ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ
им. А.Н. ТИХОНОВА

Методические указания по компиляции RISC-V кода

Москва 2025

1 Введение

Программы, написанные на языке ассемблера RISC-V можно запускать разными способами:

– используя кросс-компилятор и компьютер с архитектурой RISC-V (например, микроконтроллер esp32 или mik32 Амур),
– эмулируя ОС на базе RISC-V процессора (QEMU),
– используя сторонние утилиты (RARS).

Ниже приведены рекомендации по работе с кодом RISC-V на разных платформах.

2 Работа в QEMU

QEMU позволяет эмулировать работу процессора отличного от процессора хост-системы. Благодаря этому можно эмулировать работу компьютера на процессоре RISC-V.

В книге С. Смита «Программирование на языке ассемблера RISC-V» в главе 1 «Начало работы» имеется инструкция по настройке и запуску Ubuntu RISC-V в QEMU в разделе «Программирование Hello World в эмуляторе QEMU». Кроме того, в качестве дополнения к этой информации были разработаны рекомендации по использованию QEMU и Docker для эмуляции работы Ubuntu RISC-V:

<ссылка на не редактируемые материалы>.

2.1 Подготовка среды в Ubuntu

Если в качестве эмулированной среды был использован Docker-образ, собранный по Dockerfile из методических указаний, данный раздел можно пропустить.

При использовании QEMU напрямую или Docker для эмуляции компьютера ОС Ubuntu на RISC-V необходимо установить пакет **<build-essential>** для возможности компилировать RISC-V код:

Листинг 1 – Установка пакета build-essential в Ubuntu RISC-V

```
apt-get update && apt-get -y upgrade && \
apt-get -y install build-essential
```

Кроме набора инструментов компиляции для редактирования документов удобно использовать пакет **<vim>** или **<nano>**:

Листинг 2 – Установка пакетов vim и nano

```
apt-get -y install vim nano
```

2.2 Компиляция RISC-V кода

Для успешной сборки подготовленного ассемблер кода RISC-V достаточно использовать следующую команду:

Листинг 3 – Сборка программы на ассемблере RISC-V

```
gcc -nostartfiles example.S -o example.elf
```

Флаг **<-nostartfiles>** необходимо использовать, если в программе определяется метка **<_start>**, иначе произойдет ошибка линковки с переопределением системной метки.

3 Работа с кросс-компилятором

Написанную на языке ассемблера RISC-V программу можно собрать в исполняемый файл на компьютере, который не использует RISC-V процессор, если для него существует пакет кросс-компилятора RISC-V.

Для кросс-компиляции рекомендуется использовать пакет **< riscv-none-elf-gcc-xpack>**, инструкцию по установке которого можно найти на официальном сайте (https://xpack-dev-tools.github.io/riscv-none-elf-gcc-xpack/docs/install/).

При использовании кросс-компилятора программу на RISC-V ассемблере можно собрать следующей командой:

Листинг 4 – Сборка программы на ассемблере RISC-V кросс-компилятором

```
riscv-none-elf-gcc -nostartfiles example.S -o example.elf
```

Программа, собранная командой выше будет успешно запускаться на компьютере с процессором RISC-V, однако стоит понимать, что для кросс-компиляции программ под микроконтроллер могут потребоваться дополнительные действия (например, включение в сборку специальных библиотек).

3.1 Сборка программы под микроконтроллер MIK32 Амур

Для сборки программ на языке ассемблера RISC-V и их успешном запуске на микроконтроллере MIK32 Амур кроме самого кросс-компилятора потребуются специальные библиотеки, определяющие функционал для работы с периферийными блоками микроконтроллера, а также библиотеки, содержащие заголовочные файлы, стартовые скрипты и скрипты линковки.

Все вышеописанные зависимости и их установка подробно описаны в методических указаниях по кросс-компиляции программ под MIK32 Амур:

<ссылка на не редактируемые материалы>.

Кроме того, для быстрой сборки программ под микроконтроллер были подготовлены правила сборки, поддерживающие:

– быструю установку пакетов кросс-компилятора и загрузчика **<openOCD>** в зависимости от архитектуры хост-системы,
– автоматическую установку библиотек и проектов со скриптами памяти микроконтроллера,
– корректное включение исходников библиотек, скриптов линковки, определение флагов компиляции,
– правила для быстрой прошивки микроконтроллера.

Данные сборочные правила доступны на github:
https://github.com/mt-omarov/mik32-guide/tree/master.

Для использования проекта необходимо:

– установить проект,
– заменить код в папке src/ на свой код,
– выполнить команду `**make**` и следовать инструкциям при возникновении ошибок,
– выполнить команду `**make upload**` для прошивки микроконтроллера.

4 Использование утилиты RARS

Для запуска и отладки кода на ассемблере RISC-V можно использовать утилиту RARS («RISC-V Assembler and Runtime Simulator»). Необходимо скачать последний доступный релиз (файл формата «**.jar**») по ссылке https://github.com/TheThirdOne/rars/releases.

Для запуска утилиты необходимо установить **<Java Runtime Environment>** для своей операционной системы https://www.java.com/en/download/manual.jsp.

После этого необходимо запустить файл «**.jar**» двойным кликом и в качестве программы для запуска выбрать «Java (™) Platform SE binary».

Исполняемый код необходимо вставить в окно «Edit», компиляция выполняется с помощью нажатия «Run → Assemble» (рис. 1).

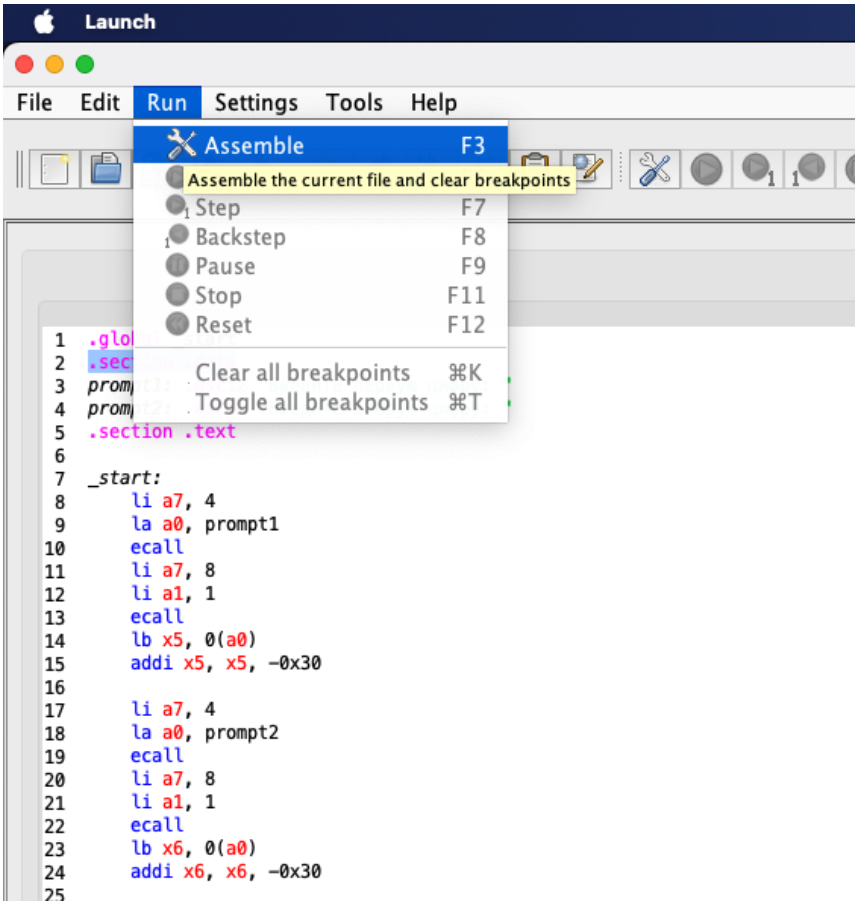

Рисунок 1 – Компиляция кода в RARS

Для запуска программы необходимо в окне «Execute» нажать на «Run» (рис. 2). После запуска программы становятся доступны состояния регистров в отдельном окне справа.

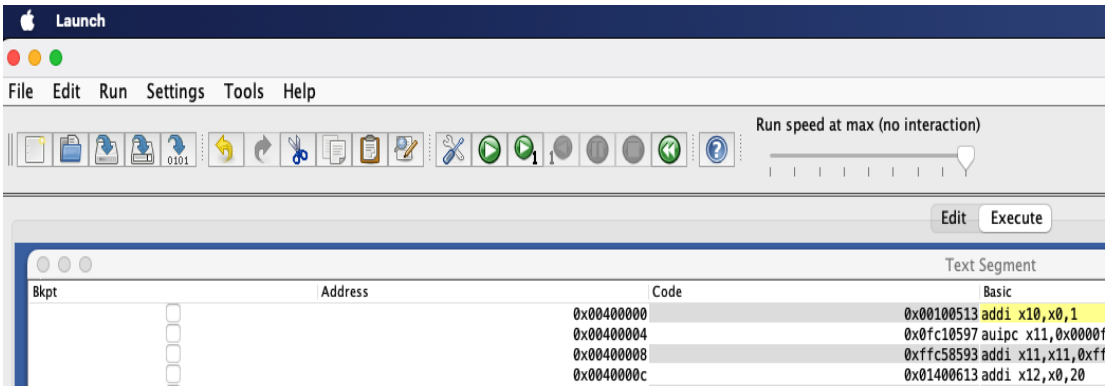

Рисунок 2 – Запуск программы в RARS