежуточный регистр (по варианту согласно таблице Л3.3).

Варианты временного РОН

Таблица Л3.3

$(№ - 1)\%7 + 1$	**Вариант**
1	Регистр A ($rax/eax/ax/al$)
2	Регистр C ($rcx/ecx/cx/cl$)
3	Регистр D ($rdx/edx/dx/dl$)
4	Регистр SI ($rsi/esi/si/sil$)
5	Регистр DI ($rdi/edi/di/dil$)
6	Регистр $R8$ ($r8/r8d/r8w/r8b$) на 64-битной платформе, A на 32
7	Регистр $R9$ ($r9/r9d/r9w/r9b$) на 64-битной платформе, C на 32

M напечатайте до и после изменения аналогично Л3.№1.

Задание Л3.№5. Реализуйте вставку, записывающую в $M[i]$ значение x аналогично Л3.№4, но во вставку передаётся адрес $\&x$.

Задание Л3.№6. Реализуйте вставку, рассчитывающую для целочисленных x и y значения $z = x + y$ и $w = x - y$ при помощи команд add и sub. Разрядность указана в таблице Л3.4; переменные x, y, z, w передаются во вставку как параметры (z и w — выходные, x и y — входные).

Значения x, y, z, w напечатайте до и после вставки.

Варианты разрядности x, y, z, w

Таблица Л3.4

$(№ - 1)\%2 + 1$	**Вариант**
1	64 бита
2	16 бит

Задание Л3.№7. Определите, доступны ли на выбранной платформе расширения AVX и SSE, используя команду $cpuid$ или документацию на процессор.

Если доступно расширение AVX (рекомендуется компанией Intel как более быстрое, но отсутствует в удешевлённых процессорах), то далее везде, где сказано «команды AVX или их SSE-аналоги» — необходимо использовать **только AVX.** SSE-аналоги (не рекомендуются Intel, но в настоящее время доступны во всех процессорах) используются только в том случае, если нет AVX.
Штраф −2 балла за задание, где смешиваются команды AVX и SSE, так как это не даёт ни производительности, ни переносимости.
При смене платформы необходимо проверить доступность AVX на новой.

Как в задании Л1.№3, создайте массивы:
— Mfl из 64-битных чисел с плавающей запятой $double$;
— Mfs из 32-битных чисел с плавающей запятой $float$;
длина N и начальные значения аналогичны Л1.№3.

Реализуйте вставку, записывающую в $M[i]$ (M по варианту согласно таблице Л3.5) значение переменной x с плавающей запятой (размер x равен размеру элемента M), используя команды AVX $vmovsd/vmovss$ или их SSE-аналоги $movsd/movss$. Значение x передаётся как входной параметр *в памяти,* адрес M и индекс i — как входные параметры *в РОН.*

Варианты массива M из значений с плавающей запятой

Таблица Л3.5

(№ − 1)%2 +1	**Вариант**
1	Mfl
2	Mfs

Используйте промежуточный регистр xmmj, где номер регистра j $\in [0,5]$ рассчитывается по варианту как (№ − 1)%6.

Задание Л3.№8. Реализуйте вставку, записывающую в $M[i]$ (M по варианту согласно таблице Л3.5) значение с плавающей запятой, равное целочисленному значению x. Преобразование целочисленного x к нужному виду выполните при помощи команд AVX $vcvtsi2sd/vcvtsi2ss$ или их SSE-аналогов $cvtsi2sd/cvtsi2ss$.

Л3.2. Дополнительные бонусные и штрафные баллы

−2 балла за каждое задание, где не печатаются исходные данные.
−2 балла за каждое задание, где смешиваются команды AVX и SSE.
−3 балла за утечку памяти (выделенные, но не освобождённые блоки).

Лабораторная работа 4 (0100 = 4)
Модули и функции

Засчитывается только весной 2025 г.; актуальная версия в https://gitlab.com/illinc/gnu-asm

Цель работы: изучить стандартные соглашения о вызовах и их соответствие платформам, научиться комбинировать функции на C/C++ и ассемблере.

Каждый из модулей в заданиях Л4 реализуется либо на чистом ассемблере, либо на чистом C/C++ (без ассемблерных вставок).

Штраф за одно пропущенное обязательное задание -2 балла.

Л4.1. Задание на лабораторную работу

Все задания Л4 выполняются **дважды,** для различных соглашений о вызовах:

а) **System V amd64 psABI** (ОС GNU/Linux 64, MacOS X 64 и другие BSD 64) — на допустимых онлайн-платформах или личном ПК студента;

б) **Microsoft 64** (только ОС Microsoft Windows 64) — в аудитории ВЦ МИЭТ (рисунки стека должны быть подготовлены заранее) или на личном ПК.

Штраф за задание, выполненное только для одного соглашения: -1 балл.

Если доступно cdecl (x86, язык C и любая 32-битная ОС; особенности для MacOS X 32) — трижды, **$+2$ балла за каждое задание с тремя соглашениями.**

Для **каждой функции** на ассемблере, в том числе $main()$, на клетчатой бумаге должен быть нарисован стек после пролога или в указанный явно момент (дважды/трижды: **по рисунку на соглашение).**

Стек изображается в виде столбца из прямоугольников (8-байтовых стековых слов, для cdecl — 4-байтовых) аналогично рис. Л4.1. Адреса растут снизу вверх, соответственно, стек — вниз; вершина стека — внизу рисунка, дно — вверху.

Должны быть показаны все стековые слова, начиная от адреса возврата (AB) из функции и заканчивая вершиной стека ($(\%rsp)$). Всё, что хранится в стеке:

— стековые локальные переменные функции;

— стековые параметры (и теневое пространство для Microsoft 64) для вложенного вызова функции $f()$, если стек рисуется «перед вызовом $f()$»;

должно быть не просто перечислено, а изображено в конкретном стековом слове. Если в одном слове несколько локальных переменных — граница между ними изображается параллельно границам между словами.

Если один рисунок соответствует нескольким функциям или соглашениям — укажите это в описании рисунка; дублировать его не надо.

Штраф -2 балла за задание, где изображений стека нет или изображение для какой-либо ОС не совпадает с программой для этой ОС.

Для проверки ассемблерных функций, вычисляющих выражение от переданных параметров — реализуйте вычисление того же выражения на C/C++.

Задание Л4.№1. Разработайте ассемблерную функцию $f1()$, вычисляющую целое выражение от двух целых аргументов (таблица Л4.1), а также головную программу на языке C/C++, использующую разработанную функцию.

Варианты целочисленных выражений для расчёта

Таблица Л4.1

$(№ - 1)\%2 + 1$	**Вариант**
1	$f1(x, y) = -7 + x + 8y$
2	$f1(x, y) = 12 + x + y$

Учтите, что для того, чтобы головная программа могла использовать $f1()$, недостаточно объявить метку $f1$ глобальной в ассемблерном модуле и описать $f1()$ как внешнюю в головной программе — необходимо отменить для $f1()$ декорирование (не просто `extern`, а `extern "C"`).

Бонус +1 балл, если вычисление производится одной командой lea.

Чем отличается и в чём схожа передача целочисленных параметров в System V amd64 psABI и Microsoft 64?

Задание Л4.№2. Разработайте ассемблерную функцию $f2()$, вычисляющую выражение от двух чисел с плавающей запятой двойной точности x и y (таблица Л4.2), используя команды AVX $vsubsd$ и $vdivsd$ или их SSE-аналоги $subsd$ и $divsd$, а также головную программу на языке C/C++.

Варианты выражений с плавающей запятой для расчёта

Таблица Л4.2

$(№ - 1)\%2 + 1$	**Вариант**
1	$f2(x, y) = x - y$
2	$f2(x, y) = x/y$

Чем отличается и в чём схожа передача параметров с плавающей запятой в System V amd64 psABI и Microsoft 64 вообще? Различается ли реализация $f2()$?

Задание Л4.№3. Разработайте программу, целиком написанную на ассемблере, которая печатает группу, номер и состав команды при помощи функции $puts()$ библиотеки libc (аналогично заданию Л1.№1).

Обратите внимание, что для вложенного вызова функции мало передать параметры через соответствующие регистры — необходимо **полностью** соблюсти соглашение, соответствующее платформе.

Используйте то, что стартовый код libc также соблюдает это соглашение, и до вызова $main()$ (по обоим 64-битным соглашениям) выравнивает стек на 16 байт (верхняя пунктирная линия $16x$ на рис. Л4.1). После вызова $main()$ на стек

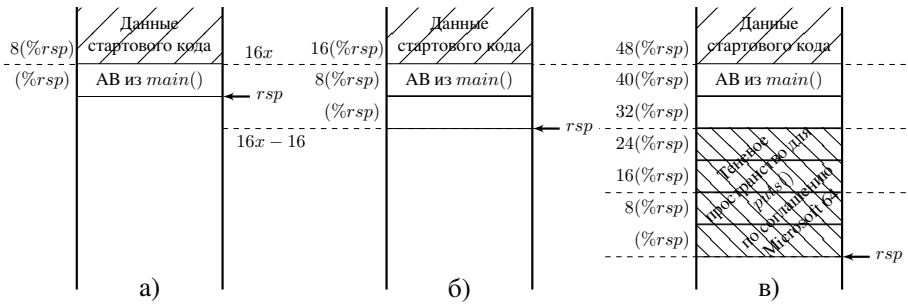

Рис. Л4.1. Стек: а) сразу после вызова $main()$ (оба соглашения);
б) перед вызовом $puts()$ (System V amd64 psABI);
в) перед вызовом $puts()$ (Microsoft 64)

ложится адрес возврата (рис. Л4.1, а). Состояние стека перед вызовом $puts()$ для двух соглашений показано на рис. Л4.1, б) и в).

Чем отличаются и в чём совпадают требования соглашений System V amd64 psABI и Microsoft 64 в случае функции *с постоянным числом параметров* — такой, как $puts()$, а также как разработанные выше $f1()$ и $f2()$?

Задание Л4.№4. Разработайте на языке C/C++ программу, которая:

а) включает пять локальных для $main()$ переменных:
- $i16$ — 16-битное целое $short/unsigned\ short$;
- $i32$ — 32-битное целое $int/unsigned\ int$;
- $i64$ — 64-битное целое $long\ long/unsigned\ long\ long$;
- $f32$ — 32-битное число с плавающей запятой $float$;
- $f64$ — 64-битное число с плавающей запятой $double$;

б) запрашивает значения $i16$, $i32$, $i64$, $f32$, $f64$ одним вызовом $scanf()$;

в) печатает значения $i16$, $i32$, $i64$, $f32$, $f64$ одним вызовом $printf()$;

убедитесь, что ввод-вывод работает корректно.

Разработайте аналогичную программу, целиком реализованную на ассемблере. Форматные строки и вообще вызовы $scanf()$ и $printf()$ должны совпадать с C/C++-прототипом. Локальные переменные на ассемблере, естественно, не имеют имён, но в документации имена используются и здесь совпадают с прототипом.

Обдумайте, как разместить пять локальных переменных $i16$, $i32$, $i64$, $f32$, $f64$ в четырёх 64-битных стековых словах с учётом выравнивания — то есть адрес переменной должен быть кратен как минимум её размеру. Изобразите фрагмент стека с переменными аналогично рис. Л4.1; укажите, где какая переменная.

Поместятся ли в регистры все параметры $scanf()$? Какие параметры $scanf()$ будут стековыми? Изобразите стек перед вызовом $scanf()$. Расположение переменных друг относительно друга должно совпасть с разработанным ранее.

Этот и последующие рисунки для Л4.№4 делайте для укороченного пролога/эпилога (только уменьшение rsp в прологе и увеличение в эпилоге; локальные переменные адресуются относительно rsp).

Изобразите стек перед вызовом $printf()$, причём расположение AB из $main()$ и переменных должно совпасть с AB и переменными перед вызовом $scanf()$.

Сколько стековых слов (и, соответственно, сколько байтов) необходимо зарезервировать в прологе, чтобы хватило и на переменные, и на $scanf()$, и на $printf()$, и были бы соблюдены все требования соглашения?

Исправьте изображения стека перед вызовами $scanf()$ и $printf()$ с учётом сделанных выводов. Рассчитайте смещения переменных $i16$, $i32$, $i64$, $f32$, $f64$, а также стековых параметров $scanf()$ и $printf()$ относительно rsp.

Реализуйте по разработанным изображениям стека программу.

Чем отличаются и в чём совпадают соглашения System V amd64 psABI и Microsoft 64 в случае функции *с переменным числом параметров,* как $scanf()$ и $printf()$?

Задание Л4.№5. Разработайте программу, целиком реализованную на ассемблере, аналогичную заданию Л4.№4, но с **классическим** прологом/эпилогом (локальные переменные адресуются относительно rbp).

Для этого также изобразите стек перед вызовами $scanf()$ и $printf()$ с учётом копии старого значения rbp. Укажите стрелкой, на какое место в стеке указывает rbp после классического пролога (какой адрес хранится в rbp).

Что изменилось по сравнению с соответствующими рисунками Л4.№4?

Рассчитайте смещения переменных $i16$, $i32$, $i64$, $f32$, $f64$, а также стековых параметров $scanf()$ и $printf()$ не только относительно rsp (положительные смещения), но и относительно rbp (отрицательные).

Реализуйте по разработанным изображениям стека программу; локальные переменные $i16$, $i32$, $i64$, $f32$, $f64$ должны адресоваться через rbp.

Какие преимущества классический пролог имеет перед укороченным?

Задание Л4.№6. Бонус +2 балла для пар, обязательное для троек.

Опишите на C/C++ функцию с восемью параметрами типа $int/unsigned$, которая печатает свои параметры и возвращает результат, равный восьмому параметру.

Разработайте головную программу на ассемблере, вызывающую эту функцию.

Л4.2. Дополнительные бонусные и штрафные баллы

−2 балла за хранение локальных переменных в сегменте данных (использование статических/глобальных констант допускается).
−3 балла за нарушение соглашения о вызовах (даже если в данной конкретной программе это не имеет видимых проявлений).

Л4.3. Подключение к проекту модулей на ассемблере

Если программа собирается в консоли, ничего особенного делать не нужно. IDE обычно заточена под модули на C/C++, так что подключение к проекту модулей на ассемблере в IDE усложнено по сравнению с консолью.

Консоль

Аналогично тому, как программа из одного модуля `main.cpp` на C++ собирается командой `g++ main.cpp` (возможно, с дополнительными ключами), а программа из двух модулей на C++ (`main.cpp` и `unit.cpp`) — командой

```
g++ main.cpp unit.cpp
```

программа из одного модуля `main.S` на ассемблере собирается `g++ main.S`, из двух (`main.cpp` на C++ и `unit.S` на ассемблере) — командой

```
g++ main.cpp unit.S
```

из `main.S` на ассемблере и `unit.cpp` на C++ — `g++ main.S unit.cpp` и т. п.

В командой строке может быть указано произвольное количество имён модулей; головным является тот, где описана функция $main()$, независимо от имени модуля и порядка упоминания в командной строке.

Язык каждого модуля должен соответствовать его расширению:
— `.cpp` — C++;
— `.c` — C;
— `.S` — ассемблер, возможно, включающий директивы препроцессора;
— `.s` — ассемблер строго без директив препроцессора.
Проект только из ассемблерных модулей может собираться как компилятором `g++` (как C++), так и `gcc` (как простой C).

Qt Creator

Файл проекта Qt Creator для добавления ассемблерного модуля необходимо отредактировать вручную, так как мастер добавления файлов не воспринимает расширения `.S` и `.s` как допустимые для исходного кода. В частности, для файла `main.S` необходимо добавить строку `SOURCES += main.S`.

Листинг Л.1. Файл проекта, содержащего `main.S` и `unit.S`

```
1 TEMPLATE = app
```

```
2 CONFIG += console
3 CONFIG -= app_bundle
4 CONFIG -= qt
5
6 SOURCES += main.S
7 SOURCES += unit.S
8
9 include(deployment.pri)
10 qtcAddDeployment()
```

Файлы `main.S` и `unit.S` здесь должны находиться в той же папке, что и проект, иначе в строке после `SOURCES +=` необходимо указать имя файла с относительным путём. Других настроек, кроме редактирования файла проекта, делать не нужно.

Code::Blocks

Создать ассемблерный модуль в Code::Blocks можно, используя меню $File \rightarrow New \rightarrow Empty\ file$. Имя файла обязательно должно иметь расширение `.S` (заглавное; расширение `.s` не воспринимается Code::Blocks как допустимое).

После создания в проекте файла с таким расширением он во время сборки проекта обрабатывается препроцессором и компилируется gcc; полученный объектный файл в дальнейшем используется компоновщиком. Дополнительных настроек делать не нужно.

Лабораторная работа 5 (0101 = 5)

Флаги и условные команды. Ветвления и циклы

Засчитывается только весной 2025 г.; актуальная версия в https://gitlab.com/illinc/gnu-asm

Цель работы: ознакомиться с набором флагов состояния регистра $flags$, семействами команд, выполняющихся по-разному в зависимости от флагов (условных команд); научиться реализовывать ветвления и циклы на ассемблере.

Штраф за одно пропущенное обязательное задание -2 балла.

Л5.1. Задание на лабораторную работу

Задание Л5.№1. Вычислите сумму двух целых чисел $z = x + y$, используя команду add. Сформируйте w (таблица Л5.1), используя семейство команд $setCC$ и анализируя флаги состояния CF, OF, SF, ZF, AF, PF после вычисления z.

Варианты w

Таблица Л5.1

$(№ - 1)\%3 + 1$	**Вариант**
1	$w = \begin{cases} 0, & \text{если не было беззнакового переполнения,} \\ 1, & \text{если было беззнаковое переполнение} \end{cases}$
2	$w = \begin{cases} 0, & \text{если не было знакового переполнения,} \\ 1, & \text{если было знаковое переполнение} \end{cases}$
3	$w = \begin{cases} 0, & \text{если } z \neq 0, \\ 1, & \text{если } z = 0 \end{cases}$

Задание Л5.№2. Вычислите z для заданного целого беззнакового x (таблица Л5.2); z принимает значение 1 либо 0, аналогично операторам сравнения C/C++.

Варианты z для заданий на $setCC$

Таблица Л5.2

$(№ - 1)\%2 + 1$	**Вариант**
1	$z = (x \leqslant 12)$
2	$z = (x > -3)$

Задание Л5.№3. Реализуйте Л5.№2 для целого знакового x.

Задание Л5.№4. Реализуйте Л5.№2 для x с плавающей запятой (таблица Л5.3), используя AVX-команды сравнения $vcomisd/vcomiss$ (или их SSE-аналоги).

Варианты типов с плавающей запятой для AVX/SSE

Таблица Л5.3

$(№ - 1)\%2 + 1$	**Вариант**
1	Двойной точности ($double$)
2	Одинарной точности ($float$)

Задание Л5.№5. Вычислите z для заданного целого беззнакового x (таблица Л5.4), используя семейство условных команд $cmovCC$ и выставляя флаги

Варианты выражений для $cmovCC$

Таблица Л5.4

$(№ - 1)\%3 + 1$	**Вариант**
1	$z = \begin{cases} 2x+1, & 2x+1 > 9, \\ 2, & 2x+1 \leqslant 9 \end{cases}$
2	$z = \begin{cases} -2+x, & -2+x \geqslant -2, \\ 25, & -2+x < -2 \end{cases}$
3	$z = \begin{cases} 32, & 5x > 7, \\ 5x, & 5x \leqslant 7 \end{cases}$

состояния при помощи команды cmp.

Бонус +1 балл, если вычисление линейной комбинации производится одной командой lea.

Задание Л5.№6. Заполните массив из N целочисленных элементов первыми N членами последовательности (таблица Л5.5).

Выделение памяти под массив может быть выполнено на языке C/C++, в этом случае в ассемблерную функцию передаётся адрес начала массива и длина N.

Задание Л5.№7. Бонус +2 балла для пар, обязательное для троек. Напечатайте первые N членов последовательности (таблица Л5.5), не сохраняя их в массиве.

Варианты последовательности

Таблица Л5.5

$(№-1)\%3+1$	**Вариант**
1	Кратные 3 неотрицательные: $0, 3, 6, 9, 12...$
2	Нечётные неотрицательные: $1, 3, 5, 7, 9...$
3	Вида $3k+2$ неотрицательные: $2, 5, 8, 11, 14...$

Так как вызов функций из ассемблерной вставки является неопределённым поведением, это задание может быть выполнено только как функция, целиком выполненная на ассемблере (или фрагмент функции $main()$, целиком выполненной на ассемблере).

Задание Л5.№8. Разработайте функцию, которая принимает адрес pM матрицы M из $R \times C$ или $N \times N$ элементов типа $int/unsigned$ и заменяет в ней часть элементов по варианту на (-1).

Варианты элементов для замены (Л5.№8)

Таблица Л5.6

$(№-1)\%5+1$	**Вариант**
1	$mre(\text{void } *pM, \text{size\_t } R, \text{size\_t } C, \text{size\_t } i)$ заменяет строку i
2	$mre(\text{void } *pM, \text{size\_t } R, \text{size\_t } C, \text{size\_t } j)$ заменяет столбец j
3	$mre(\text{void } *pM, \text{size\_t } N)$ заменяет главную диагональ
4	$mre(\text{void } *pM, \text{size\_t } N)$ заменяет побочную диагональ
5	$mre(\text{void } *pM, \text{size\_t } N, \text{size\_t } i)$ заменяет побочную ломаную диагональ $M_{i,0}, M_{i-1,1}, ..., M_{0,i}, M_{R-1,i+1}, ...$

Выделение памяти, заполнение и печать M до и после изменения — на языке C/C++.

Элементы матрицы M печатаются как матрица: элементы одной строки на одной строке и разделяются пробелами; младшая цифра $M_{i+1,j}$ под младшей цифрой $M_{i,j}$.

Штраф -1 балл, если цикл по элементам строки/столбца/диагонали содержит вложенные ветвления для расчёта i и j. Используйте арифметические операции.

Л5.2. Дополнительные бонусные и штрафные баллы

−1 балл за каждую некорректную секцию перезаписываемых элементов (clobbers) во вставках — и с неуказанием нужного, и с указанием лишнего.
−2 балла за каждое задание, где смешиваются команды AVX и SSE.
−2 балла за хранение локальных переменных в сегменте данных (использование статических/глобальных констант допускается).
−3 балла за нарушение соглашения о вызовах (даже если в данной конкретной программе это не имеет видимых проявлений).
−3 балла за утечку памяти (выделенные, но не освобождённые блоки).
−3 балла за команды `loop*`, `jcx*` или `jecx*` (`dec` + `jz`/`jnz` вдвое быстрее).

Лабораторная работа 6 (0110 = 6)

Целочисленные вычисления. Команды общего назначения

Засчитывается только весной 2025 г.; актуальная версия в https://gitlab.com/illinc/gnu-asm

Цель работы: научиться использовать команды общего назначения x86/amd64, в том числе — предназначенные для целочисленных вычислений. Научиться использовать для вычислений lea и битовые операции.

Штраф за одно пропущенное обязательное задание -2 балла.

Л6.1. Задание на лабораторную работу

Задание Л6.№1. Разработайте на C/C++ функцию $fc16\_c(\text{void } *p)$, которая принимает адрес 16-битной целочисленной переменной x и выполняет беззнаковое округление её значения до кратного D (таблица Л6.1) дважды:

Варианты значений

Таблица Л6.1

$(№ - 1)\%5 + 1$	**Вариант**
1	$D = 16$
2	$D = 32$
3	$D = 64$
4	$D = 128$
5	$D = 256$

а) вниз ($x_1 \leqslant x$);
б) вверх ($x_2 \geqslant x$).

Исходное значение x и каждый результат x_1 и x_2 печатается $print16()$ из Л2.

При x, некратном D, получится два различных результата: $x_1 < x_2$, но при кратном они должны совпасть: $x_1 = x_2 = x = k \cdot D$.

Как реализовать это без ветвлений, на битовых и арифметических операциях? Если изначально в $fc16\_c()$ были ветвления, избавьтесь от них.

Разработайте аналогичную функцию $fc16\_asm(\text{void } *p)$ на ассемблере.

Задание Л6.№2. Разработайте на ассемблере функцию $ab16\_asm(\text{void } *p)$, которая принимает адрес 16-битной целочисленной переменной x и выполняет над её копиями все уникальные операции, описанные в задании Л2.№9.

Дублирующиеся арифметические/битовые операции необходимо реализовать один раз — как битовые.

Исходное значение x и каждый результат печатается $print16()$.

Дальнейшие задания могут быть реализованы как в виде функций на ассемблере, так и в виде ассемблерных вставок в программу на C/C++.

Задание Л6.№3. Вычислите целочисленное выражение (таблица Л6.2) для заданных целых x и y. Разрядность x, y, z совпадает.

Варианты целочисленных выражений

Таблица Л6.2

$(№ - 1)\%2 + 1$	**Вариант**
1	$z = x + x \cdot y - 7$
2	$z = 12 - x - y^2$

Задание Л6.№4. Вычислите беззнаковое целочисленное выражение (таблица Л6.3) для заданных целых x и y.

Варианты беззнакового/знакового деления

Таблица Л6.3

$(№ - 1)\%3 + 1$	**Вариант**
1	$\begin{cases} z = x/(y-1) \\ w = x\%(y-1) \end{cases}$
2	$\begin{cases} z = (x+2)/y \\ w = (x+2)\%y \end{cases}$
3	$\begin{cases} z = x/(y-3) \\ w = x\%(y-3) \end{cases}$

Вычислите знаковое целочисленное выражение (таблица Л6.3) для тех же x и y.

Сравните результаты беззнакового и знакового деления в случае, когда делимое равно -3, а делитель $+2$.

Задание Л6.№5. Вычислите целочисленное выражение (таблица Л6.4) для заданного целого x, используя *одну* команду. Разрядность x и z совпадает.

Штраф -2 балла, если умножение выполняется командами $mul/imul$.

Варианты целочисленных выражений для lea и битовых операций

Таблица Л6.4

$(№ - 1)\%9 + 1$	**Вариант**
1	$z = x + 9$
2	$z = 2x - 8$
3	$z = 3x + 7$
4	$z = 4x - 6$
5	$z = 5x + 5$
6	$z = 8x - 4$
7	$z = 9x + 4$
8	$z = 16x$
9	$z = 32x$

Задание Л6.№6. Разработайте функцию $\text{int } mce\_si(\text{void } *p, \text{size\_t } N)$, которая принимает адрес массива из N элементов типа $int/unsigned$ и рассчитывает для него значение по варианту (таблица Л6.5).

Варианты скаляризации массива

Таблица Л6.5

$(№ - 1)\%2 + 1$	**Вариант**
1	сумма
2	произведение

Л6.2. Дополнительные бонусные и штрафные баллы

—1 балл за каждую некорректную секцию перезаписываемых элементов (clobbers) во вставках — и с неуказанием нужного, и с указанием лишнего.
—2 балла за хранение локальных переменных в сегменте данных (использование статических/глобальных констант допускается).
—3 балла за нарушение соглашения о вызовах (даже если в данной конкретной программе это не имеет видимых проявлений).
—2 балла за включение в отчёт заведомо недостоверных цифр.
—3 балла за утечку памяти (выделенные, но не освобождённые блоки).

Лабораторная работа 7 (0111 = 7)

Вычисления с плавающей запятой. Скалярные команды AVX/SSE

Засчитывается только весной 2025 г.; актуальная версия в https://gitlab.com/illinc/gnu-asm

Цель работы: исследовать особенности арифметики с плавающей запятой. Научиться использовать скалярные команды расширений AVX/SSE.

Л7.1. Задание на лабораторную работу

Для НБ-3* все задания выполняются на C/C++. Л7 — третья по счёту у НБ-3*.

Задание Л7.№1. Разработайте программу на языке C/C++, выполняющую вычисления над числами с плавающей запятой одинарной точности ($float$). Проверьте, что программа действительно работает с операндами одинарной точности, а не приводит к типу $float$ окончательный результат.

Для частичной суммы гармонического ряда $S(N) = \sum_{i=1}^{N} \frac{1}{i} \in \mathbb{R}$ найдите две её оценки: $S_d(N)$ — последовательно складывая члены, начиная от $i = 1$ и заканчивая $i = N$ («наивный» порядок), и $S_a(N)$ — от $i = N$ к $i = 1$. Сравните $S_d(N)$ и $S_a(N)$ для различных значений N: $10^3, 10^6, 10^9$. Объясните результат.

Измените тип операндов на $double$. Объясните результат.

Все N печатаются в экспоненциальной форме, а не с хвостом из нулей. Все $S_d(N)$ и $S_a(N)$ печатаются $print32()$ и $print64()$ из Л2.

Задание Л7.№2. Рассчитайте на языке C/C++ для заданного 32-битного значения с плавающей запятой x его модуль $|x|$, используя только битовые (целочисленные) операции и преобразование указателей. Исходное значение и результат печатаются $print32()$ из Л2.

Задание Л7.№3. Разработайте на ассемблере функцию $inc32\_asm(\text{void } *p)$, которая принимает адрес переменной p и выполняет целочисленный инкремент (команда inc) 32-битного (суффикс l) значения по этому адресу.

Разработайте на C/C++ программу, которая применяет $inc32\_asm(\text{void } *p)$ к 32-битным переменным с плавающей запятой типа $float$. Значения переменных a, b, c, d соответствуют варианту (таблица Л2.2 из Л2). Каждое исходное значение и каждый результат печатаются $print32()$ из Л2.

Как можно реализовать аналогичную функцию $inc32\_c(\text{void } *p)$ на C/C++?

Задание Л7.№4. Вычислите для заданных x и y с плавающей запятой двойной точности выражение из таблицы Л7.1, используя скалярные AVX-команды либо их SSE-аналоги.

Варианты выражений для расчёта простыми командами

Таблица Л7.1

$(№-1)\%2+1$	**Вариант**
1	$z = x/3 + 2/y - xy$
2	$z = 1 - 5/x - y^2/7$

Задание Л7.№5. Разработайте программу, вычисляющую по введённым значениям x и y с плавающей запятой двойной точности значение z (таблица Л7.2), вызывая функции libm $pow()/atan2()$.

Варианты выражений для расчёта libm

Таблица Л7.2

$(№-1)\%2+1$	**Вариант**
1	$z = \mathrm{pow}(x, y)$, x^y
2	$z = \mathrm{atan2}(x, y)$, угол между вектором (x, y) и осью абсцисс

Если программа не собирается из-за отсутствия ссылок на $pow()/atan2()$, добавьте к команде сборки ключ `-lm` (указание компоновщику использовать libm).

Задание Л7.№6. Разработайте функцию double mce_sd(void $*\,p$, size_t N), которая аналогично Л6.№6 обрабатывает массив из $double$ (таблица вариантов Л6.5).

Л7.2. Дополнительные бонусные и штрафные баллы

—1 балл за каждую некорректную секцию перезаписываемых элементов (clobbers) во вставках — и с неуказанием нужного, и с указанием лишнего.
—2 балла за хранение локальных переменных в сегменте данных (использование статических/глобальных констант допускается).
—2 балла за включение в отчёт заведомо недостоверных цифр.
—2 балла за каждое задание, где смешиваются команды AVX и SSE.
—3 балла за утечку памяти (выделенные, но не освобождённые блоки).
—3 балла за команды `loop*`, `jcx*` или `jecx*` (`dec` + `jz/jnz` вдвое быстрее).
—3 балла за нарушение соглашения о вызовах (даже если в данной конкретной программе это не имеет видимых проявлений).

Лабораторная работа 8 (1000 = 8)

Вычисления с плавающей запятой. Векторные команды AVX/SSE

Засчитывается только весной 2025 г.; актуальная версия в https://gitlab.com/illinc/gnu-asm

Цель работы: научиться использовать векторные команды расширений AVX/SSE.

Л8.1. Задание на лабораторную работу

Задание Л8.№1. Разработайте ассемблерную функцию size_t $init\_pd$(void $*$ p, size_t N, double x), которая, если длина массива N кратна четырём, инициализирует массив из $double$ по адресу p из N элементов одинаковыми значениями x и возвращает N. используя векторные команды AVX $vmovupd$, $vpbroadcastd$ и ymm-регистры (если они недоступны — SSE-аналоги и xmm).

При некратном четырём N вернуть -1.

Задание Л8.№2. Вычислите для массивов $(x_0, ...x_3)$ и $(y_0, ...y_3)$ из четырёх чисел с плавающей запятой двойной точности значения $(z_0, ...z_3)$ согласно таблице Л8.1, используя векторные команды AVX $vmovupd$, $vaddpd$, $vsubpd$, $vmulpd$, $vdivpd$, $vpbroadcastd$ и ymm-регистры (если они недоступны — SSE-аналоги и xmm).

Варианты действий с массивами

Таблица Л8.1

(№ − 1)%3 +1	**Вариант**
1	$z_i = x_i - y_i + 1$
2	$z_i = x_i / y_i + 2$
3	$z_i = x_i y_i - 5$

Выделение памяти под x, y, z и заполнение массивов x, y может быть выполнено на C/C++. Проверьте расчёт, реализовав то же самое на C/C++.

Задание Л8.№3. Разработайте ассемблерную функцию int $v4$(void $*px$, void $*$ py, void $*$ pz, size_t N), рассчитывающую z согласно таблице Л8.1 для длины N, кратной четырём.

Возвращаемое значение должно быть равно -1 при N, не кратном 4, и количеству успешно рассчитанных элементов z при корректном N.

Задание Л8.№4. Разработайте ассемблерную функцию int $v1$(void $*px$, void $*$ py, void $*$ pz, size_t N), аналогичную Л8.№3 для произвольной длины N.

Проверьте, что массив z корректно заполняется (то есть ячейки от $pz[0]$ до $pz[N-1]$ перезаписываются верными значениями, а $pz[N]$ и далее не изменяются) при $N \in \{4k, 4k+1, 4k+2, 4k+3\}$ для выбранного k.

Л8.2. Дополнительные бонусные и штрафные баллы

−1 балл за каждую некорректную секцию перезаписываемых элементов (clobbers) во вставках — и с неуказанием нужного, и с указанием лишнего.
−2 балла за хранение локальных переменных в сегменте данных (использование статических/глобальных констант допускается).
−2 балла за включение в отчёт заведомо недостоверных цифр.
−2 балла за каждое задание, где смешиваются команды AVX и SSE.
−3 балла за утечку памяти (выделенные, но не освобождённые блоки).
−3 балла за команды `loop*`, `jcx*` или `jecx*` (`dec` + `jz/jnz` вдвое быстрее).
−3 балла за нарушение соглашения о вызовах (даже если в данной конкретной программе это не имеет видимых проявлений).

Лабораторная работа 9 (1001 = 9)

Вычисления с плавающей запятой. Команды FPU

Засчитывается только весной 2025 г.; актуальная версия в https://gitlab.com/illinc/gnu-asm

Цель работы: научиться использовать команды FPU.

Л9.1. Задание на лабораторную работу

Задание Л9.№1. Вычислите для заданных x и y с плавающей запятой двойной точности выражение из таблицы Л7.1, используя FPU.

Результаты заданий Л7.№4 и Л9.№1 напечатайте $print64()$ из Л2. Совпадают ли результаты побитово? Если значения различаются — как вы думаете, какое из них точнее? Достаточно ли они близки друг к другу, чтобы можно было считать их совпадающими в пределах допустимой погрешности?

Задание Л9.№2. Вычислите для заданных x и y с плавающей запятой двойной точности выражение из таблицы Л7.2, используя FPU и не используя функций libm.

Результаты заданий Л7.№5 и Л9.№2 напечатайте $print64()$ из Л2 и сравните аналогично Л9.№1.

Задание Л9.№3. Вычислите для заданных x и y с плавающей запятой двойной точности выражение из таблицы Л9.1, используя FPU.

Варианты выражений для расчёта

Таблица Л9.1

$(№-1)\%2+1$	**Вариант**
1	$(100{,}69 \mod x)\cdot((-47{,}46) \mod y)$
2	$y\cdot\log_2(x+1)$

Задание Л9.№4. Реализуйте Л5.№2 для x с плавающей запятой (таблица Л9.2), используя FPU-команды сравнения $f[u]comi[p]$.

Л9.2. Дополнительные бонусные и штрафные баллы

−2 балла за несбалансированный стек FPU после завершения вычислений.
−1 балл за каждую некорректную секцию перезаписываемых элементов (clobbers) во вставках — и с неуказанием нужного, и с указанием лишнего.

Варианты типов с плавающей запятой для FPU

Таблица Л9.2

$(№ - 1)\%3 + 1$	**Вариант**
1	Двойной расширенной точности ($long\ double$)
2	Двойной точности ($double$)
3	Одинарной точности ($float$)

—2 балла за хранение локальных переменных в сегменте данных (использование статических/глобальных констант допускается).
—2 балла за включение в отчёт заведомо недостоверных цифр.
—2 балла за каждое задание, где смешиваются команды AVX и SSE.
—3 балла за утечку памяти (выделенные, но не освобождённые блоки).
—3 балла за команды `loop*`, `jcx*` или `jecx*` (`dec` + `jz`/`jnz` вдвое быстрее).
—3 балла за нарушение соглашения о вызовах (даже если в данной конкретной программе это не имеет видимых проявлений).

Лабораторная работа 10 (1010 = A)

Флаги и условные команды. Вложенные ветвления и циклы

Засчитывается только весной 2025 г.; актуальная версия в https://gitlab.com/illinc/gnu-asm

Цель работы: научиться реализовывать вложенные ветвления и циклы на ассемблере.

Штраф за одно пропущенное обязательное задание -2 балла.

ЛА.1. Задание на лабораторную работу

Задание ЛА.№1. Разработайте функцию $\text{size\_t}\ sa(\text{void}*p, \text{size\_t}\ N, \text{double}\ x)$, которая принимает адрес массива из N элементов типа $double$ и возвращает номер найденного элемента согласно таблице ЛА.1. Выделение памяти и заполнение массива может быть выполнено на языке C/C++.

Варианты обработки массива

Таблица ЛА.1

$(№-1)\%3+1$	**Вариант**
1	Найти минимальный элемент массива
2	Найти максимальный элемент массива
3	Найти ближайший к 0 элемент массива

Задание ЛА.№2. Разработайте функцию $na(\text{void}\ *M, \text{size\_t}\ N)$, которая принимает адрес массива из N элементов типа $double$ и нормирует все его элементы (таблица ЛВ.1).

Варианты нормировки

Таблица ЛА.2

$(№-1)\%2+1$	**Вариант**
1	Сумма элементов должна быть равна 64
2	Произведение элементов должно быть равно 1

Задание ЛА.№3. Разработайте функцию $\text{void}*\ malloc\_mmul(\text{void}\ *pM1, \text{size\_t}\ R1, \text{size\_t}\ C1, \text{void}\ *pM2, \text{size\_t}\ R2, \text{size\_t}\ C2)$, которая принимает адреса двух матриц M_1 из $R_1 \times C_1$ элементов и M_2 из $R_2 \times C_2$ элементов типа $double$,

при $C_1 = R_2$ выделяет память под произведение $M_1 \cdot M_2$ из $R_1 \times C_2$ элементов, заполняет его и возвращает адрес сформированной матрицы $M_1 \cdot M_2$.

Если $C_1 \neq R_2$, вернуть $NULL$.

ЛА.2. Дополнительные бонусные и штрафные баллы

−1 балл за каждую некорректную секцию перезаписываемых элементов (clobbers) во вставках — и с неуказанием нужного, и с указанием лишнего.
−2 балла за каждое задание, где смешиваются команды AVX и SSE.
−2 балла за хранение локальных переменных в сегменте данных (использование статических/глобальных констант допускается).
−3 балла за нарушение соглашения о вызовах (даже если в данной конкретной программе это не имеет видимых проявлений).
−3 балла за утечку памяти (выделенные, но не освобождённые блоки).
−3 балла за команды `loop*`, `jcx*` или `jecx*` (`dec` + `jz`/`jnz` вдвое быстрее).

Лабораторная работа 11 (1011 = B)

Структуры и массивы структур

Засчитывается только весной 2025 г.; актуальная версия в https://gitlab.com/illinc/gnu-asm

Цель работы: ознакомиться с понятием выравнивания; научиться обрабатывать структуры и массивы структур

Штраф за одно пропущенное обязательное задание -2 балла.

ЛВ.1. Задание на лабораторную работу

Для НБ-3* все задания выполняются на C/C++. ЛВ — четвёртая у НБ-3*.

Задание ЛВ.№1. Опишите на C/C++ структуру, состоящую из двух полей:
— целочисленное поле $\mathrm{int}\ key$ — ключ;
— поле с плавающей запятой $\mathrm{double}\ val$ — значение.
Каков размер структуры (в байтах)? Каков суммарный размер её полей (в байтах)? Каковы адреса key и val?

Пересоберите программу, указав максимальную кратность выравнивания 4, для чего либо укажите в командной строке ключ компилятора `-fpack-struct=4`, либо добавьте в начало файла директиву препроцессора `#pragma pack(4)`. Что изменилось?

Пересоберите программу, указав максимальную кратность выравнивания 8. Что изменилось?

Какие поля надо добавить в структуру, чтобы её размер не зависел от максимальной кратности выравнивания, а key и val всегда были выравнены кратно своему размеру? Как получить адрес поля полученной структуры, зная адрес начала структуры и её состав?

Задание ЛВ.№2. Разработайте $\mathrm{size\_t}\ sk(\mathrm{void}\ *p, \mathrm{size\_t}\ N, \mathrm{double}\ x, \mathrm{int}\ k)$, которая принимает адрес массива из структур задания ЛВ.№1 (размер не должен зависеть от настроек выравнивания) и реализует задание ЛА.№1 для значений тех элементов, ключ которых равен заданному числу k.

При отсутствии элементов с ключом k вернуть -1.

Задание ЛВ.№3. Разработайте функцию $\mathrm{size\_t}\ na(\mathrm{void}\ *M, \mathrm{size\_t}\ N, \mathrm{int}\ k)$, массива из структур задания ЛВ.№1 и нормирует, как в задании ЛА.№2 (таблица ЛВ.1), те элементы, ключ которых равен заданному числу k.

Вернуть количество обработанных элементов.

ЛВ.2. Дополнительные бонусные и штрафные баллы

-1 балл за каждую некорректную секцию перезаписываемых элементов (clobbers) во вставках — и с неуказанием нужного, и с указанием лишнего.

Варианты нормировки

Таблица ЛВ.1

$(№ - 1)\%2 + 1$	**Вариант**
1	Сумма элементов должна быть равна 64
2	Произведение элементов должно быть равно 1

—2 балла за каждое задание, где смешиваются команды AVX и SSE.
—2 балла за хранение локальных переменных в сегменте данных (использование статических/глобальных констант допускается).
—3 балла за нарушение соглашения о вызовах (даже если в данной конкретной программе это не имеет видимых проявлений).
—3 балла за утечку памяти (выделенные, но не освобождённые блоки).
—3 балла за команды `loop*`, `jcx*` или `jecx*` (`dec` + `jz`/`jnz` вдвое быстрее).

Лабораторная работа 12 (1100 = C)

Просмотр и редактирование файлов в шестнадцатеричном представлении. Работа с файлами в C/C++

Засчитывается только весной 2025 г.; актуальная версия в https://gitlab.com/illinc/gnu-asm

Лабораторная работа предназначена для групп ПМ-3* и ПИН-41Д; группы ПИН-2* выполнят её задания через полгода, осенью — сейчас не нужно.

Цель работы: 1) изучить двоичное и шестнадцатеричное представление информации, научиться работать с шестнадцатеричным редактором; 2) изучить структуру простейших форматов файлов; 3) изучить кодировки и кодовые таблицы русского языка.

Штраф за одно пропущенное обязательное задание -2 балла.

Так как задания данной лабораторной работы являются также частью курса ОТИК групп ПИН-, файлы лежат в репозитории ОТИК `gitlab.com/illinc/otik/`, в подпапках папки `/labs-files/`.*

ЛС.1. Задание на лабораторную работу

Задание ЛС.№1. С помощью hexdump/xxd или шестнадцатеричного просмотровщика/редактора исследуйте файлы различных форматов (некоторые файлы представлены в папке «labs-files/Файлы в разных форматах»). Выделите сигнатуры или иные признаки формата там, где это возможно.

Описания части форматов находятся в папке «labs-files/Описание некоторых форматов», для прочих можно найти в Сети.

Задание ЛС.№2. Определите тип файла, соответствующего номеру варианта $((№ - 1)\%10)$, из папки «labs-files/Варианты 1 — *». Откройте его корректным приложением. Поместите в отчёт тип и описание содержимого файла, а также признаки формата, по которым удалось определить тип.

Задание ЛС.№3. В соответствии с номером варианта отредактируйте (таблица ЛС.1) изображение colorchess16x16x2.bmp, используя шестнадцатеричный редактор. Откройте изменённый файл и убедитесь, что изменения корректны.

Файл colorchess16x16x2.bmp представляет собой изображение 16×16 пикселей, сиренево-болотное (две сиреневые и две болотные клетки по 8×8 пикселей), глубина цвета — 1 бит на пиксель. Убедитесь, что просмотровщик отображает его как цветное (некоторые игнорируют палитру файлов с глубиной 1 бит на пиксель).

Задание ЛС.№4. С помощью hexdump/xxd или шестнадцатеричного просмотровщика/редактора исследуйте файлы формата «простой текст» (plain text), представленные в различных кодировках (папка `labs-files/Файлы в формате простого текста - кодировки разные`, раздел ЛС.2).

Варианты действий для редактирования

Таблица ЛС.1

$(№ - 1)\%3 + 1$	**Вариант**
1	Поставить зелёную точку в правом нижнем углу изображения
2	Поставить сиреневую точку в правом верхнем углу изображения
3	Поставить зелёную точку в левом верхнем углу изображения

Есть ли у простого текста заголовок?

Сравните один и тот же текст, представленный в различных кодировках: размер и шестнадцатеричное представление.

Сравните шестнадцатеричное представление с тем, как текстовый редактор читает этот текст в кодировке платформы (обычно UTF-8). Учтите, что малофункциональные редакторы, такие как Блокнот MS Windows, поддерживают только одну-две кодировки, и «кракозябры» для остальных — нормальное явление.

Обратите внимание на файлы `Алфавит - *`, включающие 192 символа национальных кодовых таблиц русского языка / кодовой таблицы Unicode, в том числе:

— 189 печатных (пробелы, цифры, латинские и русские буквы, символ @);

— 3 управляющих (переводы строки LF в стиле UNIX).

Одинаково ли количество байтов для представления одного символа кодовой таблицы (печатного или управляющего, как LF) в различных кодировках?

Одинаково ли шестнадцатеричное представление пробела в различных кодировках? Цифр? Латинских букв? Русских букв?

Задание ЛС.№5. Для ПМ-3* и ПИН-*Д — бонус +1 или +10 баллов; для очных групп ПИН-3* 0 баллов: включено в Л2, здесь не засчитывается. Для файла $(№ - 1)\%9$ из папки `labs-files/Варианты 2 - определение кодировки простого текста` (далее — файл W):

— определите, является ли W простым текстом на русском языке в одной из стандартных кодировок (один из вариантов представляет собой нерусскоязычный текст);

— если да — определите кодировку и декодируйте в UTF-8.

Если для определения кодировки используется существующая программа определения кодировок, задание засчитывается на $+1$ балл. Для получения $+10$ баллов необходимо провести частотный анализ самостоятельно.

Для проведения частотного анализа выполните следующие шаги.

1. Разработайте программу для определения частот октетов (байтов x86) в заданном файле (это может быть как скрипт-однострочник, использующий

стандартные утилиты GNU/Linux, так и проект на любом языке программирования в любой среде).

Хотя в этом задании далее анализироваться будут файлы в формате простого текста — программа, анализирующая распределение октетов, *должна корректно обрабатывать любые файлы.*

2. Рассчитайте частоты появления октетов в файлах, являющихся осмысленным русскоязычным текстом достаточного объёма в различных кодировках (`labs-files/Файлы в формате простого текста - кодировки разные`). Определите:
   — четыре наиболее частых октета среди всех используемых;
   — четыре наиболее частых октета, не являющихся кодами печатных символов ASCII; для однобайтовых кодировок сопоставьте соотношение их частот с частотами символов русского языка.

   Обратите внимание на распределение октетов многобайтовых кодировок Unicode (UTF-8, UTF-16, UTF-32).

3. Рассчитайте частоты появления октетов в файле W. Определите, аналогично п. 2, четыре наиболее частых октета среди всех и четыре — среди не являющихся кодами печатных символов ASCII.
   Сопоставьте их с результатами п. 2 и с частотами символов русского языка. Определите наиболее вероятную кодировку или нерусскоязычность текста.

4. Если по результатам п. **??** файл W является русскоязычным текстом в кодировке X — декодируйте W из X в UTF-8 любой утилитой перекодировки. Проверьте корректность результата.

ЛС.2. Кодировки и кодовые таблицы русского языка

Кодировки и кодовые таблицы

Строго говоря, необходимо различать понятия:

— *кодовая таблица* или таблица кодов — соответствие символов кодам в каком-то диапазоне;
— *кодировка* — представление кода символа в памяти или на диске.

Но обычно их смешивают и называют для краткости «кодировкой» совокупность кодовой таблицы и собственно кодировки.

Это в большинстве случаев не приводит к недопониманию, так как:

— кодировкам UTF-8, UTF-16, UTF-32 всегда соответствует одна и та же универсальная кодовая таблица Unicode, содержащая *все* национальные алфавиты;
— национальным кодовым таблицам (KOI8-R, IBM CP866, Windows-1251 и т. п.) всегда соответствует одна и та же кодировка: код записывается октетом «как есть».

Таким образом, *однобайтовыми кодировками* на практике называются *кодовые таблицы,* сопоставляющие символы кодам в диапазоне 0–255 (0x00–0xFF); коды символов таких таблиц всегда записывается одним октетом (байтом x86). В настоящее время все такие кодовые таблицы сопоставляют кодам 0–127 те же символы, что и кодовая таблица ASCII (являются расширениями кодовой таблицы ASCII). Коды в диапазоне 128–255 описывают национальные кодовые страницы.

Кодовая таблица ASCII сопоставляет символы кодам в диапазоне 0–127 (0x00–0x7F) (см. приложение В). При этом понятие «однобайтовая кодировка ASCII» *не определено,* и в зависимости от контекста может описывать как исторические способы записи семибитных ASCII-кодов восемью битами (старший бит мог использоваться для контроля чётности, дублировать один из семи младших или всегда быть нулевым), так и Latin-1 (ISO 8859-1), и, чаще, ту однобайтовую кодировку, которая используется на компьютере говорящего.

Кодовая таблица Unicode сопоставляет символы кодам `0–0x10 FFFF` (кодам 0–127 соответствуют символы ASCII) то есть не может быть представлена однобайтовой кодировкой. Не всем Unicode-кодам соответствуют символы; так, коды `0xD800–0xDFFF` зарезервированы для представления суррогатных пар в UTF-16.

Для кодирования символов Unicode используются три основные кодировки.

1. UTF-8, кодировка переменной длины (изначально от 1 до 6 октетов, позже ограничили до 4) для узких строк:
   — `0xxx xxxx` — ASCII-символы (коды от 0 до $127 = $ `0x7F`) представляются одним байтом, равным ASCII-коду (старший бит — 0).

   Прочие, включая 128–255, представляются не менее чем двумя байтами. Старшие биты первого из k байтов содержат k единиц, затем следует разделитель 0, затем — старшие биты Unicode-кода. Старшие биты последующих байтов — 10, так что представление символа в UTF-8 не равно его коду Unicode:
   — `110x xxxx 10xx xxxx` — символы с Unicode-кодами от $128 = $ `0x80` до `0x7FF`, в том числе греческие, русские и арабские буквы, представляются двумя байтами;
   — `1110 xxxx 10xx xxxx 10xx xxxx` — далее символы до `0xFFFF`, в том числе основные китайские и японские иероглифы — тремя;
   — `1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx` — прочие символы (с запасом — до `0x1F FFFF`) могут быть представлены четырьмя байтами.

   UTF-8 — наиболее распространённая в настоящее время кодировка Unicode.
2. UTF-16, кодировка переменной длины (от 1 до 2 элементов) для широких строк из двухоктетных (16-битных) элементов:
   — символы с Unicode-кодами `0x0000–0xFFFF` записываются одним 16-битным элементом «как есть»; символов с кодами `0xD800–0xDFFF` не существует;
   — символы `0x1 0000–0x10 FFFF` — двумя элементами: первый лежит в диапазоне `0xD800–0xDBFF`, второй — `0xDC00–0xDFFF` (суррогатной парой);

UTF-16 — старейшая кодировка Unicode (первые версии Unicode умещались в один 16-битный элемент). UTF-16, в отличие от UTF-8 и UTF-32, не позволяет записать коды свыше `0x10 FFFF`.

3. UTF-32, кодировка постоянной длины (один 32-битный элемент с запасом вмещает все Unicode-коды) для широких строк из четырёхоктетных (32-битных) элементов.

Кодировки UTF-16 и UTF-32 имеют варианты, соответствующие разному порядку байтов в двух- или четырёхоктетном элементе (LE/BE); по умолчанию подразумевается порядок байтов платформы (LE для x86).

В MS Windows «Unicode» обозначает устаревшую версию кодировки UTF-16 (включающую только 16-битные символы, но не суррогатные пары), что неверно. Однобайтовая кодировка в MS Windows (для русского языка используется кодовая страница Windows-1251) обозначается «ANSI».

Представление русского языка

В папке `labs-files/Файлы в формате простого текста - кодировки разные` представлены основные кодировки/таблицы для русского языка:

1. Многобайтовые кодировки универсальной кодовой таблицы Unicode:
   - — UTF-8 — суффикс имени файла utf8;
   - — UTF-16 — суффикс utf16;
   - — UTF-32 — суффикс utf32.
2. Однобайтовые расширения ASCII:
   - — КОИ-8 (код обмена информацией 8-битный) для русского алфавита (KOI8-R), использовавшаяся в России до широкого распространения MS DOS/MS Windows — суффикс koi8r;
   - — ISO 8859-5, разработанная ISO и IEC и одно время считавшаяся стандартной, но не использовавшаяся — суффикс iso;
   - — IBM CP866 (альтернативная кодировка ГОСТ), использовавшаяся в русифицированной MS DOS — суффикс dos;
   - — Windows-1251 (CP1251), использующаяся в русифицированной MS Windows — суффикс windows;
   - — MacCyrillic, использующаяся в Mac OS X — суффикс maccyrillic.

   Из-за совпадения кодов наиболее частотных русских букв утилиты распознавания кодировок регулярно путают Windows-1251 и MacCyrillic; для различения этих кодировок необходим дополнительный анализ.

ЛС.3. Вопросы

1. Для чего нужен шестнадцатеричный редактор?
2. Какие функции libc используются для чтения/записи бинарных файлов?
3. Известно, что файл содержит осмысленный русскоязычный текст в одной из представленных в данной работе кодировок. Можно ли, используя толь-

ко шестнадцатеричный редактор, без частотного анализа, отличить UTF-8, UTF-16, UTF-32: а) друг от друга, б) от однобайтовой кодировки? По каким признакам?