Лекция 9

Разделяемый загрузочный файл

- Не является программой – не имеет точки входа, предназначен для использования другими программами
- Содержит таблицы экспорта и импорта
- Может размещаться в памяти с произвольного адреса – позиционно независимый код (PIC)
- Подгружается в адресное пространство программы загрузчиком при запуске программы
- Может подгружаться явно с помощью API

Использование .SO файлов

- Примеры:
  - /lib/libc.so – стандартная библиотека
  - /lib/libm.so – математические функции
- Утилита ldd – выводит список зависимостей исполняемого файла от библиотек
- Файл LIB.so используется при компоновке для обработки внешних ссылок, но в исполняемый файл записывается имя LIB.so.VER
- LIB.so.VER (например, libc.so.6) подгружается при запуске программы
- Решение проблемы “DLL hell”

Запуск исполняемого файла

- Исполняемый ELF-файл, скомпонованный динамически, запускается на выполнение в несколько этапов:
  - Секция .interp содержит путь к “интерпретатору” - динамическому загрузчику
  - Ядро ОС грузит в память исполняемый файл и динамический загрузчик
  - Управление передается загрузчику
  - Загрузчик анализирует зависимости и подгружает требуемые динамические библиотеки
  - Управление передается загруженной программе

Кросс-компиляция

- Host-платформа — платформа, на которой работает компилятор
- Target-платформа — платформа, для которой компилируется код
- Если host != target — кросс-компиляция, если код для target не может быть запущен на host
- Компиляция для x86 на x86_64 — это не кросс-компиляция

Triple

- Идентифицируют платформу
- В целом имеют вид: machine-vendor-operatingsystem
- Часто vendor опускается
- Часто vendor — это pc или unknown

Примеры triple

- i686-redhat-linux
- x86_64-redhat-linux
- arm-linux-gnueabihf
- avr
- sparc-sun-solaris2.10

API

- API – Application Programming Interface
  - Определения типов данных, констант, функций, классов и т. д. в терминах языка программирования для решения определенной задачи
- Примеры:
  - POSIX API – определения в терминах языка Си для взаимодействия с ядром операционной системы, соответствующей POSIX
  - WINAPI – взаимодействие с ядром Windows, язык Си
  - Другие языки программирования могут предоставлять “привязки” (bindings) к API

ABI

- ABI – Application Binary Interface
  - Интерфейс между компонентами системы на уровне бинарного (скомпилированного) кода
  - Стандартизирует размеры типов, выравнивания, формат бинарных данных, способ передачи параметров и т. п.
- Примеры:
  - Linux System Call ABI
  - Itanium C++ ABI
  - Win32 ABI

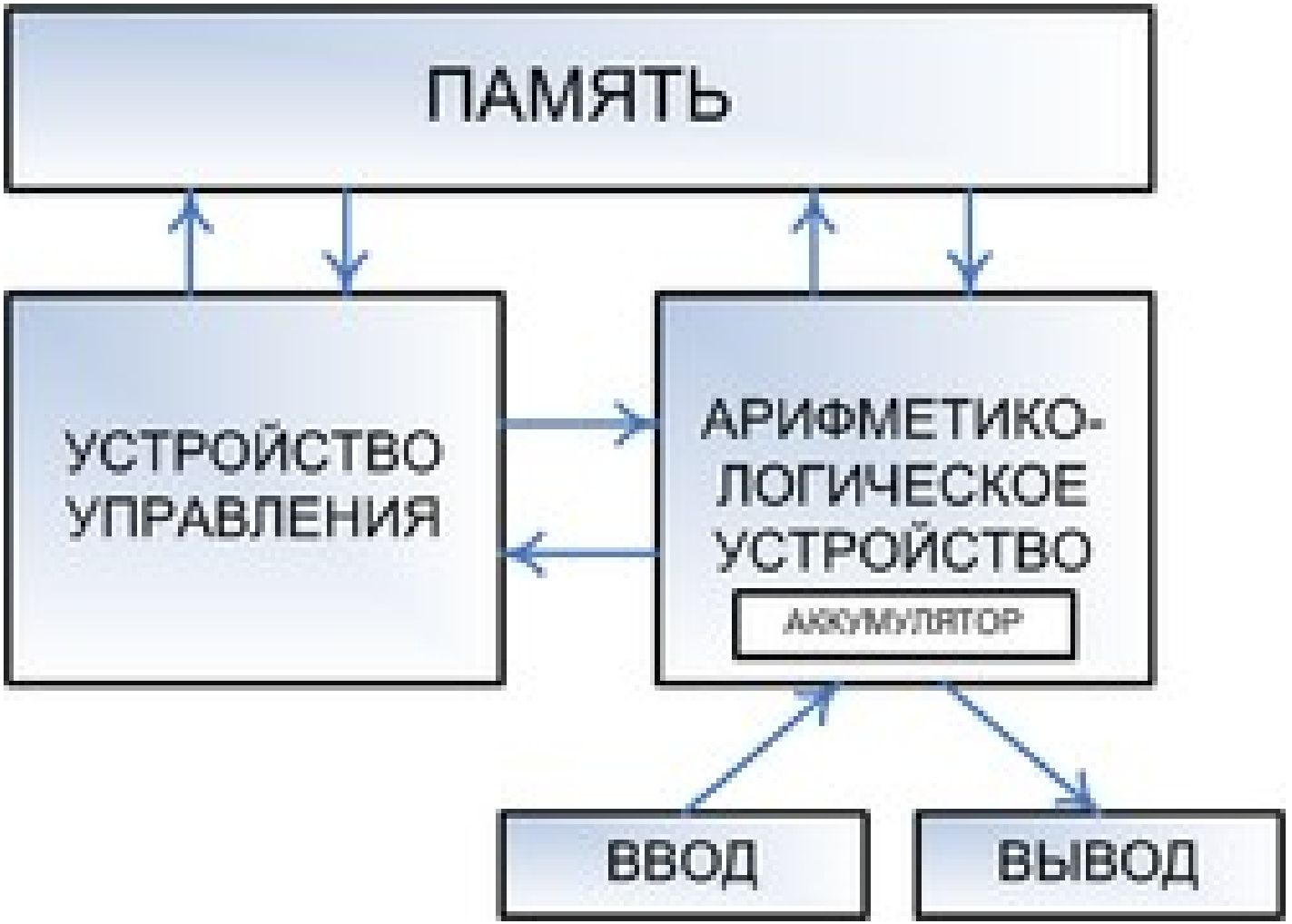
API vs ABI

- API – обеспечивает совместимость и переносимость программ на одном языке программирования на уровне исходного кода
- ABI – обеспечивает совместимость программ на разных языках программирования, скомпилированных разными компиляторами, для разных версий операционной системы и т. п. - не требуется перекомпиляция

Принципы фон Неймана (von Neumann architecture)

- Концептуальная модель цифрового компьютера общего назначения (1945)
- Лежит в основе (концептуально) современных процессоров
  - Адресность
  - Однородность памяти
  - Программное управление
  - Двоичное кодирование

Концептуальная схема

- 

Принципы фон Неймана

- Адресность
  - Оперативная память (ОЗУ) – память произвольного доступа (RAM), в любой момент времени доступна любая ячейка
  - ОЗУ разбито на ячейки фиксированного размера
  - Каждая ячейка имеет фиксированный номер – адрес, работа с ОЗУ – по адресам
  - При необходимости ячейки могут группироваться
- Двоичное кодирование

Однородность памяти

- И программа, и данные хранятся в одной памяти
- Только по ячейке памяти невозможно определить, что в ней хранится (память не тегирована)
  - Например, один и те же 4 байта могут быть целым числом, числом float, символом UCS4, указателем, инструкцией процессора
- “Смысл” значения в ячейке определяется только тогда, когда процессор обращается к ней, и может меняться во времени

Программное управление

- Программа кодируется в виде инструкций процессора
- Программа хранится в оперативной памяти
- Инструкции процессора располагаются в памяти последовательно
- Инструкции выполняются последовательно, но порядок выполнения можно изменить
- Шаги выполнения инструкции:
  - Чтение инструкции из памяти
  - Декодирование
  - Чтение аргументов из памяти
  - Выполнение операции
  - Сохранение результата

Модификации

- Гарвардская архитектура – несколько отдельных адресных пространств: для кода программы, для данных, для ввода-вывода
  - В основном используется в low-end микроконтроллерах
  - Разные инструкции для чтения из пространства кода и работы с пространством данных
- Современные ОС, как правило, запрещают модификацию кода программы на лету, исполнение кода в пространстве данных
- Многоядерность и прогопроцессорность

Язык ассемблера

- Ассемблер – программа, переводящая текстовый формат инструкций процессора в объектный код
- Язык ассемблера - “текстовый формат” представлений инструкций процессора
- Каждая процессорная архитектура (x86, x64, ARM, ARMv8, MIPS, …) имеет свой набор инструкций
- Ассемблеры достаточно похожи друг на друга

Области применения

- Программирование микроконтроллеров (но обычно Си)
- Низкоуровневые части ядер ОС и драйверов (например, точка входа в ядро Linux при системном вызове или прерывании)
- Генераторы кода компиляторов, бинарных трансляторов, интерпретаторов
- Исследование бинарного кода (антивирусы и т. п)

Наши цели изучения

- Понимание ассемблера позволяет лучше понять архитектуру процессора
- Изучение кода, сгенерированного компилятором, полезно (иногда необходимо) для понимания оптимизаций
- Понимание ассемблера позволяет лучше понять влияние архитектуры компьютера на операционные системы и языки программирования

Ассемблер x86 (i386)

- На лекциях и семинарах рассматриваться не будет, но вы можете выполнять задания на ассемблере x64 (x86_64)
- X86 – наиболее доступная платформа, поэтому выбрана она
- Для инструкций x86 существует несколько форм записи: Intel ASM, nasm, AT&T asm, мы будем использовать AT&T asm – синтаксис GNU assembler по умолчанию

Inline assembly

- Gcc, clang, MSVC поддерживают написание вставок на ассемблере непосредственно в коде на Си/Си++
  - asm(“nop”);
- Синтаксис не стандартизирован, каждый компилятор по-своему решает задачу сочетания кода на си и ассемблере
- Рассматривать не будем

GNU assembler

- Комментарии как в Си (/* */ или //)
- Целые числа, символьные константы, вещественные константы как в Си (10, 0xa, '\n', 10.0)
- Строки как в Си (со всеми \ где нужно, но без неявного \0 в конце)
- Каждая инструкция процессора на отдельной строке
- Используем TAB для разделения полей инструкции

Инструкции

- Каждая инструкция записывается на отдельной строке
- Инструкция может быть “помечена”:
  LABEL:
  (после имени метки стоит двоеточие)
- Директива ассемблера – управляет трансляцией, инструкция – транслируется в машинный код
- Общий вид инструкции или директивы
  OPCODE PARAMS

Компиляция

- Файл называем с суффиксом .S или .s
  - .S, если нужен препроцессор Си
- Компиляция с помощью as
  - as FILE.s -o FILE.o -g -a
- Компиляция с помощью gcc
  - gcc -m32 FILE.S -c -g
- Чтобы отключить стандартную библиотеку Си и startup код:
  - gcc -m32 FILE.S -oFILE -g -nostdlib

Структура единицы трансляции

- Программа состоит из секций – логических частей программы
- Компоновщик объединяет содержимое секций из входных объектных файлов, размещает секции в исполняемом файле
- Стандартные секции (минимальный набор)
  - .text – код программы и read-only data
  - .data – глобальные переменные
  - .bss – глобальные переменные, инициализированные нулем

Дополнительные секции

- Можно определять секции с произвольными именами
- Стандартные дополнительные секции:
  - .rodata
    .section .rodata, “a”
- Нестандартные секции:
  - .string для размещения строк:
    .section .string, “aMS”, @progbits, 1

Правила использования секций

- Программный код должен размещаться в секции .text
- Константы и константные строки могут размещаться в .text или в .rodata
- Глобальные переменные размещаются в .data или .bss

Метки (labels, symbols)

- Метки – это символические константы, значение которых известно при компиляции или компоновке программы
  - Метка как адрес, по которому размещается инструкция при выполнении программы
  - Метка как константное значение
- По умолчанию метки видны только в текущей единице компиляции (в том числе объявленные после использования)
- Чтобы сделать метку доступной компоновщику используется .global NAME

Точка входа в программу

- Программа должна иметь точку входа – метку, на которую передается управление в начале выполнения программы
- Если компилируем без стандартной библиотеки (-nostdlib), точка входа должна называться _start и должна экспортироваться (.global _start)
- Если компилируем со стандартной библиотекой, точка входа называется main и должна экспортироваться (.global main)