Роль RISC-V

Оглавление

Введение

В этой главе описывается роль, которую выполняет архитектура RISC-V, и ее место в современном компьютерном мире. Рассматривается процесс компиляции программы и ее выполнение на процессоре RISC-V.

В этой главе вы узнаете:

- роль компиляторов и ассемблеров;
- роль архитектуры набора команд (ISA);
- общие характеристики RISC-V по сравнению с другими ISA.

Программное обеспечение, компиляторы и центральный процессор

Скорее всего, у вас есть опыт разработки программ на таких языках, как Python, JavaScript, Java, C++ и др. Эти языки являются переносимыми и программы, разработанные на них, могут выполняться практически на любом аппаратном обеспечении процессора. Процессоры не выполняют команды этих языков программирования непосредственно напрямую. Они выполняют *машинные команды*, закодированные в биты в соответствии с *архитектурой набора команд* (ISA). К наиболее популярным ISA относятся x86, ARM, MIPS, RISC-V и т.д.

Компилятор выполняет работу по переводу исходного кода программы в *двоичный* файл или *исполняемый* файл, содержащий машинные команды для определенного ISA. Операционная система (и, возможно, среда выполнения) выполняет работу по загрузке двоичного файла в память программ для выполнения процессором, понимающими конкретный ISA. Ниже приведена схема, описывающая этот процесс.

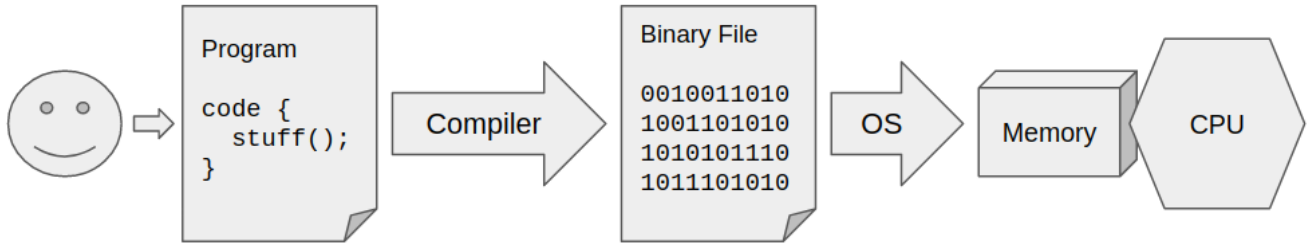

Рисунок 1 – Разработка и выполнение программного обеспечения

Двоичный файл легко интерпретируется аппаратурой, но для человека это нелегко. ISA определяет понятную человеку форму каждой команды так же, как и преобразование этих читаемых *ассемблерных команд* в биты. Помимо создания двоичных файлов, компиляторы могут генерировать *ассемблерный код*. *Ассемблер* позволяет скомпилировать ассемблерный код в двоичный файл. Помимо того, что ассемблер обеспечивает преобразование вывода компилятора программы в человеко-читаемый формат, на ассемблере можно разрабатываться программы и непосредственно. Это полезно для тестирования оборудования и других ситуаций, когда требуется прямое низкоуровневое взаимодействие. В этом курсе используются тестовые программы на ассемблере для отладки проекта RISC-V. Весь описанный процесс работы с ассемблерным кодом можно представить в виде схемы ниже.

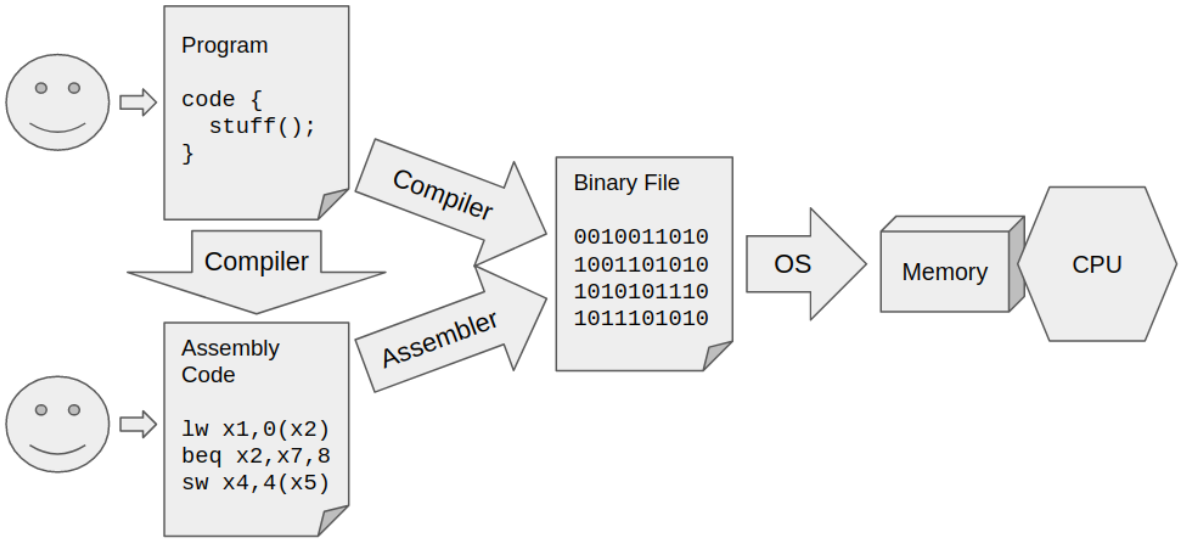

Рисунок 2 – Разработка и выполнение кода на ассемблере

Обзор RISC-V

В этом курсе вы создадите простой процессор, поддерживающий RISC-V ISA. RISC-V очень быстро завоевала популярность благодаря своей открытой архитектуре – отсутствию патентной защиты и ориентированности на сообщество. Следуя примеру RISC-V, MIPS и PowerPC впоследствии также стали открытыми.

RISC-V также популярна своей простотой и расширяемостью, что делает ее отличным выбором для данного курса. RISC означает «архитектуру процессора с сокращенным набором команд» и противопоставляется «архитектуре процессора со сложным набором команд» (CISC). RISC-V (произносится как «риск файв») – пятая ISA в серии RISC от Калифорнийского университета в Беркли. Вам предстоит реализовать основные команды базового набора команд RISC-V (RV32I), который содержит 47 команд. Из них вы реализуете 31 (из оставшихся 16, 10 связаны с окружающей системой, а 6 обеспечивают поддержку хранения и загрузки небольших значений в память и из памяти).

Как и другие RISC (и даже CISC) ISA, RISC-V – это аккумуляторная архитектура. Она содержит регистровый файл, который может хранить до 32 значений (на самом деле – 31). Большинство инструкций считывают данные из регистрового файла и записывают их обратно. Инструкции загрузки и хранения переносят значения между памятью и регистровым файлом.

Инструкции RISC-V могут содержать следующие поля:

- **opcode**

  Содержит код классификации команд и определяет, какие из оставшихся полей необходимы, и то, как они располагаются, или кодируются, в оставшихся битах команд.

- **function field** (funct3/funct7)

  Описывает точную функцию, выполняемую командой, если она не полностью определена в коде операции (opcode).

- **rs1/rs2**

  Индексы (0-31), идентифицирующие номера регистров в регистровом файле, содержащие значения операндов аргумента, над которыми работает команда.

- **rd**

  Индекс (0-31) регистра, в который записывается результат выполнения команды.

- **Immediate**

  Значение константы (непосредственного операнда), содержащейся в самих битах команды. Это значение может служить смещением для индексации в память или значением, над которым нужно выполнить операцию (вместо значения регистра, задаваемого в rs2).

Все команды являются 32 битными. Кодировка R-типа обеспечивает общее расположение полей команд, используемых всеми типами команд. Команды R-типа не имеют непосредственного значения. Другие типы команд используют подмножество полей R-типа и могут содержать непосредственный операнд, расположенное в оставшихся битах команд. Ниже приведен рисунок, демонстрирующий отличие в типах команд.

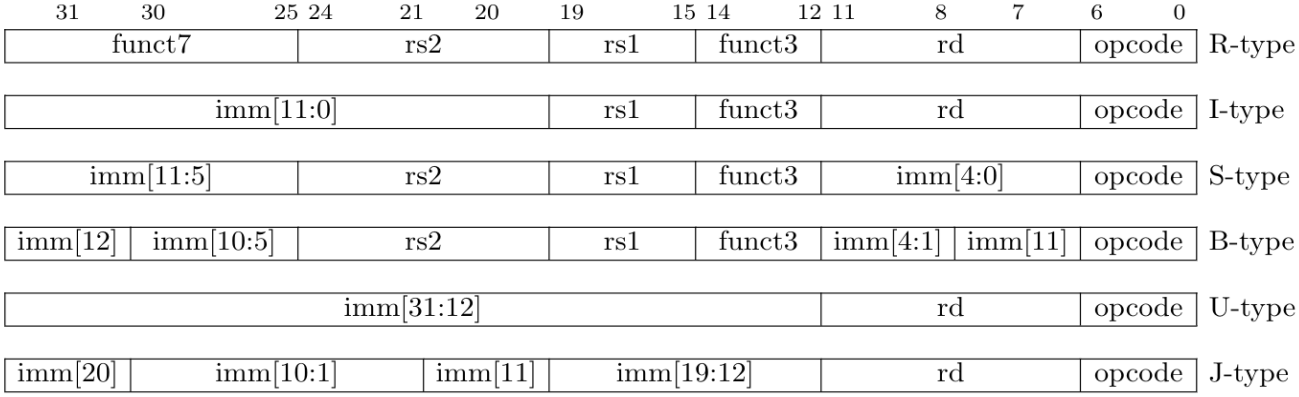

Рисунок 3 – Форматы базовых команд (взято из спецификаций RISC-V [1])

Список источников

1. RISC-V Specifications: [Электронный ресурс] // RISC-V. URL: https://riscv.org/technical/specifications/