GNU RISC-V 툴체인

차 례

제 1 절 이 문서에 대하여

이 문서는 **IoT 장치 및 웨어러블 기기에 적합한 RISC-V 확장형 ISA 기반 경량 프로세서 기술 개발** 과제에서 GNU RISC-V 툴체인을 다운로드 받고 컴파일하는 과정과 생성된 컴파일러를 이용하여 사용자 응용프로그램을 컴파일하고 실행하는 과정을 설명한다.

제 2 절 GNU RISC-V 툴체인 설치

2.1 Github를 이용한 GNU RISC-V 툴체인 소스 코드 받기

아래와 같이 github에서 소스 코드를 다운 받는다.

```
$ git clone https://github.com/riscv/riscv-gnu-toolchain
```

2.2 필요 프로그램 설치

GNU RISC-V 툴체인을 설치하기 위해서는 여러 가지 프로그램이 필요하며, 이 프로그램들은 다음과 같은 명령을 수행하여 설치가 가능하다. 아래는 Ubuntu 운영체제에서의 예이다. 운영체제에 따라 적당한 명령을 수행한다.

```
$ sudo apt-get install autoconf automake autotools-dev curl python3 libmpc-dev libmpfr-dev
        libgmp-dev gawk build-essential bison flex texinfo gperf libtool patchutils bc
        zlib1g-dev libexpat-dev
```

2.3 컴파일 및 설치

현재 배포되고 있는 GNU RISC-V Toolchain은 사용하는 라이브러리에 따라 Newlib과 Linux 모드로 컴파일이 가능하다. Newlib 모드는 표준 C 라이브러리의 일종인 Newlib을 사용하며, Linux 모드에서는 GNU에서 제공되는 표준 C 라이브러리를 사용한다.

Newlib 모드 설치 Newlib 모드의 툴체인을 설치하기 위해서는 아래와 같이 configuration 한다. "–prefix" 옵션은 툴이 설치될 위치를 지정한다. 아래와 같이 configuration을 진행하면, 실행파일들은 `/opt/riscv/bin`에 설치된다.

```
$ ./configure --prefix=/opt/riscv
$ make
```

위와 같이 설치하면 64 비트 컴파일러인 `riscv64-unknown-elf-gcc`가 default로 설치된다.

Linux 모드 설치 Linux 모드의 툴체인을 설치하기 위해서는 아래와 같이 configuration 한다. "–prefix" 옵션은 툴이 설치될 위치를 지정한다. 아래와 같이 configuration을 진행하면, 실행파일들은 `/opt/riscv/bin`에 설치된다.

```
$ ./configure --prefix=/opt/riscv
$ make linux
```

위와 같이 설치하면 RV64GC (64비트) 툴체인이 설치된다. 만약 32비트 RV32GC 툴체인을 설치하기 위해서는 아래와 같이 configuration하고 컴파일, 설치한다.

```
$ ./configure --prefix=/opt/riscv --with-arch=rv32gc --with-abi=ilp32d
$ make linux
```

현재 `--with-arch` 옵션으로 지정할 수 있는 아키텍춰는 rv32i 또는 rv64i과 (A)tomics, (M)ultiplication 및 division, (F)loat, (D)ouble 확장을 지원한다. (g)eneral 옵션은 MAFD를 모두 표시한 것과 같다.

제공되는 ABI는 ilp32 (32-bit soft-float), ilp32d (32-bit hard-float), ilp32f (32-bit with single-precision in registers and double in memory, niche use only), lp64 lp64f lp64d (same but with 64-bit long and pointers)이다.

multilib 모드 32비트와 64비트 컴파일러와 라이브러리를 모두 설치하기 위해서는 아래와 같이 설치한다.

```
$ ./configure --prefix=/opt/riscv --enable-multilib
$ make (또는 make linux)
```

make 명령은 Newlib을 사용하는 32비트, 64비트 컴파일러를 생성한다. make linux 명령은 GLIBC (GNU 표준 C 라이브러리)을 사용하는 32비트, 64비트 컴파일러를 생성한다.

The multilib compiler will have the prefix riscv64-unknown-elf- or riscv64-unknown-linux-gnu-, but will be able to target both 32-bit and 64-bit systems. It will support the most common -march/-mabi options, which can be seen by using the –print-multi-lib flag on either cross-compiler.

제 3 절 GNU RISC-V 툴체인 사용

이 섹션에서는 GNU RISC-V 툴체인을 사용하여 사용자 소스 코드를 RISC-V 실행파일로 컴파일하는 방법을 설명한다.

3.1 RISC-V 컴파일을 이용한 소스 코드 컴파일

2절에서 설명된 대로 GNU RISC-V 컴파일러를 설치하였다면 아래와 같이 C 소스코드를 RISC-V 실행 파일로 컴파일 할 수 있다. 설치된 RISC-V 컴파일러는 "-march, -mabi" 옵션을 이용하여 대상 아키텍춰와 ABI를 지정할 수 있다.

-march 이 옵션에 지정할 수 있는 아키텍춰는 rv32i 또는 rv64i과 (A)tomics, (M)ultiplication 및 division, (F)loat, (D)ouble 확장을 지원한다. (g)eneral 옵션은 MAFD를 모두 표시한 것과 같다.

-mabi ilp32 (32-bit soft-float), ilp32d (32-bit hard-float), ilp32f (32-bit with single-precision in registers and double in memory, niche use only), lp64 lp64f lp64d (same but with 64-bit long and pointers)

다음의 스크립트(compile_gnu_riscv.bash)를 이용하면 컴파일 과정을 쉽게 진행할 수 있다. GCC_32BIT_COMP, GCC_64BIT_COMP 변수는 컴파이러의 실행 파일의 이름이며, ARCH32, ABI32, ARCH64, ABI64 변수는 각각 32비트와 64비트 컴파일러의 -march, -mabi 옵션의 값이다. 이 변수들을 적당한 값으로 지정한 후 이 스크립트를 사용할 수 있다.

```
################################################
# compile_gnu_riscv.bash
################################################
#! /bin/bash

GCC_32BIT_COMP=riscv32-unknown-elf-gcc
GCC_64BIT_COMP=riscv64-unknown-elf-gcc

ARCH32=rv32im
ABI32=ilp32
```

```
ARCH64=rv64imafdc
ABI64=lp64d


if [ "$#" -ne 2 ]; then
  echo "Usage: $0 target <C source codes>"
  echo "     target = 1 for 32-bit code generation"
  echo "     target = 2 for 64-bit code generation"
  exit 2
else
  name=‘echo $2 | sed ’s/\.c//g’‘

  case $1 in
    "1")
      echo "### 32-bit code generation ..."
      echo "### Clearing previous results ..."
      rm ${name}_32.out

      echo "### Running ${GCC_32BIT_COMP} -march=${ARCH32} -mabi=${ABI32} -o ${name}_32.out $2"
      ${GCC_32BIT_COMP} -march=${ARCH32} -mabi=${ABI32} -o ${name}_32.out $2
      ;;

    "2")
      echo "### 64-bit code generation ..."
      echo "### Clearing previous results ..."
      rm ${name}_64.out

      echo "### Running ${GCC_64BIT_COMP} -march=${ABI64} -mabi=${ABI64} -o ${name}_64.out $2"
      ${GCC_64BIT_COMP} -march=${ARCH32} -mabi=${ABI32} -o ${name}_32.out $2
      ;;

    *)
      echo "Wrong target. target must be 1 for 32-bit code generation and 2 for 64-bit code generation"
      exit 2;;
  esac
fi
################################################
```

3.2 RISC-V 파일 실행

여기서는 3.1에서와 같이 컴파일된 RISC-V 실행파일을 시뮬레이터 (`riscv32-unknown-elf-run`, `riscv64-unknown-elf-run`, `spike`)상에서 실행하는 방법을 설명한다.

`riscv32-unknown-elf-run`, `riscv64-unknown-elf-run` 이 툴은 GNU Toolchain에 포함된 시뮬레이터로 인자로 RISC-V 실행파일을 받는다. 만약 실행파일의 이름이 `sim_32.out`(32비트 실행파일)이라면 다음과 같이 실행한다.

```
$ riscv32-unknown-elf-run sim_32.out
```

`spike` spike를 이용하여 RISC-V 실행파일을 실행하기 위해서는 RISC-V Proxy Kernel (PK)를 사용한다. 만약 실행파일의 이름이 `sim_32.out`(32비트 실행파일)이라면 다음과 같이 실행한다.

```
$ spike /opt/riscv/bin/pk sim_32.out
```

다음의 스크립트(`run_riscv_exe.bash`)를 이용하면 시뮬레이션 과정을 쉽게 진행할 수 있다. `SPIKE_32`, `SPIKE_64` 변수는 spike 실행파일의 이름이며, `RUN_32`, `RUN_32`는 GNU RISC-V 툴체인의 시뮬레이터 이름이다. `PK_32`, `PK_64` 변수는 RISC-V Proxy Kernel의 이름이다. 이 변수들을 적당한 값으로 지정한 후 이 스크립트를 사용할 수 있다.

```
################################################
# run_riscv_exe.bash
################################################
#! /bin/bash

SPIKE_32=spike_32
SPIKE_64=spike_64
RUN_32=riscv32-unknown-elf-run
RUN_64=riscv64-unknown-elf-run
PK_32=/home/jong/Projects/RISCV/riscv32-unknown-elf/bin/pk
PK_64=/home/jong/Projects/RISCV/riscv64-unknown-elf/bin/pk

if [ "$#" -ne 2 ]; then
  echo "Error running simulation"
  echo "Usage: $0 sim_mode executable"
  echo "     sim_mode = 1: ${RUN_32} <executable>"
  echo "     sim_mode = 2: ${SPIKE_32} ${PK_32} <executable>"
  echo "     sim_mode = 3: ${RUN_64} <executable>"
  echo "     sim_mode = 4: ${SPIKE_64} ${PK_64} <executable>"
  echo "Example running a.out by using run program: $0 1 a.out"
  exit 2
```

```
fi

if [ ! -f $2 ]; then
  echo "Executable $2 does not exist"
  exit 3
fi

case $1 in
  "1")
    echo "### Running by using riscv32-unknown-elf-run $2"
    echo "-----------------------------------------------"
    echo ""
    time ${RUN_32} $2
    ;;
  "2")
    echo "### Running by using spike_32 $2"
    echo "-----------------------------------------------"
    echo ""
    time ${SPIKE_32} ${PK_32} $2
    ;;
  "3")
    echo "### Running by using riscv64-unknown-elf-run $2"
    echo "-----------------------------------------------"
    echo ""
    time ${RUN_64} $2
    ;;
  "4")
    echo "### Running by using spike_64 $2"
    echo "-----------------------------------------------"
    echo ""
    time ${SPIKE_64} ${PK_64} $2
    ;;

  *)
    echo "Wrong mode. MODE must be one of 1, 2, or 3"
    exit 2;;
 esac
```