Вопросы. Архитектура и программирование.

(SE) Архитектура компьютера: архитектура фон Неймана, гарвардская архитектура.

В 1940-х годах в процессе работы над первыми электронными вычислительными машинами Джон фон Нейман и его коллеги определили ряд принципов построения вычислительных машин.

Принципы фон Неймана:

1. *Принцип однородности памяти*

   Команды и данные хранятся в одной и той же памяти и внешне в памяти неразличимы. Распознать их можно только по способу использования; то есть одно и то же значение в ячейке памяти может использоваться и как данные, и как команда, и как адрес в зависимости лишь от способа обращения к нему. Это позволяет производить над командами те же операции, что и над числами, и, соответственно, открывает ряд возможностей. Так, например, команды одной программы могут быть получены как результат исполнения другой программы. Эта возможность лежит в основе трансляции — перевода текста программы с языка высокого уровня на язык конкретной вычислительной машины.

2. *Принцип адресности*

   Структурно основная память состоит из пронумерованных ячеек, причём процессору в произвольный момент доступна любая ячейка. Двоичные коды команд и данных разделяются на единицы информации, называемые словами, и хранятся в ячейках памяти, а для доступа к ним используются номера соответствующих ячеек — адреса.

3. *Принцип программного управления*

   Все вычисления, предусмотренные алгоритмом решения задачи, должны быть представлены в виде программы, состоящей из последовательности управляющих слов — команд. Каждая команда предписывает некоторую операцию из набора операций, реализуемых вычислительной машиной. Команды программы хранятся в

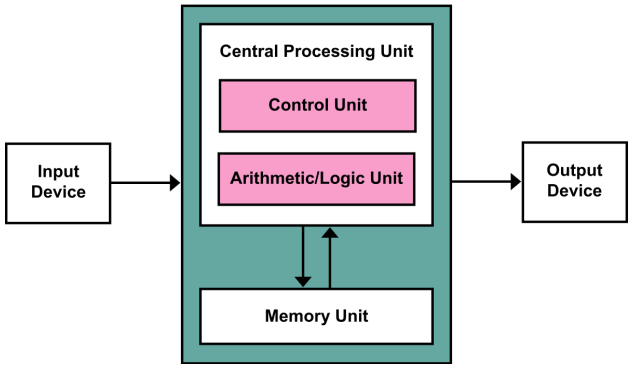

Рис. 1: Архитектура фон Неймана

последовательных ячейках памяти вычислительной машины и выполняются в естественной последовательности, то есть в порядке их положения в программе. При необходимости, с помощью специальных команд, эта последовательность может быть изменена. Решение об изменении порядка выполнения команд программы принимается либо на основании анализа результатов предшествующих вычислений, либо безусловно.

Узкое место архитектуры фон Неймана

Совместное использование шины для памяти программ и памяти данных приводит к узкому месту архитектуры фон Неймана, а именно ограничению пропускной способности между процессором и памятью по сравнению с объёмом памяти. Из-за того, что память программ и память данных не могут быть доступны в одно и то же время, пропускная способность канала «процессор-память» и скорость работы памяти существенно ограничивают скорость работы процессора — гораздо сильнее, чем если бы программы и данные хранились в разных местах.

Данная проблема решается совершенствованием систем кэширования, что в свою очередь усложняет архитектуру систем и увеличивает риск возникновения побочных ошибок (например, проблема когерентности памяти).

Гарвардская архитектура

Гарвардская архитектура — архитектура ЭВМ, разработанная в конце 1930-х годов в Гарвардском университете. Отличительными признаками данной архитектуры являются:

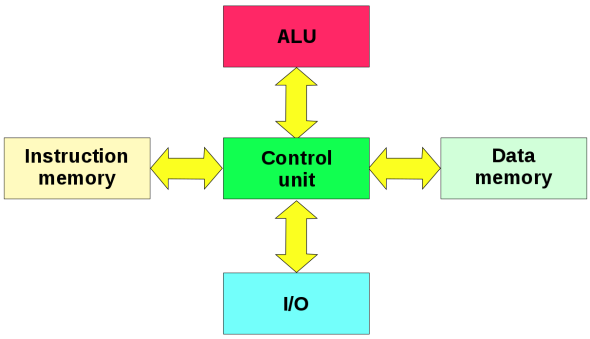

Рис. 2: Гарвардская архитектура

- хранилище инструкций и хранилище данных представляют собой разные физические устройства;
- канал инструкций и канал данных также физически разделены.

В архитектуре фон Неймана процессор в каждый момент времени может либо читать инструкцию, либо читать/записывать единицу данных из/в памяти. Оба действия одновременно происходить не могут, поскольку инструкции и данные используют один и тот же поток (шину).

В компьютере с использованием гарвардской архитектуры процессор может считывать очередную команду и оперировать памятью данных одновременно и без использования кэш-памяти.

Исходя из физического разделения шин команд и данных, разрядности этих шин могут различаться и физически не могут пересекаться.

(SE) Объектно-ориентированное программирование. Основные принципы.

Объектно-ориентированное программирование — методология программирования, основанная на представлении программы в виде совокупности взаимодействующих *объектов*, каждый из которых является экземпляром определённого *класса*, а классы образуют *иерархию наследования*.

Объект — это сущность, которой можно посылать сообщения и которая может на них реагировать, используя свои данные. Объект — это экземпляр класса. Данные объекта скрыты от остальной программы.

Класс — в объектно-ориентированном программировании, представляет собой шаблон для создания объектов, обеспечивающий начальные значения состояний: инициализация полей-переменных и реализация поведения функций или методов.

Объекты внутри программы взаимодействуют с помощью **сообщений**. Во многих языках программирования используется концепция «*отправка сообщений как вызов метода*» — объекты имеют доступные извне методы, вызовами которых и обеспечивается взаимодействие объектов.

Основные принципы ООП.

1. **Абстракция**

   Абстракция в объектно-ориентированном программировании — это использование только тех характеристик объекта, которые с достаточной точностью представляют его в данной системе. Основная идея состоит в том, чтобы представить объект минимальным набором полей и методов и при этом с достаточной точностью для решаемой задачи.

2. **Инкапсуляция. Две трактовки**

   Инкапсуляция — свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе. В общем случае в разных языках программирования термин *инкапсуляция* относится к одной или обеим одновременно следующим нотациям:

   - механизм языка, позволяющий ограничить доступ одних компонентов программы к другим;
   - языковая конструкция, позволяющая связать данные с методами, предназначенными для обработки этих данных.

3. **Наследование**

   Наследование — свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствованной функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперк-

лассом. Новый класс — потомком, наследником, дочерним или производным классом.

4. **Полиморфизм**

   Полиморфизмом называется возможность единообразно обрабатывать разные типы данных. В языке C++ можно выделить следующие механизмы полиморфизма:

   - Перегрузка функций (не обязательно ООП?). Выбор функции происходит в момент компиляции на основе типов аргументов функции, *статический полиморфизм.*
   - Виртуальные методы. Выбор метода происходит в момент выполнения на основе типа объекта, у которого вызывается виртуальный метод, *динамический полиморфизм.*