深圳大學
SHENZHEN UNIVERSITY

1 引论

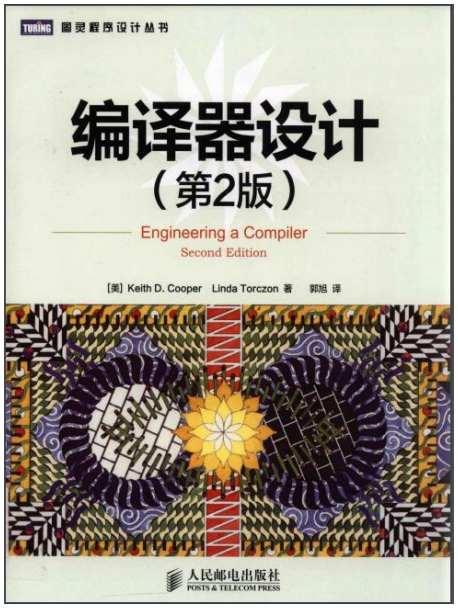
潘志铭

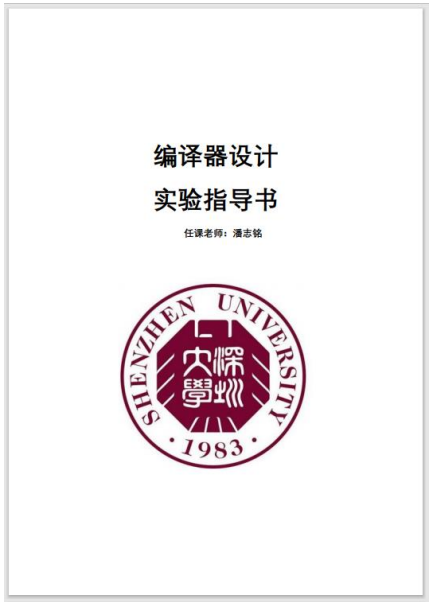
深圳大學
SHENZHEN UNIVERSITY

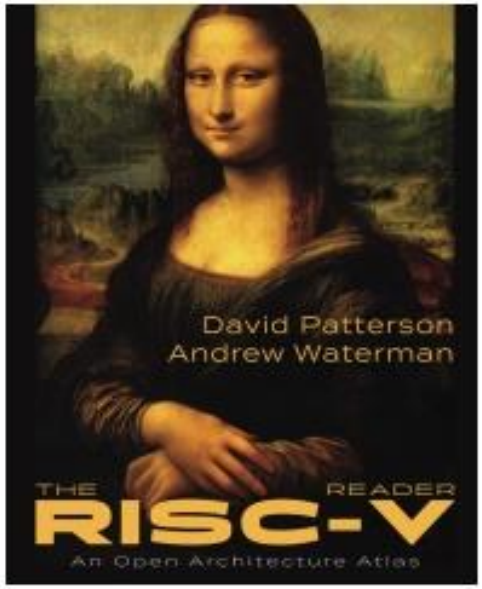
课程介绍

课程概要

- 教材
  - 《编译器设计：第二版》
  - 《编译器设计：实验教程书》
  - *The RISC-V Reader: An Open Architecture Atlas*







深圳大学 SHENZHEN UNIVERSITY

课程概要

- 安排
  - 网站：https://github.com/xicongye/compiler-design
  - 学时：36
  - 学分：2
  - 先修课程：C语言、计算机组成原理
- 课程结构
  - 理论部分：上课听讲，下课作业，交书面作业
  - 实践部分：按照《实验指导书》步骤完成实验报告

深圳大学
SHENZHEN UNIVERSITY

课程内容

- 1 引论
- 2 RISC-V汇编语言介绍
- 3 编译原理的基础知识
- 4 RISC-V工具链的开发

深圳大学 SHENZHEN UNIVERSITY

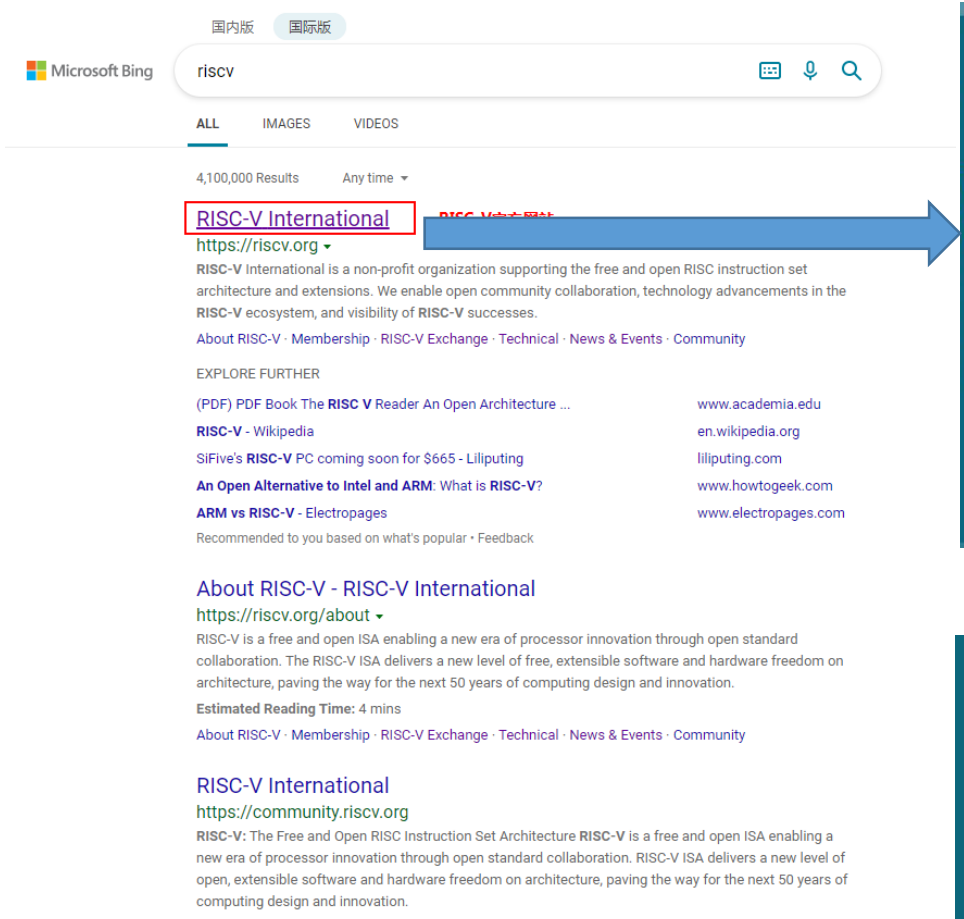
开始本课程之前需要做的准备

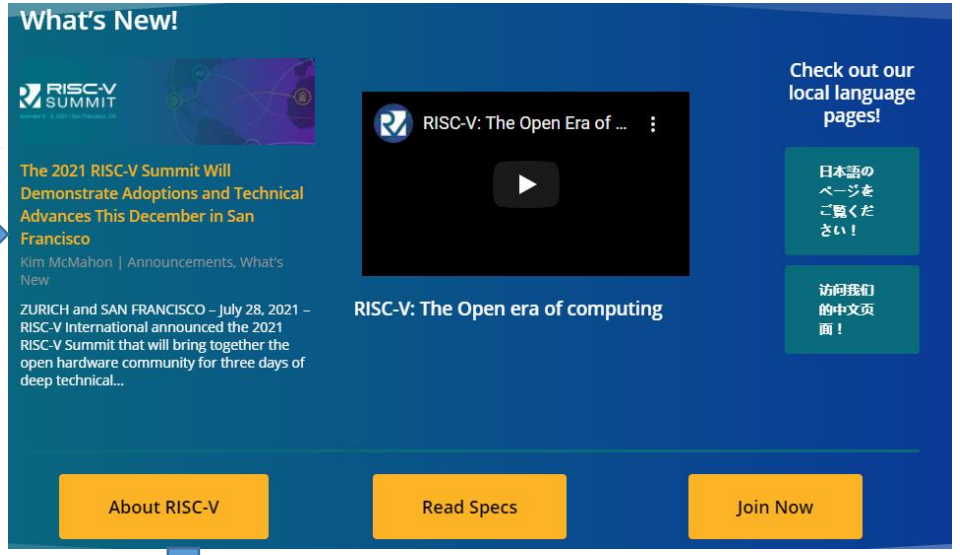
- 安装Ubuntu系统
  - WSL
  - VMware
- Git的基本操作
  - git clone, git diff, git status, git checkout
  - ……
- GitHub
  - 网页版使用界面
  - ……
- Bash Shell的基本命令
  - ls, cp, mv, mkdir, vi
  - ……

深圳大學
SHENZHEN UNIVERSITY

RISC-V介绍

RISC-V的定义





RISC-V（读作“risk five”）是基于精简指令集原理建立的开放指令集架构。

深圳大学 SHENZHEN UNIVERSITY

RISC-V的特点

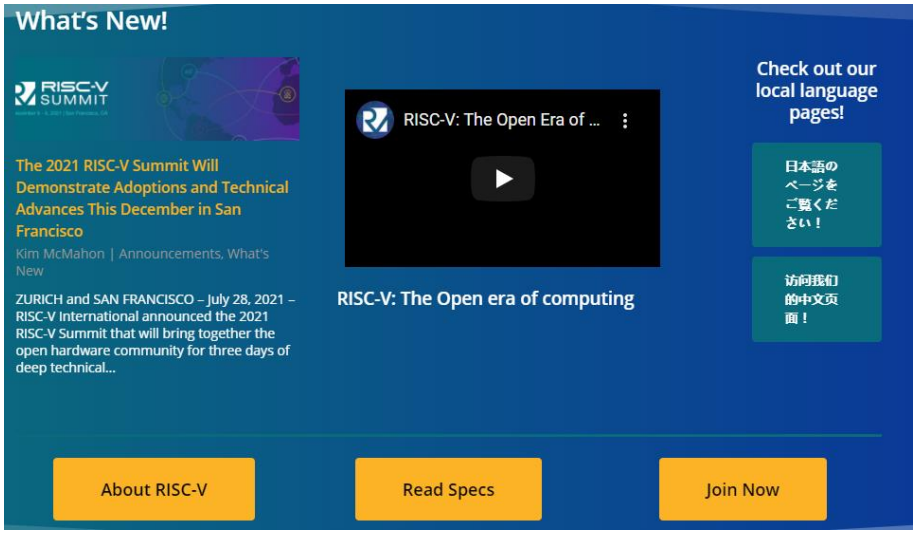
- 完全开源
  - 开源采用宽松的BSD协议，企业完全自由免费使用，同时也容许企业添加自有指令集拓展而不必开放共享以实现差异化发展。
- 架构简单
  - RISC-V基础指令集则只有40多条，加上其他的模块化扩展指令总共几十条指令。RISC-V的规范文档仅有145页。
- 易于移植*nix
  - RISC-V提供了特权级指令和用户级指令，使开发者能非常方便的移植linux和unix系统到RISC-V平台。

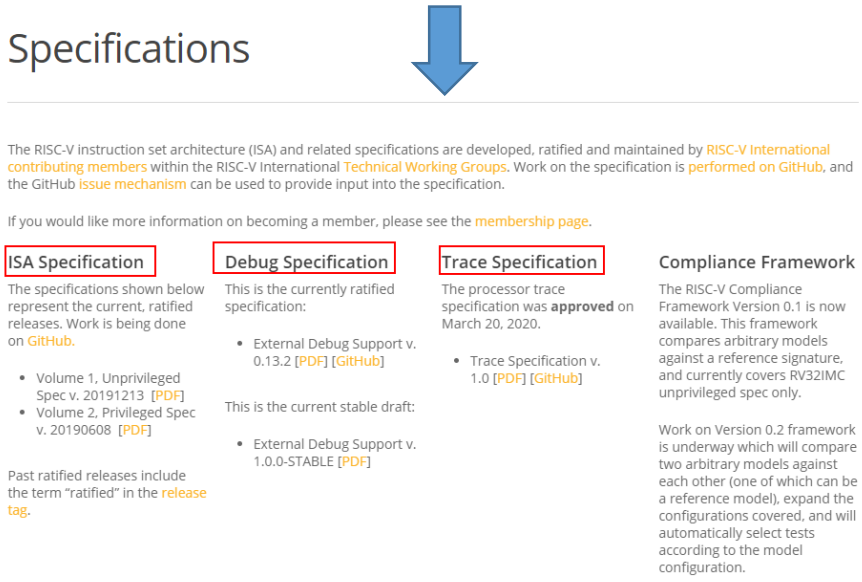
深圳大学 SHENZHEN UNIVERSITY

RISC-V的特点

- 模块化设计
  - 用户能够灵活选择不同的模块组合，来实现自己定制化设备的需要。
- 完整的工具链
  - RISC-V社区提供了完整的工具链，并且RISC-V基金会持续维护该工具链。当前RISC-V的支持已经合并到主要的工具中，比如编译工具链gcc, 仿真工具qemu等。
- 社区贡献
  - 完整的工具链维护，大量的开源项目。risc-v的google讨论组(名称：RISC-V ISA Dev)吸引各地自愿者参与讨论来不断改进risc-v架构。

RISC-V的规范文档





ISA Specification

- Volume 1, Unprivileged Spec
- Volume 2, Privileged Spec

*Unprivileged Spec*描述的是RISC-V用户级（user-level）架构，例如RISC-V指令编码和用法；
*Privileged Spec*描述的是RISC-V特权架构，例如机器模式（machine-level）、管理员模式（supervisor-level）；

Debug Specification

- External Debug Support

Trace Specification

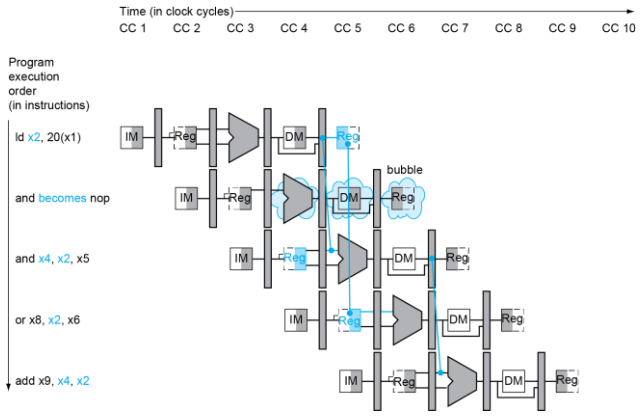
- Trace Specification

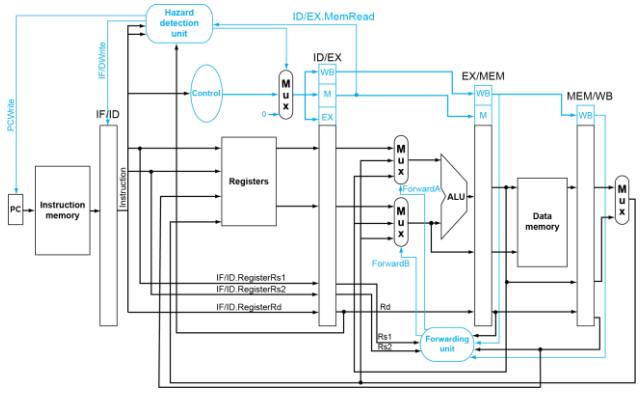
*Debug*和*Trace*描述的是RISC-V软硬件的调试机制规范

深圳大学 SHENZHEN UNIVERSITY

开源的RISC-V处理器

- Rocket Core
  - 5级顺序流水线
  - 使用Chisel语言开发
  - https://github.com/chipsalliance/rocket-chip
- Boom Core
  - 超标量乱序发射、乱序执行
  - 使用Chisel语言开发
  - https://github.com/riscv-boom/riscv-boom
- PULPino Core
  - 4级顺序流水线
  - 使用SystemVerilog语言开发
  - https://github.com/pulp-platform/pulpino
- 蜂鸟E203 Core
  - 2~3级顺序流水线
  - 使用Verilog语言开发
  - https://github.com/riscv-mcu/e203_hbirdv2

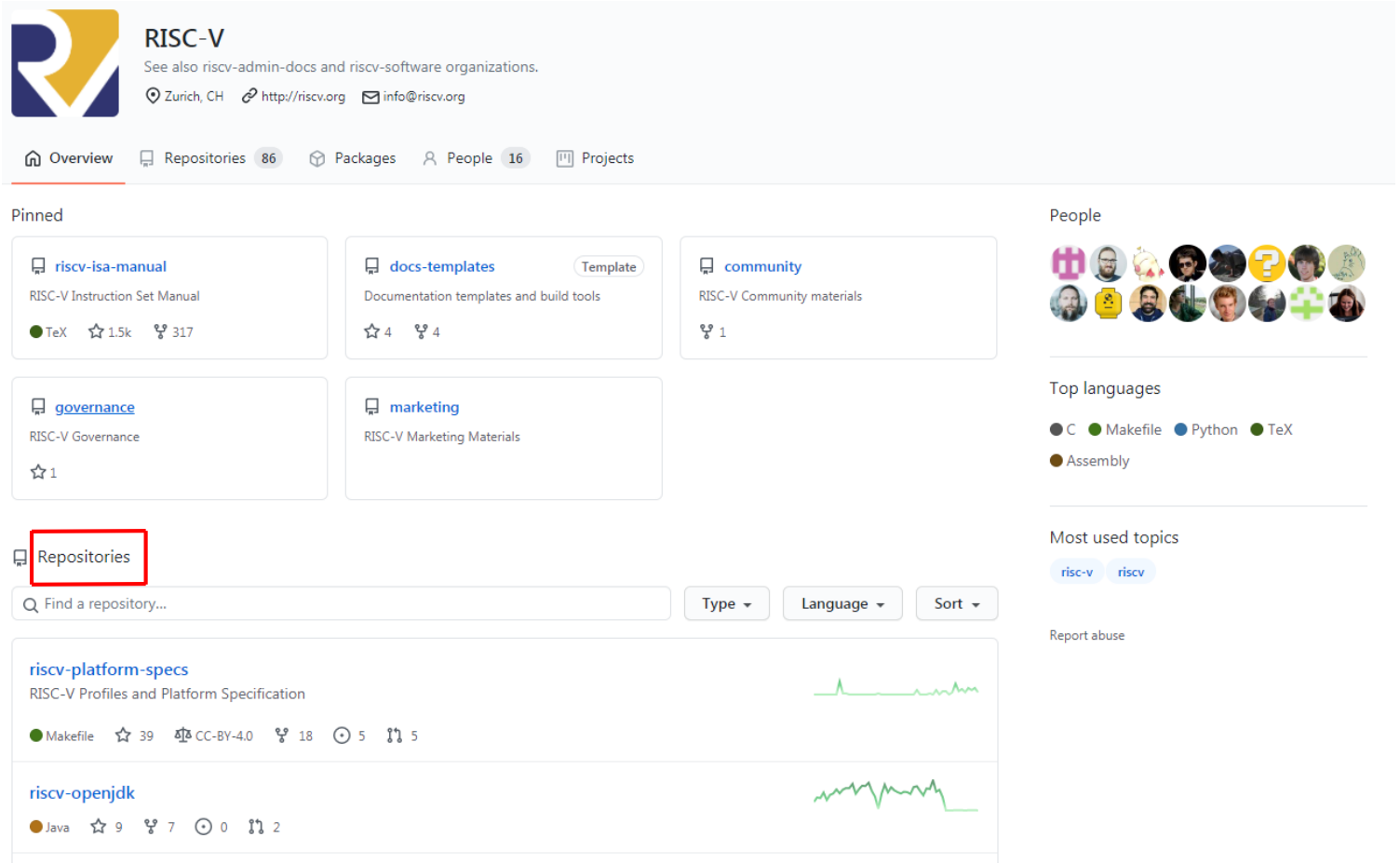




深圳大学 SHENZHEN UNIVERSITY

RISC-V的软件工具

https://github.com/riscv

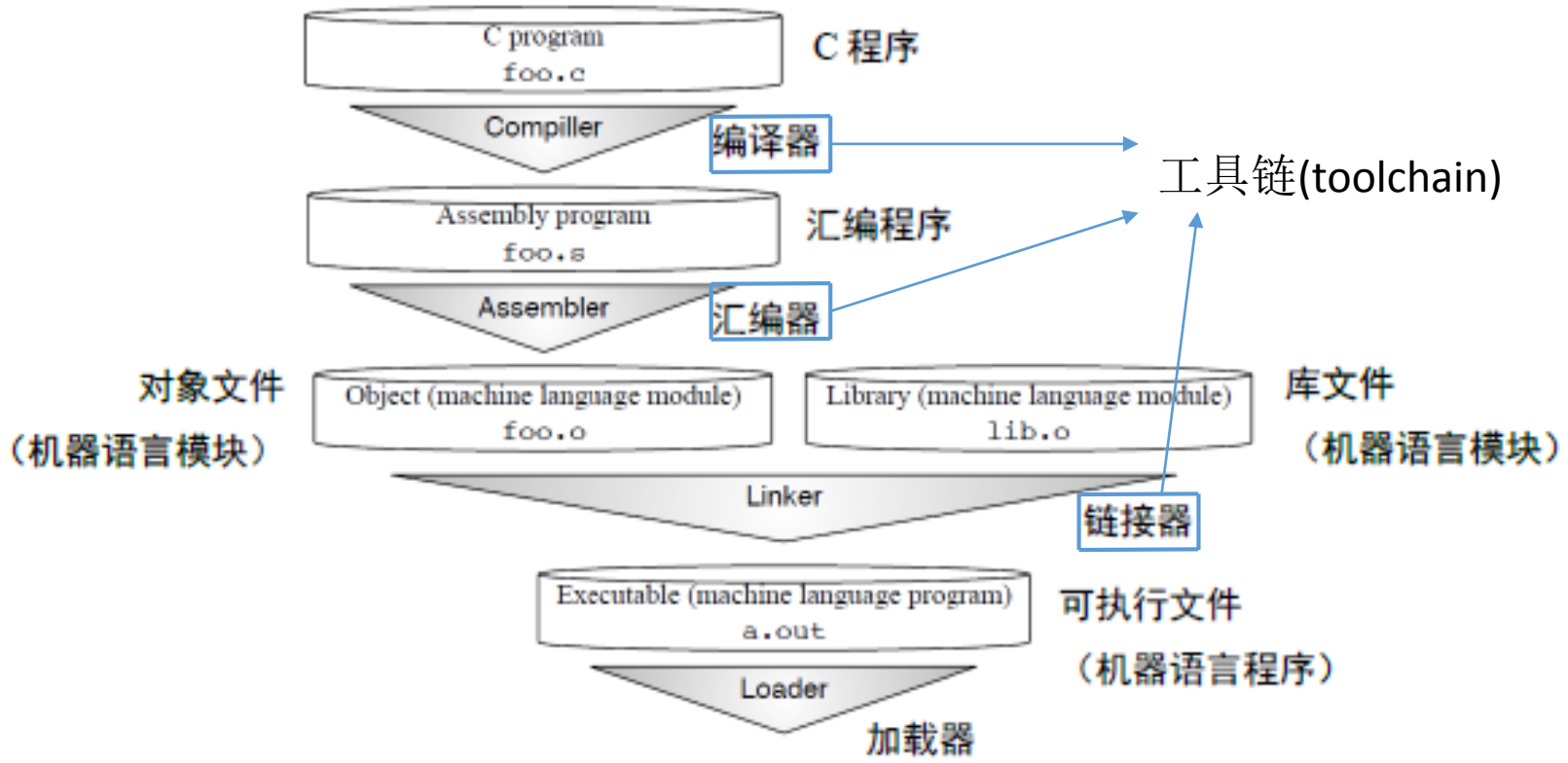


RISC-V相关的软件工具和文档都可以在RISC-V的官方GitHub账号上找到

深圳大學
SHENZHEN UNIVERSITY

RISC-V工具链介绍

工具链的作用

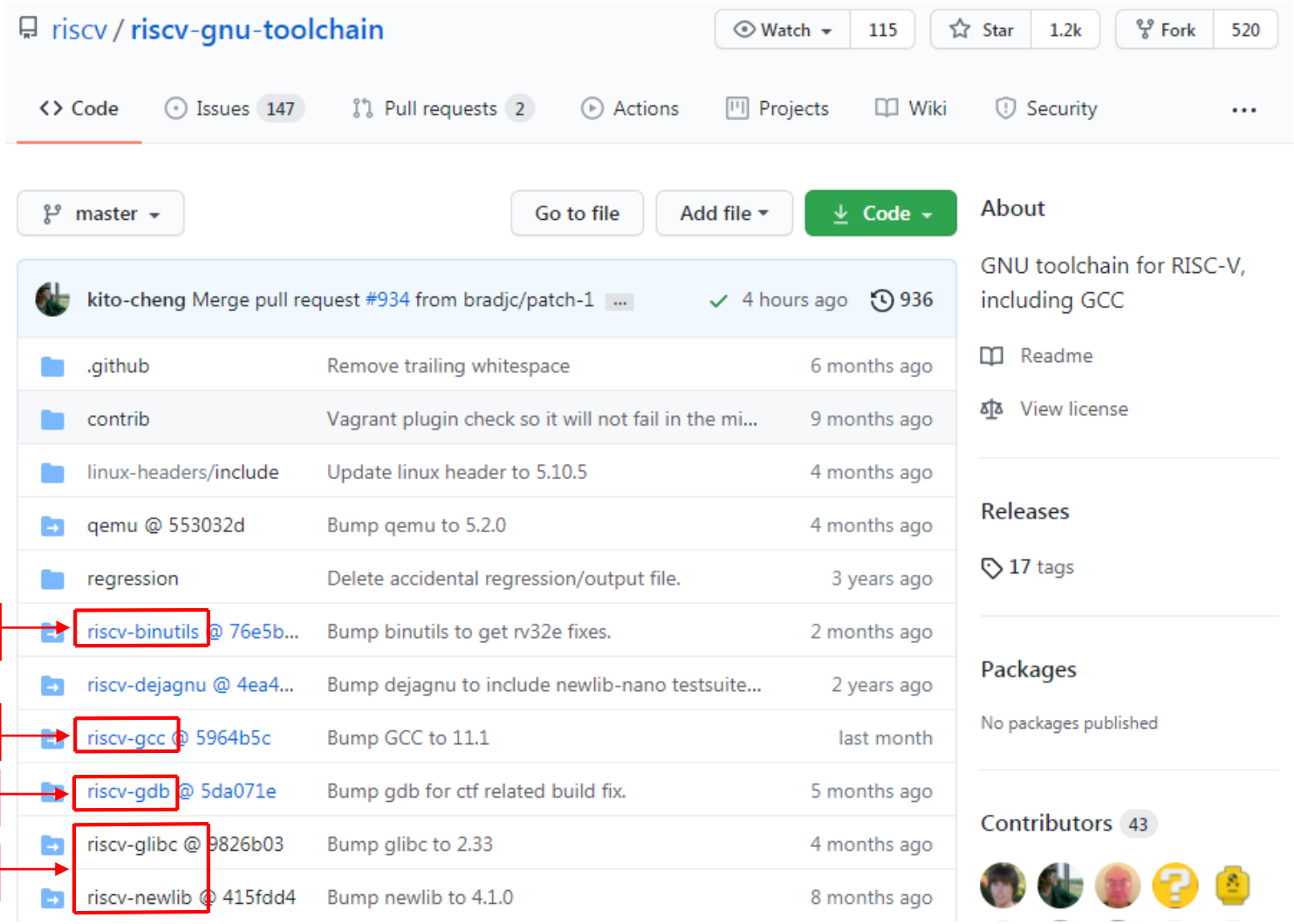


从C源代码翻译为可运行程序的步骤

--- 上图来自《RISC-V手册》第41页

深圳大学 SHENZHEN UNIVERSITY

RISC-V的工具链



汇编器、链接器

编译器

调试器

C运行库

深圳大学 SHENZHEN UNIVERSITY

GNU介绍

riscv / riscv-gnu-toolchain

国内版 国际版

gnu

ALL IMAGES VIDEOS

23,700,000 Results Any time

The GNU Operating System and the Free Software Movement

https://www.gnu.org

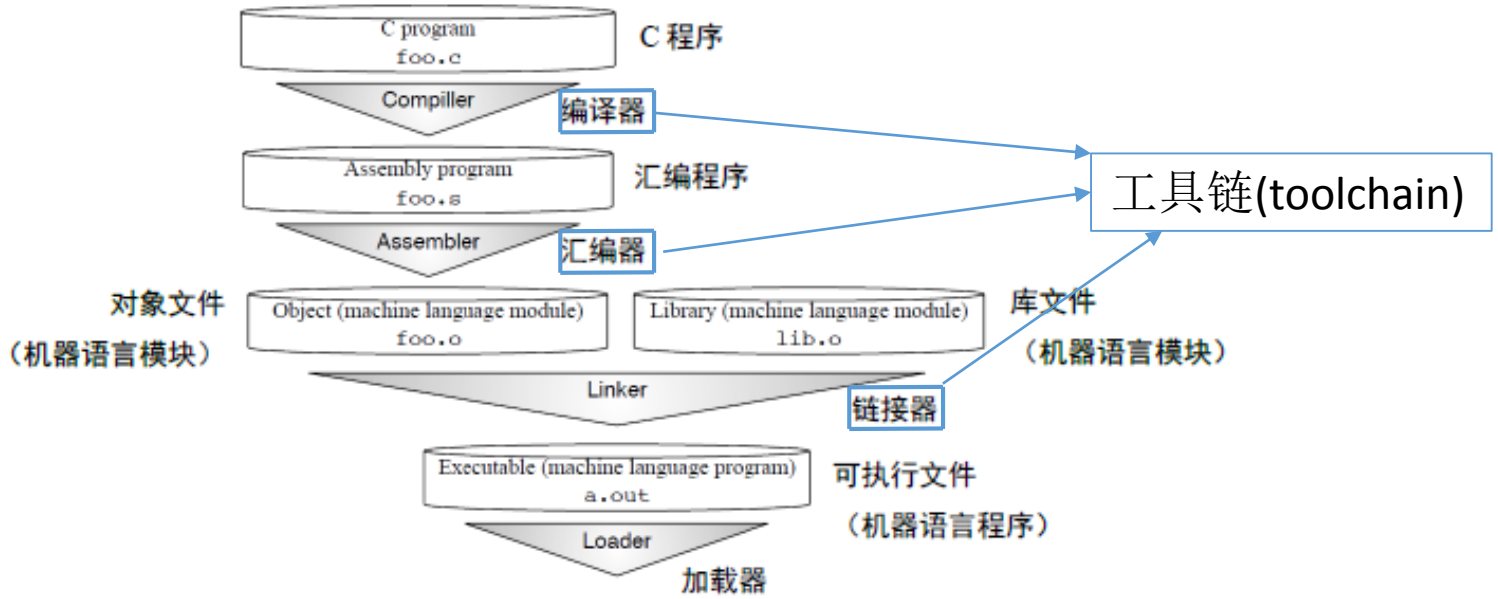
Jul 21, 2021 · GNU is a Unix-like operating system. That means it is a collection of many programs: applications, libraries, developer tools, even games. The development of GNU, started in January 1984, is known as the GNU Project. Many of the programs in GNU are released under the auspices of the GNU Project; those we call GNU packages.

GNU是什么?

GNU是一个自由软件操作系统—就是说，它尊重其使用者的自由。GNU操作系统包括GNU软件包（专门由GNU工程发布的程序）和由第三方发布的自由软件。GNU的开发使你能够使用电脑而无需安装可能会侵害你自由的软件。

我们建议安装这些GNU版本（更确切地说是，GNU/Linux发行版），它们完全是自由软件。更多关于GNU。

GNU有哪些重要的软件工具？



GCC, the GNU Compiler Collection

The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran, Ada, Go, and D, as well as libraries for these languages (libstdc++,...). GCC was originally written as the compiler for the GNU operating system. The GNU system was developed to be 100% free software, free in the sense that it respects the user's freedom.

We strive to provide regular, high quality releases, which we want to work well on a variety of native and cross targets (including GNU/Linux), and encourage everyone to contribute changes or help testing GCC. Our sources are readily and freely available via Git and weekly snapshots.

https://gcc.gnu.org

GNU Binutils

The GNU Binutils are a collection of binary tools. The main ones are:

- **ld** - the GNU linker.
- **as** - the GNU assembler.

https://www.gnu.org/software/binutils

--- 上图来自《RISC-V手册》第41页

深圳大学 SHENZHEN UNIVERSITY

GNU Compiler Collection(GCC)

- 广义的GCC实质上是多个程序的集合：
  - GCC(GNU C Compiler)是编译工具，能够将C/C++语言编写的程序转换成处理器能够执行的二进制代码；
  - GCC既支持本地编译（即在一个平台上编译该平台运行的程序），也支持交叉编译（即在一个平台上编译供另外一个平台运行的程序）；
  - Binutils(Binary Utilities)是一组二进制程序处理工具，包含汇编器as，链接器ld，反汇编器objdump，查看ELF文件信息的readelf，查看ELF文件大小的size等等；
  - C运行库，应用最为广泛的有glibc(GNU C Library)，以及newlib（相比glibc更小，适合单片机系统使用）

深圳大学 SHENZHEN UNIVERSITY

GCC工具链的命名方式

- 命名方式：arch[-vendor][-os][-(gnu)eabi]-gcc
  - arm-none-eabi-gcc
  - arm-none-linux-gnueabi-gcc
  - arm-linux-gnueabi-gcc
  - arm-linux-gnueabihf-gcc
  - riscv64-unknown-linux-gnu-gcc
  - riscv32-unknown-linux-gnu-gcc
  - riscv64-unknown-elf-gcc
  - riscv32-unknown-elf-gcc

ARM architecture, no vendor, not target an operating system, complies with the ARM EABI

用于编译 ARM 架构的裸机系统（包括 ARM Linux 的 boot、kernel，不适用编译 Linux 应用 Application），一般适合 ARM7、Cortex-M 和 Cortex-R 内核的芯片使用，所以不支持那些跟操作系统关系密切的函数（比如fork函数），该工具链使用的是 newlib 库。

深圳大学 SHENZHEN UNIVERSITY

GCC工具链的命名方式

- 命名方式：arch[-vendor][-os][-(gnu)eabi]-gcc
  - arm-none-eabi-gcc
  - arm-none-linux-gnueabi-gcc
  - arm-linux-gnueabi-gcc
  - arm-linux-gnueabihf-gcc
  - riscv64-unknown-linux-gnu-gcc
  - riscv32-unknown-linux-gnu-gcc
  - riscv64-unknown-elf-gcc
  - riscv32-unknown-elf-gcc

ARM architecture, no vendor, creates binaries that run on the Linux operating system, and uses the GNU EABI

主要用于基于ARM架构的Linux系统，可用于编译 ARM 架构的 u-boot、Linux 内核、linux应用等，一般ARM9、ARM11、Cortex-A 内核，带有 Linux 操作系统的会用到，该工具链使用的是glibc库。

深圳大学 SHENZHEN UNIVERSITY

GCC工具链的命名方式

- 命名方式：arch[-vendor][-os][-(gnu)eabi]-gcc
  - arm-none-eabi-gcc
  - arm-none-linux-gnueabi-gcc
  - arm-linux-gnueabi-gcc
  - arm-linux-gnueabihf-gcc
  - riscv64-unknown-linux-gnu-gcc
  - riscv32-unknown-linux-gnu-gcc
  - riscv64-unknown-elf-gcc
  - riscv32-unknown-elf-gcc

两者区别在于选项-mfloat-abi的默认值不同，前者默认的选项是-mfloat-abi=softfp，也就是使用fpu执行浮点运算，但是传参数时使用的是通用整型寄存器，这样在处理中断时，只需要保存通用整型寄存器，优点是中断负荷小，绝点是参数需要转换成浮点形式再计算。后者的默认选项是-mfloat-abi=hard，即不仅使用fpu执行浮点运算，也使用浮点寄存器传递参数，这样参数就省去了转换的步骤，优点是性能好，缺点是中断负荷高。

深圳大学 SHENZHEN UNIVERSITY

GCC工具链的命名方式

- 命名方式：arch[-vendor][-os][-(gnu)eabi]-gcc
  - arm-none-eabi-gcc
  - arm-none-linux-gnueabi-gcc
  - arm-linux-gnueabi-gcc
  - arm-linux-gnueabihf-gcc
  - riscv64-unknown-linux-gnu-gcc
  - riscv32-unknown-linux-gnu-gcc
  - riscv64-unknown-elf-gcc
  - riscv32-unknown-elf-gcc

riscv64-unknown-linux-gnu-前缀表示该工具链是64位RISC-V架构的Linux版本工具链。这里的linux指的是该工具链使用glibc库。

同理，riscv32-unknown-linux-gnu-前缀是指32位RISC-V架构的Linux版本工具链。

前缀riscv64（还有riscv32的版本）与运行在64位或者32位电脑上毫无关系，此处的64和32是指如果没有通过-march和-mabi选项指定RISC-V架构的位宽，默认将会按照64位还是32位的RISC-V架构来编译程序。有关-march和-mabi选项的含义会在后面描述。

深圳大学 SHENZHEN UNIVERSITY

GCC工具链的命名方式

- 命名方式：arch[-vendor][-os][-(gnu)eabi]-gcc
  - arm-none-eabi-gcc
  - arm-none-linux-gnueabi-gcc
  - arm-linux-gnueabi-gcc
  - arm-linux-gnueabihf-gcc
  - riscv64-unknown-linux-gnu-gcc
  - riscv32-unknown-linux-gnu-gcc
  - riscv64-unknown-elf-gcc
  - riscv32-unknown-elf-gcc

以riscv64-unknown-elf-/riscv32-unknown-elf为前缀表示该工具链为非Linux(Non-linux)版本的工具链。

Non-Linux不是指当前版本工具链一定不能运行在Linux操作系统的电脑上，Non-Linux是指该GCC工具链会使用newlib作为C运行库。

RISC-V工具链的简单使用

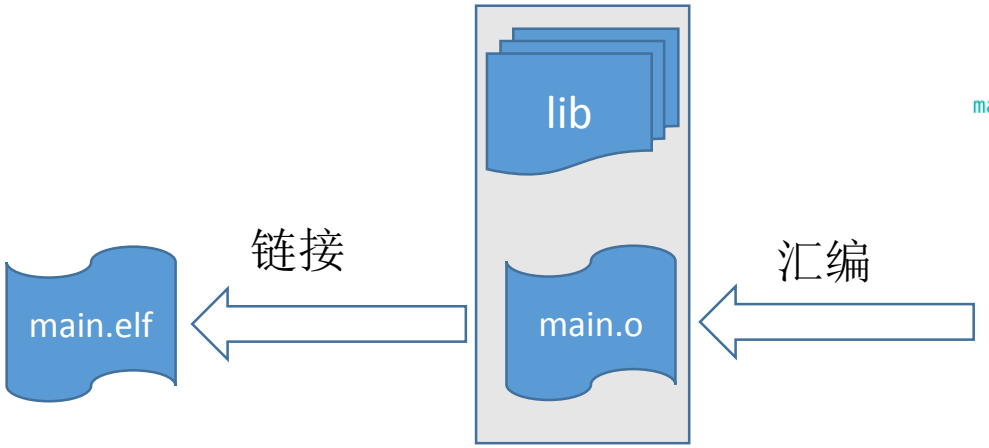
```c
#include <stdio.h>
int main()
{
  printf("Hello World!\n");
  return 0;
}
```

预处理

```c
1262 static __inline int
1263 _putchar_unlocked(int _c)
1264 {
1265  struct _reent *_ptr;
1266
1267  _ptr = _impure_ptr;
1268  return (__sputc_r(_ptr, _c, ((_ptr)->_stdout)));
1269 }
1270 # 797 "/home/common/riscv-tools/install/tools-for-
1271
1272 # 2 "main.c" 2
1273
1274 # 2 "main.c"
1275 int main()
1276 {
1277   printf("Hello World!\n");
1278   return 0;
1279 }
```

编译

```
main:
        addi    sp,sp,-16
        sd      ra,8(sp)
        sd      s0,0(sp)
        addi    s0,sp,16
        lui     a5,%hi(.LC0)
        addi    a0,a5,%lo(.LC0)
        call    puts
        li      a5,0
        mv      a0,a5
        ld      ra,8(sp)
        ld      s0,0(sp)
        addi    sp,sp,16
        jr      ra
        .size   main, .-main
        .ident  "GCC: (GNU) 10.1.0"
```



上面的操作可以通过一个命令实现：
$ riscv64-unknown-elf-gcc main.c -o main.elf

如果出现`riscv64-unknown-elf-gcc: command not found`的错误，需要将工具链路径加入PATH中：
$ export PATH=/opt/riscv:$PATH

深圳大学 SHENZHEN UNIVERSITY

RISC-V Binutils

• riscv64-unknown-elf-gdb	调试器
• riscv64-unknown-elf-as	汇编器
• riscv64-unknown-elf-ld	链接器
• riscv64-unknown-elf-ar	用于打包静态库
• riscv64-unknown-elf-objdump	反汇编器
• riscv64-unknown-elf-readelf	查看有关ELF文件的信息
• riscv64-unknown-elf-size	查看有关ELF文件大小信息
• riscv64-unknown-elf-objcopy	将ELF文件转成另外一种格式

深圳大学
SHENZHEN UNIVERSITY

C运行库

为了解释C运行库，需要先回忆一下C语言标准。C语言标准主要由两部分组成：一部分描述C的语法，另一部分描述C标准库。C标准库定义了一组标准头文件，每个头文件中包含一些相关的函数、变量、类型声明和宏定义，譬如常见的printf函数便是一个C标准库函数，其原型定义在stdio头文件中。

C语言标准仅仅定义了C标准库函数原型，并没有提供实现。因此，C语言编译器通常需要一个C运行时库（C Run Time Libray，CRT）的支持。C运行时库又常简称为C运行库。与C语言类似，C++也定义了自己的标准，同时提供相关支持库，称为C++运行时库。

深圳大学 SHENZHEN UNIVERSITY

C运行库

要在一个平台上支持C语言，不仅要实现C编译器，还要实现C标准库，这样的实现才能完全支持C标准。glibc（GNU C Library）是Linux下面C标准库的实现，其要点如下：

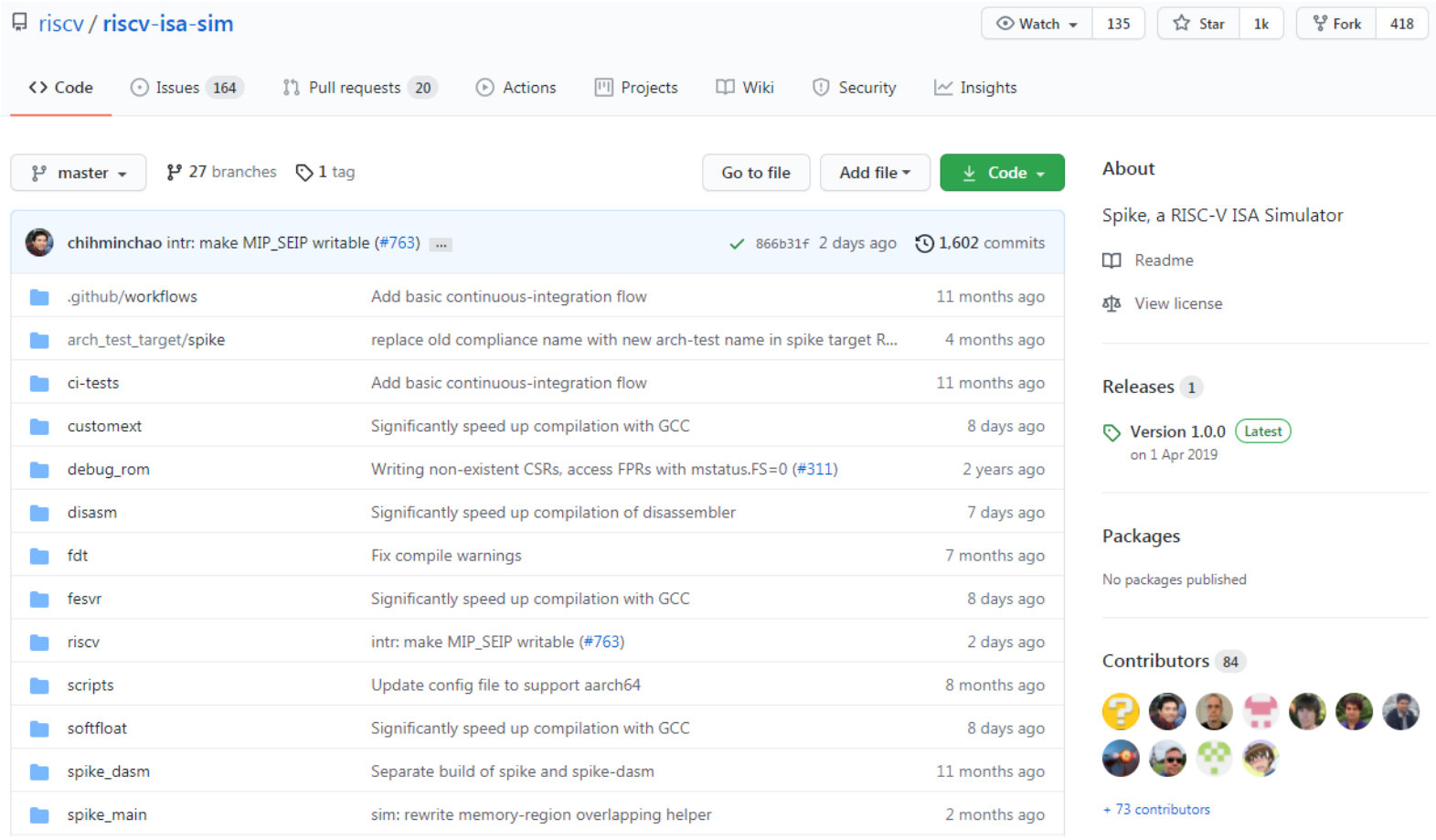
- glibc本身是GNU旗下的C标准库，后来逐渐成为了Linux的标准C库。glibc的主体分布在Linux系统的/lib与/usr/lib目录中，包括 libc 标准 C 函式库、libm数学函式库等等，都以.so做结尾；
- Linux系统通常将libc库作为操作系统的一部分，它被视为操作系统与用户程序的接口。譬如：glibc不仅实现标准C语言中的函数，还封装了操作系统提供的系统服务，即系统调用的封装；
- 对于C++语言，常用的C++标准库为libstdc++；

newlib是一个面向嵌入式系统的C运行库。相对于glibc，newlib实现了大部分的功能函数，但体积却小很多。newlib独特的体系结构将功能实现与具体的操作系统分层，使之能够很好地进行配置以满足嵌入式系统的要求。由于专为嵌入式系统设计，newlib具有可移植性强、轻量级、速度快、功能完备等特点，已广泛应用于各种嵌入式系统中。

深圳大学 SHENZHEN UNIVERSITY

spike介绍

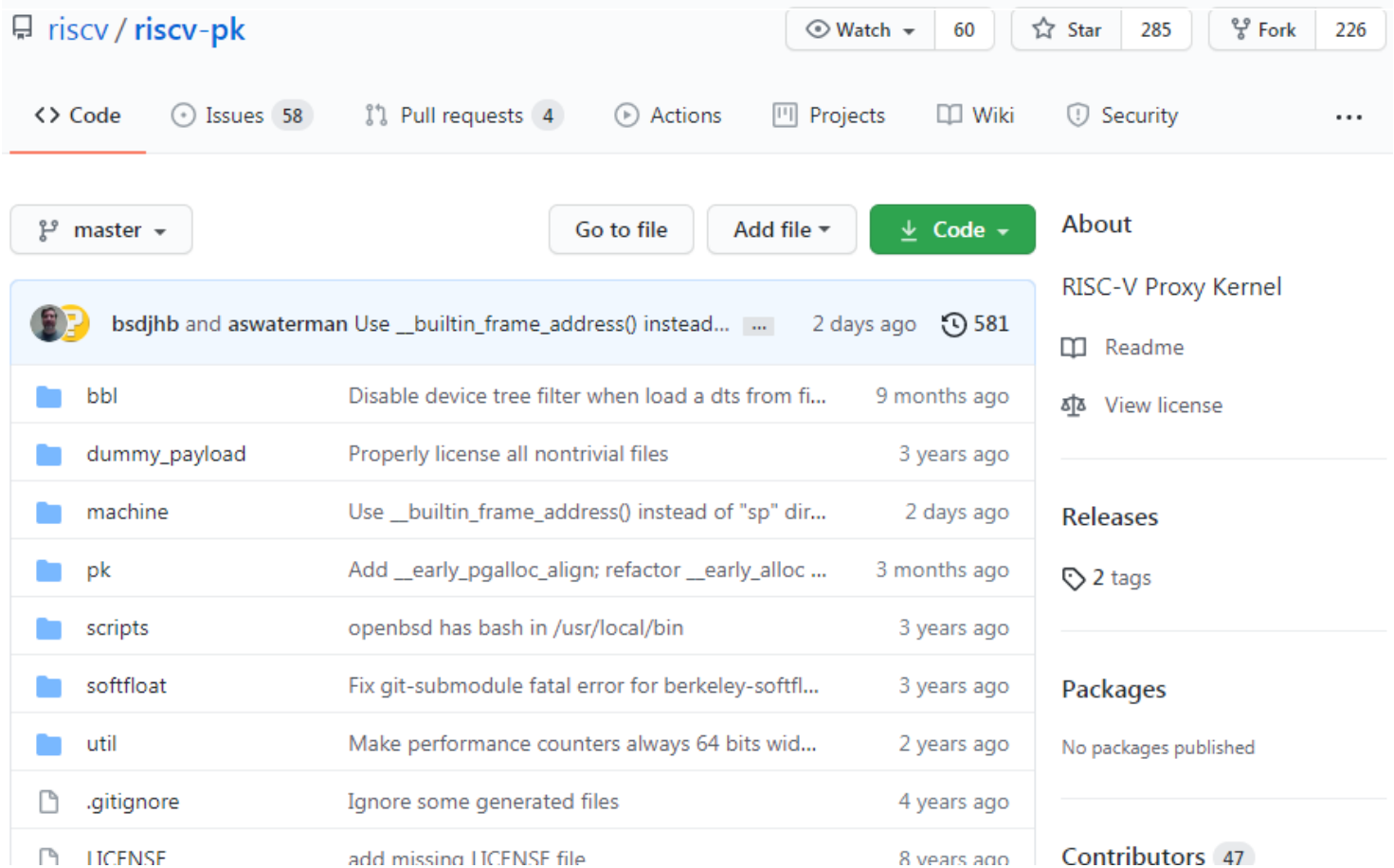
spike是一个RISC-V指令模拟器，它能够模拟一个或者多个RISC-V硬件线程的功能。



深圳大学 SHENZHEN UNIVERSITY

pk介绍

RISC-V Proxy Kernel（RISC-V代理内核）用于给RISC-V执行文件提供执行环境。



深圳大学 SHENZHEN UNIVERSITY

结合spike和pk运行程序

- 编译helloword程序

```
$ riscv64-unknown-elf-gcc main.c -o main.elf
```

- 将spike路径加入PATH环境变量中

```
$ export PATH=/opt/riscv:$PATH
```

- 将pk路径加入PATH环境变量中

```
$ export PATH=/opt/riscv/riscv64-unknown-elf/bin:$PATH
```

- 运行spike和pk

```
$ spike pk main.elf
```

深圳大學
SHENZHEN UNIVERSITY

Q & A

深圳大學
SHENZHEN UNIVERSITY

Backup

深圳大学 SHENZHEN UNIVERSITY

RISC-V工具链的构建步骤

- 获取源代码

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

- 安装依赖文件

  $ sudo apt-get install autoconf automake autotools-dev curl libmpc-dev

  $ sudo apt-get install libmpfr-dev libgmp-dev gawk build-essential bison flex

  $ sudo apt-get install texinfo gperf libtool patchutils bc zlib1g-dev libexpat-dev

- 开始构建

  $ cd riscv-gnu-toolchain

  $ mkdir build && cd build

  $ ../configure --prefix=/opt/riscv --enable-multilib

  $ make

  $ make install

深圳大学
SHENZHEN UNIVERSITY

spike的构建步骤

- 获取源码

  $ git clone https://github.com/riscv/riscv.isa-sim.git

- 安装依赖文件

  $ sudo apt-get install device-tree-compile

- 开始构建

  $ cd riscv-isa-sim

  $ mkdir build

  $ cd build

  $ ../configure –prefix=/opt/riscv -enable-histogram

  $ make

  $ make install

深圳大学 SHENZHEN UNIVERSITY

pk的构建步骤

- 获取源码

  $ git clone https://github.com/riscv/riscv-pk.git

- 开始构建

  $ cd riscv-pk

  $ mkdir build

  $ cd build

  $ ../configure -prefix=/opt/riscv -host=riscv64-unknown-elf

  $ make

  $ make install

构建pk之前需要先完成riscv-gnu-toolchain的安装

如果出现`riscv64-unknown-elf-gcc: command not found`的错误，需要将工具链路径加入PATH中：
$ export PATH=/opt/riscv:$PATH

深圳大學
SHENZHEN UNIVERSITY

2 RISC-V汇编语言

潘志铭

深圳大學
SHENZHEN UNIVERSITY

RISC-V寄存器

RISC-V通用寄存器

寄存器	接口名称	描述	在调用中是否保留?
Register	ABI Name	Description	Preserved across call?
x0	zero	Hard-wired zero 硬编码 0	—
x1	ra	Return address 返回地址	No
x2	sp	Stack pointer 栈指针	Yes
x3	gp	Global pointer 全局指针	—
x4	tp	Thread pointer 线程指针	—
x5	t0	Temporary/alternate link register 临时寄存器	No /备用链接寄存器
x6–7	t1–2	Temporaries 临时寄存器	No
x8	s0/fp	Saved register/frame pointer 保存寄存器	Yes /帧指针
x9	s1	Saved register 保存寄存器	Yes
x10–11	a0–1	Function arguments/return values 函数参数	No /返回值
x12–17	a2–7	Function arguments 函数参数	No
x18–27	s2–11	Saved registers 保存寄存器	Yes
x28–31	t3–6	Temporaries 临时寄存器	No
f0–7	ft0–7	FP temporaries 浮点临时寄存器	No
f8–9	fs0–1	FP saved registers 浮点保存寄存器	Yes
f10–11	fa0–1	FP arguments/return values 浮点参数/返回值	No
f12–17	fa2–7	FP arguments 浮点参数	No
f18–27	fs2–11	FP saved registers 浮点保存寄存器	Yes
f28–31	ft8–11	FP temporaries 浮点临时寄存器	No

RISC-V定义了32个XLEN-bit位宽的整型寄存器，和32个XLEN-bit位宽的浮点寄存器[1]

RISC-V各个寄存器的ABI别名

RISC-V各个寄存器的作用描述

在函数调用前后，寄存器的值是否需要保持不变

在RISC-V规范中，PC寄存器独立于通用寄存器

在RISC-V规范中，x0寄存器硬编码为0，读为0，写无效

--- 上图来自《RISC-V手册》第42页

--- [1] XLEN ∈ {32, 64}

深圳大学 SHENZHEN UNIVERSITY

RISC-V CSR寄存器

Number	Privilege	Name	Description
		User Trap Setup	
0x000	URW	ustatus	User status register.
0x004	URW	uie	User interrupt-enable register.
0x005	URW	utvec	User trap handler base address.
		User Trap Handling	
0x040	URW	uscratch	Scratch register for user trap handlers.
0x041	URW	uepc	User exception program counter.
0x042	URW	ucause	User trap cause.
0x043	URW	utval	User bad address or instruction.
0x044	URW	uip	User interrupt pending.
		User Floating-Point CSRs	
0x001	URW	fflags	Floating-Point Accrued Exceptions.
0x002	URW	frm	Floating-Point Dynamic Rounding Mode.
0x003	URW	fcsr	Floating-Point Control and Status Register (frm + fflags).
		User Counter/Timers	
0xC00	URO	cycle	Cycle counter for RDCYCLE instruction.
0xC01	URO	time	Timer for RDTIME instruction.
0xC02	URO	instret	Instructions-retired counter for RDINSTRET instruction.
0xC03	URO	hpmcounter3	Performance-monitoring counter.
0xC04	URO	hpmcounter4	Performance-monitoring counter.
		⋮	
0xC1F	URO	hpmcounter31	Performance-monitoring counter.
0xC80	URO	cycleh	Upper 32 bits of cycle, RV32I only.
0xC81	URO	timeh	Upper 32 bits of time, RV32I only.
0xC82	URO	instreth	Upper 32 bits of instret, RV32I only.
0xC83	URO	hpmcounter3h	Upper 32 bits of hpmcounter3, RV32I only.
0xC84	URO	hpmcounter4h	Upper 32 bits of hpmcounter4, RV32I only.
		⋮	
0xC9F	URO	hpmcounter31h	Upper 32 bits of hpmcounter31, RV32I only.

--- 上图来自 *riscv-privilege*

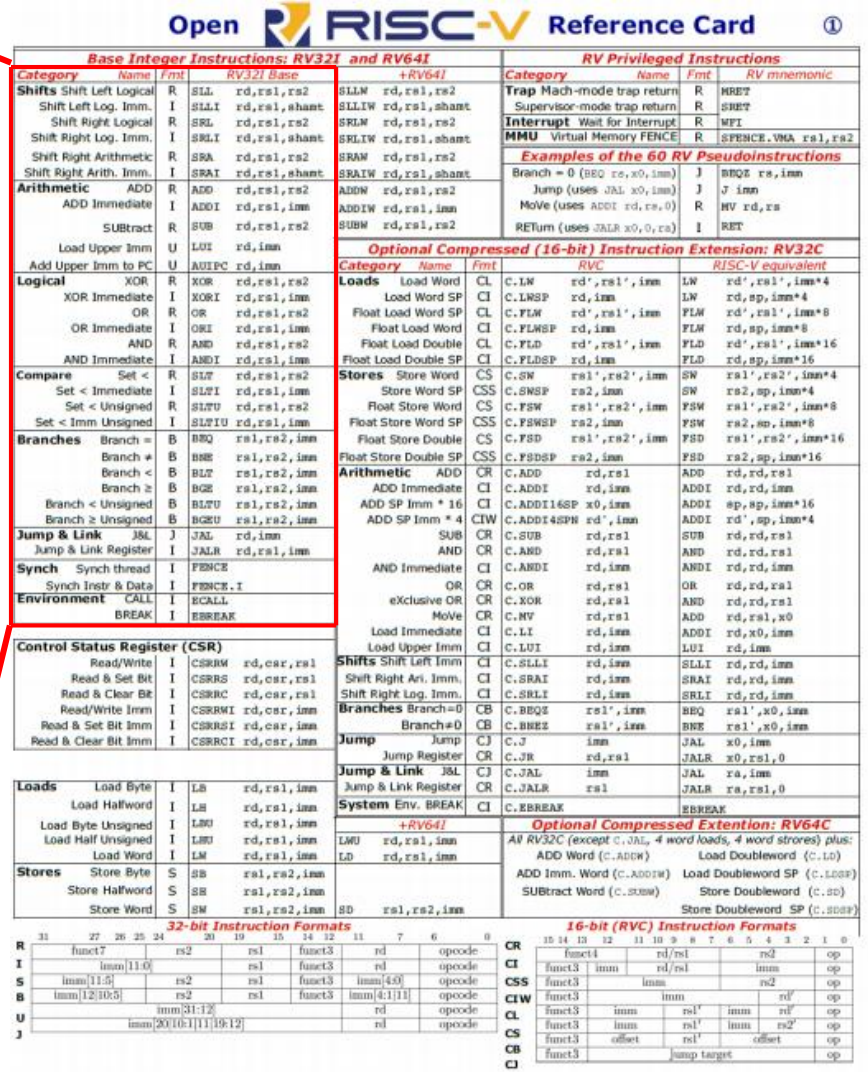
深圳大學
SHENZHEN UNIVERSITY

RISC-V汇编指令

--- 以RV32I为例说明

RISC-V基础整型指令

Category	Name	Fmt	RV32I Base	
Shifts	Shift Left Logical	R	SLL	rd,rs1,rs2
	Shift Left Log. Imm.	I	SLLI	rd,rs1,shamt
	Shift Right Logical	R	SRL	rd,rs1,rs2
	Shift Right Log. Imm.	I	SRLI	rd,rs1,shamt
	Shift Right Arithmetic	R	SRA	rd,rs1,rs2
	Shift Right Arith. Imm.	I	SRAI	rd,rs1,shamt
Arithmetic	ADD	R	ADD	rd,rs1,rs2
	ADD Immediate	I	ADDI	rd,rs1,imm
	SUBtract	R	SUB	rd,rs1,rs2
	Load Upper Imm	U	LUI	rd,imm
	Add Upper Imm to PC	U	AUIPC	rd,imm
Logical	XOR	R	XOR	rd,rs1,rs2
	XOR Immediate	I	XORI	rd,rs1,imm
	OR	R	OR	rd,rs1,rs2
	OR Immediate	I	ORI	rd,rs1,imm
	AND	R	AND	rd,rs1,rs2
	AND Immediate	I	ANDI	rd,rs1,imm
Compare	Set <	R	SLT	rd,rs1,rs2
	Set < Immediate	I	SLTI	rd,rs1,imm
	Set < Unsigned	R	SLTU	rd,rs1,rs2
	Set < Imm Unsigned	I	SLTIU	rd,rs1,imm
Branches	Branch =	B	BEQ	rs1,rs2,imm
	Branch ≠	B	BNE	rs1,rs2,imm
	Branch <	B	BLT	rs1,rs2,imm
	Branch ≥	B	BGE	rs1,rs2,imm
	Branch < Unsigned	B	BLTU	rs1,rs2,imm
	Branch ≥ Unsigned	B	BGEU	rs1,rs2,imm
Jump & Link	J&L	J	JAL	rd,imm
	Jump & Link Register	I	JALR	rd,rs1,imm
Synch	Synch thread	I	FENCE	
	Synch Instr & Data	I	FENCE.I	
Environment	CALL	I	ECALL	
	BREAK	I	EBREAK	



--- 上图来自《RISC-V手册》第1页

深圳大学 SHENZHEN UNIVERSITY

RISC-V基础整型指令

Category	Name	Fmt	RV32I Base	
Shifts	Shift Left Logical	R	SLL	rd,rs1,rs2
	Shift Left Log. Imm.	I	SLLI	rd,rs1,shamt
	Shift Right Logical	R	SRL	rd,rs1,rs2
	Shift Right Log. Imm.	I	SRLI	rd,rs1,shamt
	Shift Right Arithmetic	R	SRA	rd,rs1,rs2
	Shift Right Arith. Imm.	I	SRAI	rd,rs1,shamt
Arithmetic	ADD	R	ADD	rd,rs1,rs2
	ADD Immediate	I	ADDI	rd,rs1,imm
	SUBtract	R	SUB	rd,rs1,rs2
	Load Upper Imm	U	LUI	rd,imm
	Add Upper Imm to PC	U	AUIPC	rd,imm
Logical	XOR	R	XOR	rd,rs1,rs2
	XOR Immediate	I	XORI	rd,rs1,imm
	OR	R	OR	rd,rs1,rs2
	OR Immediate	I	ORI	rd,rs1,imm
	AND	R	AND	rd,rs1,rs2
	AND Immediate	I	ANDI	rd,rs1,imm
Compare	Set <	R	SLT	rd,rs1,rs2
	Set < Immediate	I	SLTI	rd,rs1,imm
	Set < Unsigned	R	SLTU	rd,rs1,rs2
	Set < Imm Unsigned	I	SLTIU	rd,rs1,imm
Branches	Branch =	B	BEQ	rs1,rs2,imm
	Branch ≠	B	BNE	rs1,rs2,imm
	Branch <	B	BLT	rs1,rs2,imm
	Branch ≥	B	BGE	rs1,rs2,imm
	Branch < Unsigned	B	BLTU	rs1,rs2,imm
	Branch ≥ Unsigned	B	BGEU	rs1,rs2,imm
Jump & Link	J&L	J	JAL	rd,imm
	Jump & Link Register	I	JALR	rd,rs1,imm
Synch	Synch thread	I	FENCE	
	Synch Instr & Data	I	FENCE.I	
Environment	CALL	I	ECALL	
	BREAK	I	EBREAK	

- RISC-V指令的一般格式：

  op dst, src1, src2

  - op = 操作符名称（operator）
  - dst = 结果寄存器（destination）
  - src1 = 操作数1（source 1）
  - src2 = 操作数2（source 2）

- RV32基础整型指令分类：
  - Shift：移位指令
  - Arithmetic：算术指令
  - Logical：逻辑操作
  - Compare：比较操作
  - Branches：分支指令
  - Jump & Link：跳转指令
  - Synch：同步指令（不涉及）
  - Environment：环境调用（不涉及）

- RV64I指令在RV32I指令基础上增加了部分移位和算术指令

深圳大学 SHENZHEN UNIVERSITY

RISC-V移位指令

Instructions Name	RISC-V
Shift Left Logical	sll rd, rs1, rs2
Shift Left Logical immediate	slli rd, rs1, shamt
Shift Right Logical	srl rd, rs1, rs2
Shift Right Logical immediate	srli rd, rs1, shamt
Shift Right Arithmetic	sra rd, rs1, rs2
Shift Right Arithmetic immediate	srai rd, rs1, shamt

- Logical shift: 逻辑移位，高位填0
- Arithmetic shift: 算术移位，高位填符号位

深圳大学 SHENZHEN UNIVERSITY

RISC-V算术指令

Instructions Name	RISC-V
Add	add rd, rs1, rs2
Add Immediate	addi rd, rs1, imm
Subtract	sub rd, rs1, rs2
Load Upper Immediate	lui rd, imm
Add Upper Immediate to PC	auipc rd, imm

lui rd, immediate　　x[rd] = sext(immediate[31:12] << 12)

高位立即数加载 *(Load Upper Immediate)*. U-type, RV32I and RV64I.

将符号位扩展的 20 位立即数 *immediate* 左移 12 位，并将低 12 位置零，写入 x[*rd*]中。

auipc rd, immediate　　x[rd] = pc + sext(immediate[31:12] << 12)

PC 加立即数 *(Add Upper Immediate to PC)*. U-type, RV32I and RV64I.

把符号位扩展的 20 位（左移 12 位）立即数加到 *pc* 上，结果写入 x[*rd*]。

深圳大学 SHENZHEN UNIVERSITY

RISC-V逻辑操作指令

Instructions Name	RISC-V
XOR	xor rd, rs1, rs2
XOR Immediate	xori rd, rs1, imm
OR	or rd, rs1, rs2
OR Immediate	ori rd, rs1, imm
AND	and rd, rs1, rs2
AND Immediate	andi rd, rs1, imm

深圳大学 SHENZHEN UNIVERSITY

RISC-V比较指令

Instructions Name	RISC-V
Set <	slt rd, rs1, rs2
Set < Immediate	slti rd, rs1, imm
Set < Unsigned	sltu rd, rs1, rs2
Set < Imm Unsigned	sltiu rd, rs1, imm

slt rd, rs1, rs2 $x[rd] = (x[rs1] <_s x[rs2])$

小于则置位*(Set if Less Than)*. R-type, RV32I and RV64I.

比较 x[*rs1*]和 x[*rs2*]中的数，如果 x[*rs1*]更小，向 x[*rd*]写入 1，否则写入 0。

sltu rd, rs1, rs2 $x[rd] = (x[rs1] <_u x[rs2])$

无符号小于则置位*(Set if Less Than, Unsigned)*. R-type, RV32I and RV64I.

比较 x[*rs1*]和 x[*rs2*]，比较时视为无符号数。如果 x[*rs1*]更小，向 x[*rd*]写入 1，否则写入 0。

深圳大学 SHENZHEN UNIVERSITY

RISC-V分支指令

Instructions Name	RISC-V
Branch =	beq rs1, rs2, imm
Branch ≠	bne rs1, rs2, imm
Branch <	blt rs1, rs2, imm
Branch ≧	bge rs1, rs2, imm
Branch < Unsigned	bltu rs1, rs2, imm
Branch ≧ Unsigned	bgeu rs1, rs2, imm

beq rs1, rs2, offset　　if (rs1 == rs2) pc += sext(offset)

相等时分支 *(Branch if Equal)*. B-type, RV32I and RV64I.

若寄存器 x[*rs1*]和寄存器 x[*rs2*]的值相等，把 *pc* 的值设为当前值加上符号位扩展的偏移 *offset*。

bltu rs1, rs2, offset　　if (rs1 $<_u$ rs2) pc += sext(offset)

无符号小于时分支 *(Branch if Less Than, Unsigned)*. B-type, RV32I and RV64I.

若寄存器 x[*rs1*]的值小于寄存器 x[*rs2*]的值（均视为无符号数），把 *pc* 的值设为当前值加上符号位扩展的偏移 *offset*。

深圳大学 SHENZHEN UNIVERSITY

RISC-V跳转指令

Instructions Name	RISC-V
Jump and Link	jal rd, imm
Jump and Link Register	jalr rd, rs1, imm

jal rd, offset　　x[rd] = pc+4; pc += sext(offset)

跳转并链接 *(Jump and Link)*. J-type, RV32I and RV64I.

把下一条指令的地址(*pc*+*4*)，然后把 *pc* 设置为当前值加上符号位扩展的 *offset*。*rd* 默认为 x1。

jalr rd, offset(rs1)　　t =pc+4; pc=(x[rs1]+sext(offset))&~1; x[rd]=t

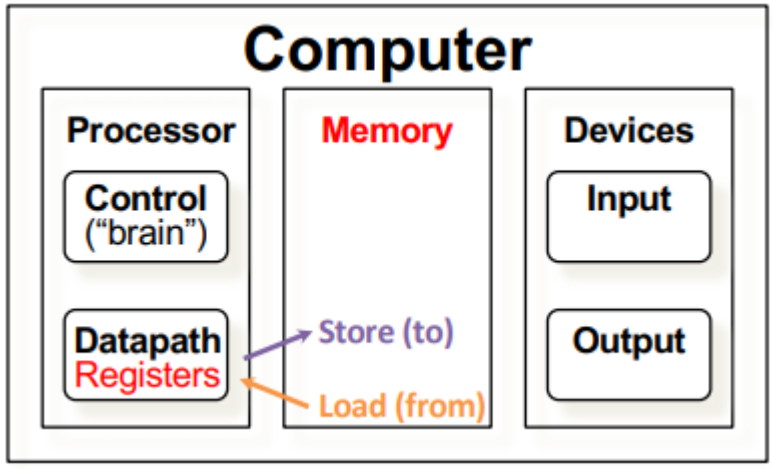
跳转并寄存器链接 *(Jump and Link Register)*. I-type, RV32I and RV64I.

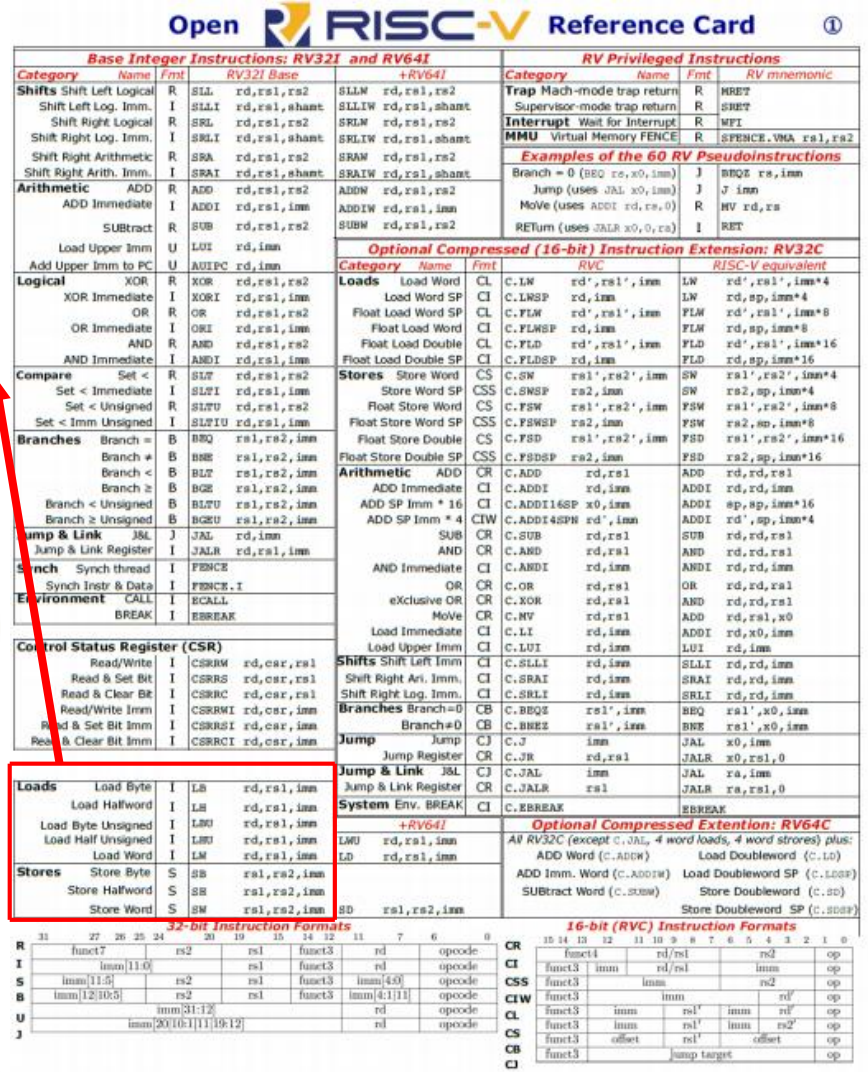
把 *pc* 设置为 x[*rs1*]+*sign-extend*(*offset*)，把计算出的地址的最低有效位设为 0，并将原 *pc*+*4* 的值写入 f[*rd*]。*rd* 默认为 x1。

RISC-V加载存储指令

Loads	Load Byte	I	LB	rd,rs1,imm
	Load Halfword	I	LH	rd,rs1,imm
	Load Byte Unsigned	I	LBU	rd,rs1,imm
	Load Half Unsigned	I	LHU	rd,rs1,imm
	Load Word	I	LW	rd,rs1,imm
Stores	Store Byte	S	SB	rs1,rs2,imm
	Store Halfword	S	SH	rs1,rs2,imm
	Store Word	S	SW	rs1,rs2,imm

Load: 数据从存储器搬移到寄存器；
Store: 数据从寄存器搬移到存储器；

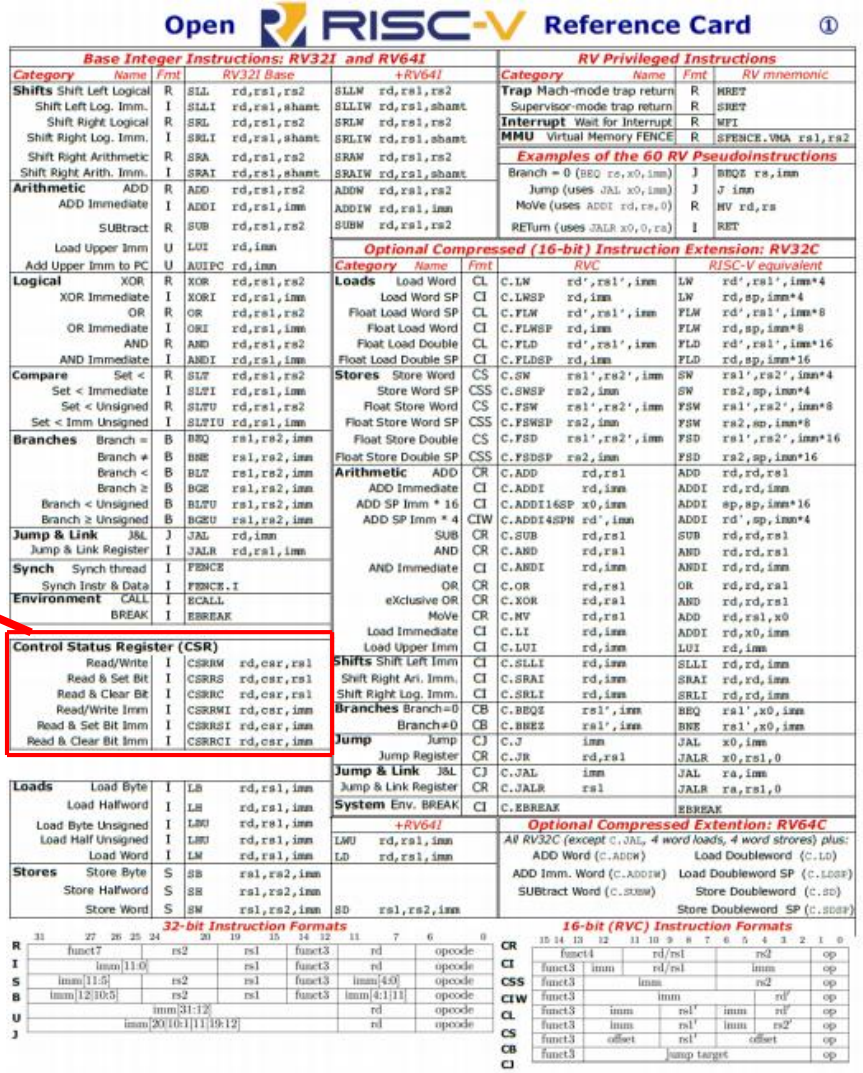




RISC-V CSR指令

Control Status Register (CSR)		
Read/Write	I	CSRRW rd,csr,rs1
Read & Set Bit	I	CSRRS rd,csr,rs1
Read & Clear Bit	I	CSRRC rd,csr,rs1
Read/Write Imm	I	CSRRWI rd,csr,imm
Read & Set Bit Imm	I	CSRRSI rd,csr,imm
Read & Clear Bit Imm	I	CSRRCI rd,csr,imm

CSR寄存器只能通过CSR指令访问



深圳大学 SHENZHEN UNIVERSITY

RISC-V伪指令

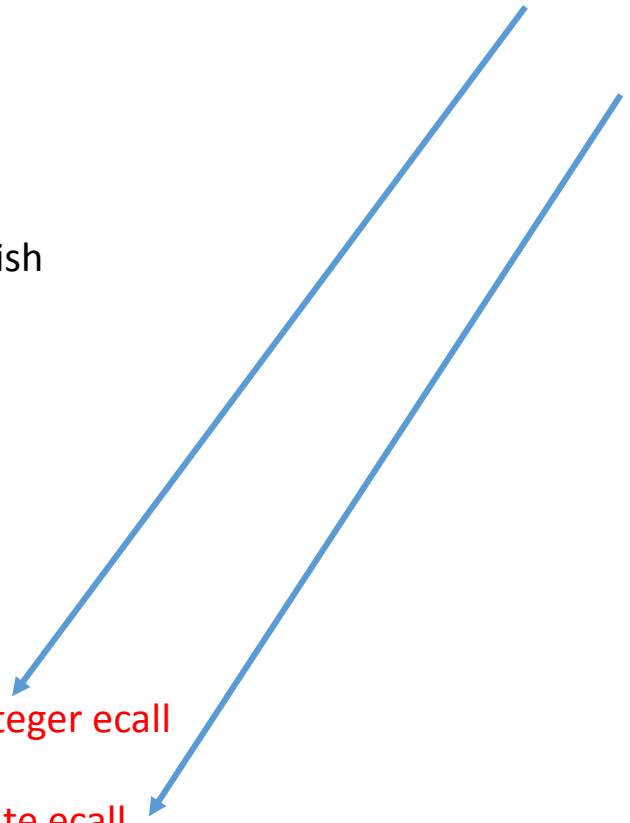
- 伪指令不是真实存在的RISC-V指令，它只是为了方便程序员理解
- 伪指令会被编译器翻译成真实的RISC-V指令
  - 伪指令move
    例子：　　mv　dst, reg1
    基础指令：　　addi　dst, reg1, 0
  - 伪指令no operation
    例子：　　nop
    基础指令：　　addi x0, x0, 0
  - 伪指令load address
    例子：　　la dst, label
    基础指令：　　auipc dst, <offset to label>
  - 伪指令jump
    例子：　　j offset
    基础指令：　　jal x0, offset
- 更多伪指令可以看《RISC-V手册》第44、45页

深圳大学 SHENZHEN UNIVERSITY

一个简单的例子

```
# Fibonacci Sequence
main:
        add   t0, x0, x0
        addi  t1, x0, 1
        la    t3, n
        lw    t3, 0(t3)
fib:
        beq   t3, x0, finish
        add   t2, t1, t0
        mv    t0, t1
        mv    t1, t2
        addi  t3, t3, -1
        j     fib
finish:
        addi  a0, x0, 1
        addi  a1, t0, 0
        ecall # print integer ecall
        addi  a0, x0, 10
        ecall # terminate ecall
```

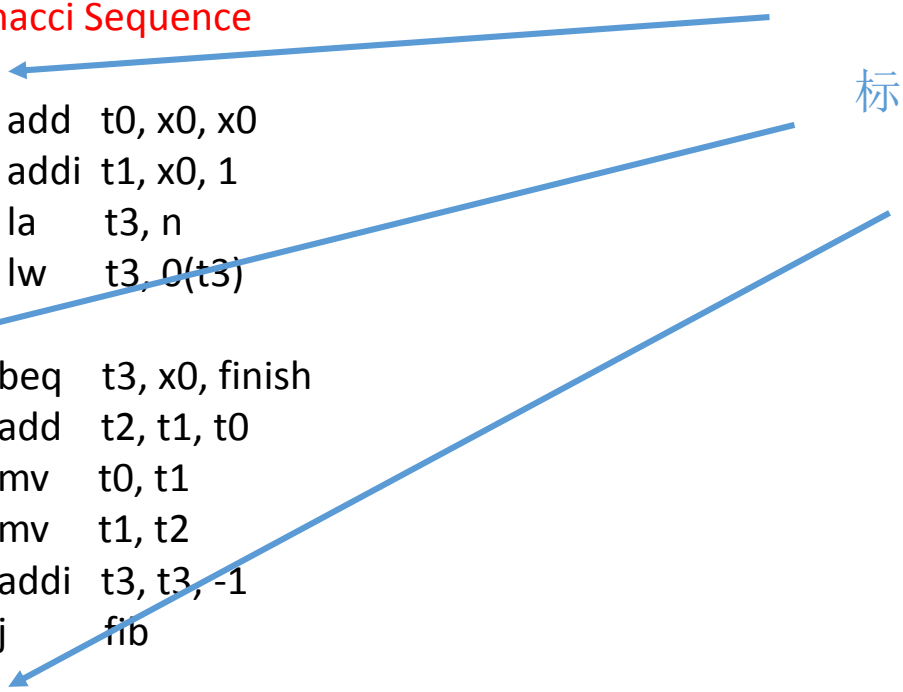
注释使用#符号



深圳大学 SHENZHEN UNIVERSITY

一个简单的例子

```
# Fibonacci Sequence
main:
        add   t0, x0, x0
        addi  t1, x0, 1
        la    t3, n
        lw    t3, 0(t3)
fib:
        beq   t3, x0, finish
        add   t2, t1, t0
        mv    t0, t1
        mv    t1, t2
        addi  t3, t3, -1
        j     fib
finish:
        addi  a0, x0, 1
        addi  a1, t0, 0
        ecall # print integer ecall
        addi  a0, x0, 10
        ecall # terminate ecall
```



标签用于标记代码段

深圳大学 SHENZHEN UNIVERSITY

一个简单的例子

```
# Fibonacci Sequence
main:
        add   t0, x0, x0
        addi  t1, x0, 1
        la    t3, n
        lw    t3, 0(t3)
fib:
        beq   t3, x0, finish
        add   t2, t1, t0
        mv    t0, t1
        mv    t1, t2
        addi  t3, t3, -1
        j     fib
finish:
        addi  a0, x0, 1
        addi  a1, t0, 0
        ecall # print integer ecall
        addi  a0, x0, 10
        ecall # terminate ecall
```

a[n] = a[n-1] + a[n-2], n>=2, a[0]=0, a[1]=1

t0=x0+x0 ===> t0=0;
t1=x0+1 ===> t1=1;
t3=address_of(n) ===> 此时t3指向n变量的地址；
t3=n ===> t3=n;

c如果t3为0，说明已经将a[n] 计算出来，可以结束循环
t2=t1+t0 ===> 斐波那契计算公式
t0=t1 ===> 更新t0
t1=t2 ===> 更新t1
t3=t3-1 ===> 计数器减1
开始下一次循环

a0=x0+1 ===> a0=1
a1=t0+0 ===> a1=t0
调用printf函数
a0=x0+10 ===> a0=10
调用exit函数

深圳大学
SHENZHEN UNIVERSITY

RISC-V汇编语法

深圳大学 SHENZHEN UNIVERSITY

汇编器指示符

Directive	Description
.text	Subsequent items are stored in the text section
.data	Subsequent items are stored in the data section
.bss	Subsequent items are stored in the bss section
.section .foo	Subsequent items are stored in the section name .foo
.align	Align to power of 2
.balign	byte align
.global sym	Declare that label sym is global
.string “str”	Store the string str in memory and null-terminate it
.type	Accepted for source compatibility
.equ	Constant definition
.macro	Begin macro definition \argname to substitute
.endm	end macro definition

深圳大学 SHENZHEN UNIVERSITY

汇编器指示符

Directive	Description
.byte b1, …, bn	Store the n 8-bit quantities in successive bytes of memory
.half w1, …, wn	Store the n 16-bit quantities in successive memory half-words
.word w1, …, wn	Store the n 32-bit quantities in successive memory words
.dword w1, …, w1	Store the n 64-bit quantities in successive memory double-words
.float f1, …, fn	Store the n single-precision floating-point numbers in successive memory words
.double d1, …, dn	Store the n double-precision floating-point numbers in successive memory words

深圳大学 SHENZHEN UNIVERSITY

汇编器重定位表达式

Directive	Description	Instruction
%hi(symbol)	Absolute (HI20)	lui
%lo(symbol)	Absolute (LO12)	load, store, add
%pcrel_hi(symbol)	PC-relative (HI20)	auipc
%pcrel_lo(label)	PC-relative (LO12)	load, store, add
%tprel_hi(symbol)	TLS LE "Local Exec"	lui
%tprel_lo(symbol)	TLS LE "Local Exec"	load, store, add
%tprel_add(symbol)	TLS LE "Local Exec"	add
%tls_ie_pcrel_hi(symbol) *	TLS IE "Initial Exec" (HI20)	auipc
%tls_gd_pcrel_hi(symbol) *	TLS GD "Global Dynamic" (HI20)	auipc
%got_pcrel_hi(symbol) *	GOT PC-relative (HI20)	auipc

深圳大学 SHENZHEN UNIVERSITY

链接器的作用

- 符号解析(symbol resolution)
  - 目标文件定义和引用符号，每个符号对应于一个函数、一个全局变量或一个静态变量（即C语言中任何以static属性声明的变量）。符号解析的目的是将每个符号引用正好和一个符号定义关联起来。

- 重定位(relocation)
  - 编译器和汇编器生成从地址0开始的代码和数据节(section)。链接器通过把每个符号定义与内存位置关联起来，从而重定位这些节，然后修改所有对这些符号的引用，是的它们指向这个内存位置。链接器使用汇编器产生的重定位条目(relocation entry)的详细信息，不加甄别地执行这样的重定位。

深圳大学 SHENZHEN UNIVERSITY

一个不简单的例子

```c
#include <stdio.h>
int main()
{
  printf("Hello, %s\n", "world");
  return 0;
}
```

链接器以一组可重定位目标文件和命令行参数作为输入，生成一个完全链接的、可以加载和运行的可执行目标文件作为输出。输入的可重定位目标文件由各种不同的代码和数据节组成，每一节都是一个连续的字节序列。

```
.text                          # Directive: enter text section
.align 2                       # Directive: align code to 2^2 bytes
.global main                   # Directive: declare global symbol main
main:                          # label for start of main
   addi  sp, sp, -16           # allocate stack frame
   sw    ra, 12(sp)            # save return address
   lui   a0, %hi(string1)      # compute address of
   addi  a0, a0, %lo(string1) #   string1
   lui   a1, %hi(string2)      # compute address of
   addi  a1, a1, %lo(string2) #   string2
   call  printf                # call function printf
   lw    ra, 12(sp)            # restore return address
   addi  sp, sp, 16            # deallocate stack frame
   li    a0, 0                 # load return value 0
   ret                         # return
.section .rodata               # Directive: enter read-only data section
.balign 4                      # Directive: align data section to 4 bytes
string1:                       # label for first string
  .string "Hello, %s!\n"       # Directive: null-terminate string
string2:                       # label for second string
  .string "world"              # Direcvive: null-terminate string
```

左图中有两个数据标签(string1和string2)和两个代码标签(main和printf)需要重定位，由于在单个32位指令中很难指定一个32位的地址，RV32I的链接器通常需要为每个标签调整两条指令。

深圳大学 SHENZHEN UNIVERSITY

一个不简单的例子

```c
#include <stdio.h>
int main()
{
  printf("Hello, %s\n", "world");
  return 0;
}
```

```
00000000 <main>:
 0: ff010113   addi   sp,sp,-16
 4: 00112623   sw     ra,12(sp)
 8: 00000537   lui    a0,0x0
 c: 00050513   mv     a0,a0
10: 000005b7   lui    a1,0x0
14: 00058593   mv     a1,a1
18: 00000097   auipc  ra,0x0
1c: 000080e7   jalr   ra
20: 00c12083   lw     ra,12(sp)
24: 01010113   addi   sp,sp,16
28: 00000513   li     a0,0
2c: 00008067   ret
```

链接器以一组可重定位目标文件和命令行参数作为输入，生成一个完全链接的、可以加载和运行的可执行目标文件作为输出。输入的可重定位目标文件由各种不同的代码和数据节组成，每一节都是一个连续的字节序列。

左图是Hello World程序经过汇编后的重定位目标文件hello.o，位置0x8到0x1c这6条指令的地址字段为0，将在后面由链接器填充。目标文件的符号表记录了链接器所需要的标签和地址。从左图可以看出：数据标签需要调整lui和addi，代码标签需要调整auipc和jalr.

深圳大学 SHENZHEN UNIVERSITY

一个不简单的例子

```c
#include <stdio.h>
int main()
{
  printf("Hello, %s\n", "world");
  return 0;
}
```

链接器以一组可重定位目标文件和命令行参数作为输入，生成一个完全链接的、可以加载和运行的可执行目标文件作为输出。输入的可重定位目标文件由各种不同的代码和数据节组成，每一节都是一个连续的字节序列。

```
000101b0 <main>:
   101b0: ff010113 addi sp,sp,-16
   101b4: 00112623 sw   ra,12(sp)
   101b8: 00021537 lui  a0,0x21
   101bc: a1050513 addi a0,a0,-1520 # 20a10 <string1>
   101c0: 000215b7 lui  a1,0x21
   101c4: a1c58593 addi a1,a1,-1508 # 20a1c <string2>
   101c8: 288000ef jal  ra,10450 <printf>
   101cc: 00c12083 lw   ra,12(sp)
   101d0: 01010113 addi sp,sp,16
   101d4: 00000513 li   a0,0
   101d8: 00008067 ret
```

左图是Hello World程序经过链接后产生的a.out文件。与之前的重定位目标文件hello.o相比，有两处修改：（1）main函数代码被赋予了一个新的地址字段；（2）位置0x8到0x1c这6条指令的地址字段被重新赋值。

深圳大學
SHENZHEN UNIVERSITY

Q & A

深圳大學
SHENZHEN UNIVERSITY

3 编译原理的基础知识

潘志铭

深圳大學
SHENZHEN UNIVERSITY

编译器的基础概念

深圳大学 SHENZHEN UNIVERSITY

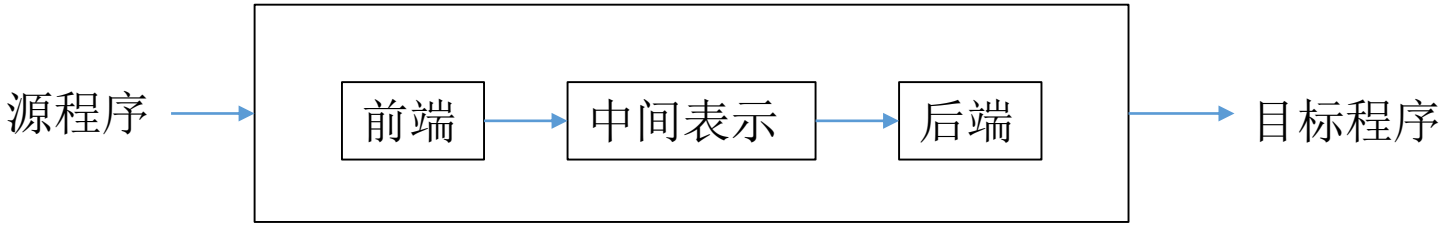
编译器的作用

- 什么是编译器？
  - 编译器是一种工具，将一种语言编写的软件转换为另一种语言；
  - 编译器将某种语言编写的程序作为输入，产生一个等价的程序作为输出；
  - C语言属于典型的编译型语言
- 什么是解释器？
  - 解释器将一种可执行程序作为输入，对外输出该程序执行后的结果；
  - Python语言属于解释性语言

深圳大学 SHENZHEN UNIVERSITY

编译器的组成

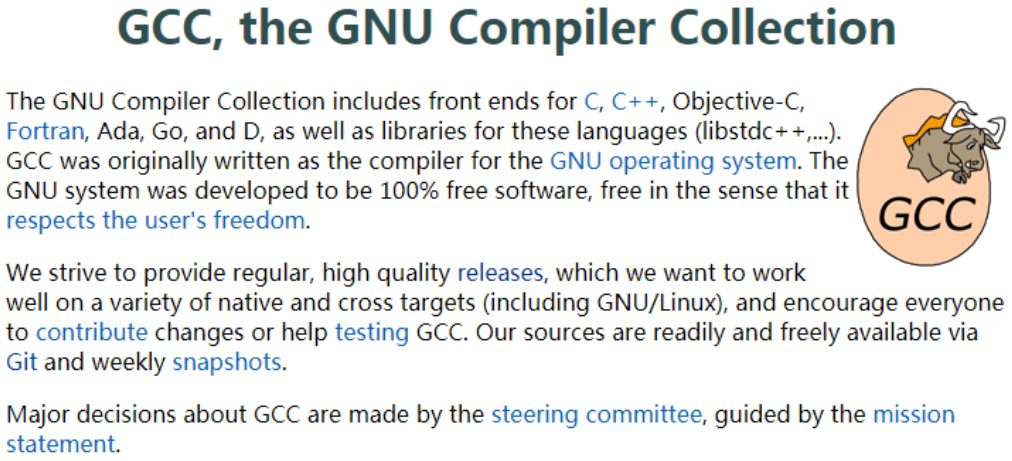
- 前端
  - 预处理，词法分析，语法分析
- 中间表示
  - 抽象语法树，中间层代码形式，中间表示优化
- 后端
  - 目标机器描述，指令选择，指令调度，寄存器分配，汇编代码生成



深圳大学 SHENZHEN UNIVERSITY

流行的编译器版本---GCC

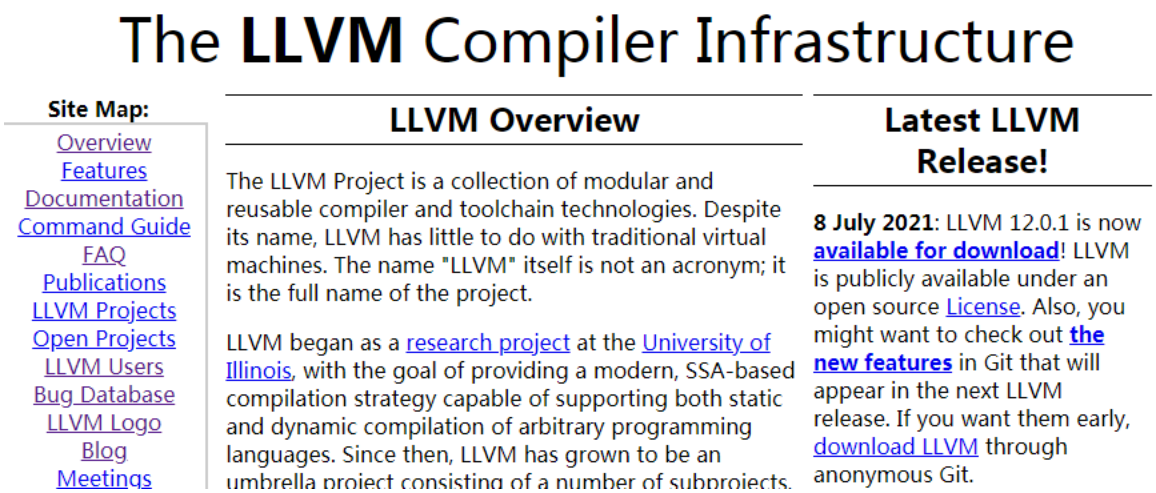
- 什么是GCC?
  - GCC 是 GNU 开发的程序语言编译器。它是一组在 GNU 通用公共许可证 (GPL) 和 GNU 宽松通用公共许可证 (LGPL) 下发布的免费软件。它是 GNU 和 Linux 系统的官方编译器，也是编译和创建其他 UNIX 操作系统的主要编译器。



深圳大学 SHENZHEN UNIVERSITY

流行的编译器版本---LLVM

- 什么是LLVM？
  - LLVM 包含一系列模块化的编译器组件和工具链。准确来说，它本身不是一个编译器，而是一个编译器框架，用于开发编译器的前端和后端。Clang 是一个支持 C/C++/Objective-C/Objective-C++的编译器，它基于 LLVM 用 C++ 构建，并在 Apache 2.0 许可下发布。


The LLVM Compiler Infrastructure

Site Map:
Overview
Features
Documentation
Command Guide
FAQ
Publications
LLVM Projects
Open Projects
LLVM Users
Bug Database
LLVM Logo
Blog
Meetings

LLVM Overview

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The name "LLVM" itself is not an acronym; it is the full name of the project.

LLVM began as a research project at the University of Illinois, with the goal of providing a modern, SSA-based compilation strategy capable of supporting both static and dynamic compilation of arbitrary programming languages. Since then, LLVM has grown to be an umbrella project consisting of a number of subprojects.

Latest LLVM Release!

8 July 2021: LLVM 12.0.1 is now **available for download**! LLVM is publicly available under an open source License. Also, you might want to check out **the new features** in Git that will appear in the next LLVM release. If you want them early, download LLVM through anonymous Git.

深圳大學
SHENZHEN UNIVERSITY

编译原理的基础概念

--- 编译器前端

深圳大学 SHENZHEN UNIVERSITY

词法分析

- 词法单词
  - 词法分析器以字符流作为输入，生成一系列名字，关键字和标点符号，同时抛弃单词之间的空白符合注释
  - 词法单词是字符组成的序列，它可以看成是程序设计语言的文法单位，并且可以归为有限的几组单词类型。

ID	foo	n14	last		
NUM	73	0	00	515	082
REAL	66.1	.5	10.	1e67	5.5e-10
IF	if				
COMMA	,				
NOTEQ	!=				
LPAREN	(				
RPAREN	)				

注释
预处理命令
预处理命令
宏
空格符、制表符和换行符

```
/* try again */
#include<stdio.h>
#define NUMS 5 , 6
NUMS
```

深圳大学 SHENZHEN UNIVERSITY

词法分析

- 词法单词

```
float match0(char *s) /* find a zero */
{if (!strncmp(s, "0.0", 3))
  return 0.;
}
```

```
FLOAT  ID(match0)  LPAREN  CHAR  STAR  ID(s)  RPAREN
LBRACE  IF  LPAREN  BANG  ID(strncmp)  LPAREN  ID(s)
COMMA  STRING(0.0)  COMMA  NUM(3)  RPAREN  RPAREN
RETURN  REAL(0.0)  SEMI  RBRACE  EOF
```

深圳大学 SHENZHEN UNIVERSITY

词法分析

- 正则表达式
  - 为了得到一个简单可读性好的词法分析器。我们将用正则表达式的形式语言来指明词法单词，用确定的有限自动机来实现词法分析器，并用数学的方法将两者联系起来。
  - 一种语言是字符串组成的集合，字符串时符号的有限序列。符号本身来自有限字母表。

深圳大学 SHENZHEN UNIVERSITY

词法分析

- 正则表达式
  - 使用正则表示用有限的描述来指明无限的语言。每个正则表达式代表一个字符串的集合
  - 符号(symbol) 可选(alternation) 联结(concatenation) ε(epsilon) 重复(repetition)

$(0\|1)^* \cdot 0$	由 2 的倍数组成的二进制数。
$b^*(abb^*)^*(a\|\epsilon)$	由 a 和 b 组成，但 a 不连续出现的字符串。
$(a\|b)^* aa(a\|b)^*$	由 a 和 b 组成，且有连续出现的 a 的字符串。

深圳大学 SHENZHEN UNIVERSITY

词法分析

- 正则表达式

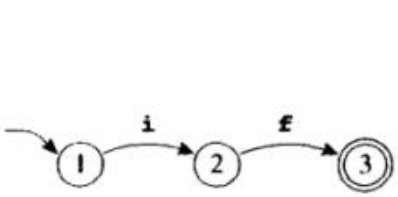
  词法分析器使用下面两条规则消除二义性

  - 最长匹配
    - 初始输入子串中，取可与任何正则表达式匹配的那个最长的字符串作为下一个单词。例如：**if8是一个标识符。**
  - 规则优先
    - 对一个特定的最长初始子串，第一个与之匹配的正则表达式决定了这个子串的单词类型。例如，**if是一个保留字**。

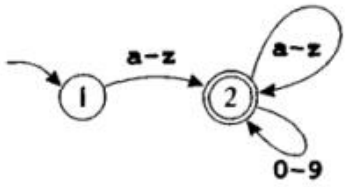
深圳大学 SHENZHEN UNIVERSITY

词法分析

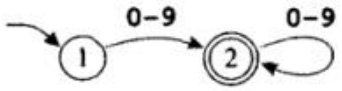
- 有限自动机
  - 有限自动机有一个有限状态集合和一些从一个状态通向另一个状态的边，每条边上标记有一个符号；其中一个状态是初态，某些状态是终态。



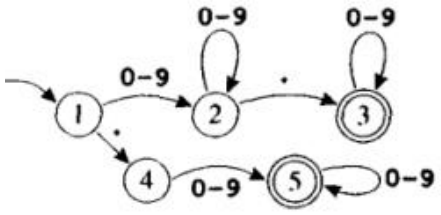
IF



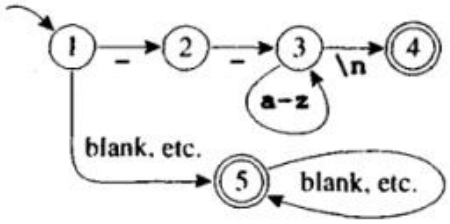
ID



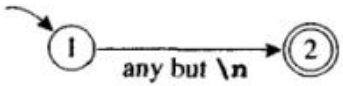
NUM



REAL



white space

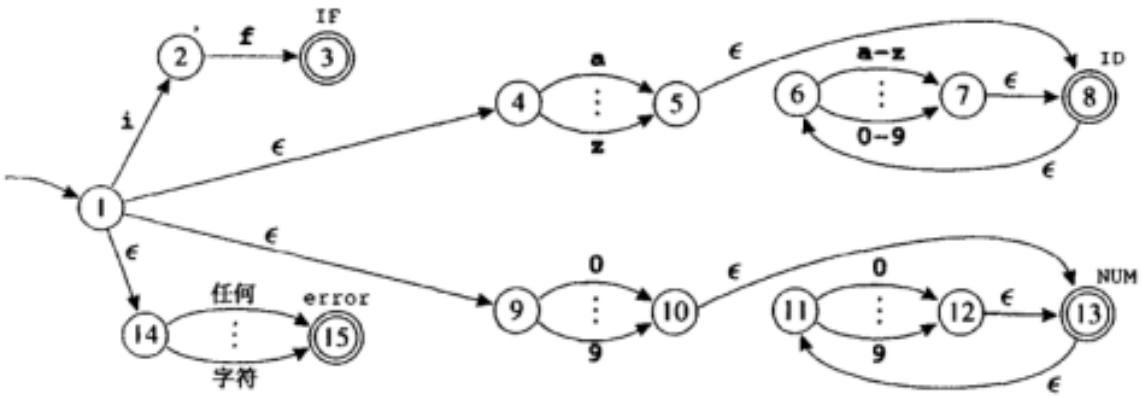


error

深圳大学 SHENZHEN UNIVERSITY

词法分析

- 非确定有限自动机
  - 确定的有限自动机(DFA)不会有从同一状态触发的两条边标记有相同的符号。
  - 分确定的有限自动机(NFA)是一种需要对从一个状态触发的多条标有相同符号的边进行选择的自动机。
  - 可以很容易将一个正则表达式转换成一个NFA



词法分析器

- Lex: 词法分析器的生成器

```
%{
/* C Declarations: */
#include "tokens.h"    /* definitions of IF, ID, NUM, ... */
#include "errormsg.h"
union {int ival; string sval; double fval;} yylval;
int charPos=1;
#define ADJ  (EM_tokPos=charPos, charPos+=yyleng)
%}
/* Lex Definitions: */
digits  [0-9]+
%%
/* Regular Expressions and Actions: */
if                          {ADJ; return IF;}
[a-z][a-z0-9]*              {ADJ; yylval.sval=String(yytext);
                              return ID;}
{digits}                   {ADJ; yylval.ival=atoi(yytext);
                              return NUM;}
({digits}"."[0-9]*)|([0-9]*"."{digits})      {ADJ;
                              yylval.fval=atof(yytext);
                              return REAL;}
("--"[a-z]*"\n")|(" "|"\n"|"\t")+     {ADJ;}
                            {ADJ; EM_error("illegal character");}
```

深圳大学 SHENZHEN UNIVERSITY

语法分析

- 上下文无关文法
  - 有限自动机缺少递归的表示方法。
  - 上下文无关文法(context-free grammar)以说明的方式来定义语法。

1 $S \rightarrow S\ ;\ S$
2 $S \rightarrow \text{id} := E$
3 $S \rightarrow \text{print}\ (\ L\ )$
4 $E \rightarrow \text{id}$
5 $E \rightarrow \text{num}$
6 $E \rightarrow E + E$
7 $E \rightarrow (S\ ,\ E\ )$
8 $L \rightarrow E$
9 $L \rightarrow L\ ,\ E$

深圳大学 SHENZHEN UNIVERSITY

语法分析

- 上下文无关文法
  - 推导有限自动机从开始符号触发对其右边的每一个非终结符，用非终结符对应的产生式中的任一右部来替换它。

$\underline{S}$
S ; $\underline{S}$
$\underline{S}$; id : = E
id : = $\underline{E}$; id : = E
id : = num ; id : = $\underline{E}$
id : = num ; id : = E + $\underline{E}$
id : = num ; id : = $\underline{E}$ + (S , E)
id : = num ; id : = id + ($\underline{S}$, E)
id : = num ; id : = id + (id : = $\underline{E}$, E)
id : = num ; id : = id + (id : = E + E , $\underline{E}$)
id : = num ; id : = id + (id : = $\underline{E}$ + E , id)
id : = num ; id : = id + (id : = num + $\underline{E}$, id)
id : = num ; id : = id + (id : = num + num , id)

最左推导：一种总是扩展最左边非终结符的推导。

最右推导：一种总是扩展最右边非终结符的推导。

深圳大学 SHENZHEN UNIVERSITY

语法分析

- 预测分析
  - 递归下降分析也称预测分析，适合于，每个子表达式的第一个终结符号能够为产生式的选择提供足够多的信息。
  - 有时使用语法分析器生成工具并不方便，预测分析器的有点就在于其算法简单，可以用它手工构造分析器。

深圳大学 SHENZHEN UNIVERSITY

语法分析

- 预测分析

$S \rightarrow$ if E then S else S
$S \rightarrow$ begin S L
$S \rightarrow$ print E

$L \rightarrow$ end
$L \rightarrow$; S L

$E \rightarrow$ num = num

```
enum token {IF, THEN, ELSE, BEGIN, END, PRINT, SEMI, NUM, EQ};
extern enum token getToken(void);

enum token tok;
void advance() {tok=getToken();}
void eat(enum token t) {if (tok==t) advance(); else error();}

void S(void) {switch(tok) {
        case IF:    eat(IF); E(); eat(THEN); S();
                                  eat(ELSE); S(); break;
        case BEGIN: eat(BEGIN); S(); L(); break;
        case PRINT: eat(PRINT); E(); break;
        default:    error();
       }}
void L(void) {switch(tok) {
        case END:   eat(END); break;
        case SEMI:  eat(SEMI); S(); L(); break;
        default:    error();
       }}
```

1. 构造预测分析表
2. 消除多重定义
3. 递归下降分析器

深圳大学 SHENZHEN UNIVERSITY

语法分析

- 预测分析

1. 构造预测分析表
2. 消除多重定义
3. 递归下降分析器

$S \rightarrow E\,\$$

$E \rightarrow T\,E'$

$E' \rightarrow +\,T\,E'$
$E' \rightarrow -\,T\,E'$
$E' \rightarrow$

$T \rightarrow F\,T'$

$T' \rightarrow *\,F\,T'$
$T' \rightarrow /\,F\,T'$
$T' \rightarrow$

$F \rightarrow \text{id}$
$F \rightarrow \text{num}$
$F \rightarrow (\,E\,)$

	+	*	id	(	)	$
S			$S \rightarrow E\$$	$S \rightarrow E\$$		
E			$E \rightarrow TE'$	$E \rightarrow TE'$		
E'	$E' \rightarrow +TE'$				$E' \rightarrow$	$E' \rightarrow$
T			$T \rightarrow FT'$	$T \rightarrow FT'$		
T'	$T' \rightarrow$	$T' \rightarrow *FT'$			$T' \rightarrow$	$T' \rightarrow$
F			$F \rightarrow \text{id}$	$F \rightarrow (E)$		

深圳大学 SHENZHEN UNIVERSITY

语法分析

- LR分析
  - LR(k)：从左至右分析、最右推导、超前查看k个单词

栈	输入	动作
	a := 7 ; b := c + (d := 5 + 6 , d) $	移进
id_4	:= 7 ; b := c + (d := 5 + 6 , d) $	移进
id_4 :=$_6$	7 ; b := c + (d := 5 + 6 , d) $	移进
id_4 :=$_6$ num_{10}	; b := c + (d := 5 + 6 , d) $	归约$E \rightarrow$ num
id_4 :=$_6$ E_{11}	; b := c + (d := 5 + 6 , d) $	归约$S \rightarrow$ id:=E
S_2	; b := c + (d := 5 + 6 , d) $	移进
S_2 ;$_3$	b := c + (d := 5 + 6 , d) $	移进
S_2 ;$_3$ id_4	:= c + (d := 5 + 6 , d) $	移进
S_2 ;$_3$ id_4 :=$_6$	c + (d := 5 + 6 , d) $	移进
S_2 ;$_3$ id_4 :=$_6$ id_{20}	+ (d := 5 + 6 , d) $	归约$E \rightarrow$ id
S_2 ;$_3$ id_4 :=$_6$ E_{11}	+ (d := 5 + 6 , d) $	移进
S_2 ;$_3$ id_4 :=$_6$ E_{11} +$_{16}$	(d := 5 + 6 , d) $	移进
S_2 ;$_3$ id_4 :=$_6$ E_{11} +$_{16}$ (8	d := 5 + 6 , d) $	移进
S_2 ;$_3$ id_4 :=$_6$ E_{11} +$_{16}$ (8 id_4	:= 5 + 6 , d) $	移进
S_2 ;$_3$ id_4 :=$_6$ E_{11} +$_{16}$ (8 id_4 :=$_6$	5 + 6 , d) $	移进
S_2 ;$_3$ id_4 :=$_6$ E_{11} +$_{16}$ (8 id_4 :=$_6$ num_{10}	+ 6 , d) $	归约$E \rightarrow$ num
S_2 ;$_3$ id_4 :=$_6$ E_{11} +$_{16}$ (8 id_4 :=$_6$ E_{11}	+ 6 , d) $	移进
S_2 ;$_3$ id_4 :=$_6$ E_{11} +$_{16}$ (8 id_4 :=$_6$ E_{11} +$_{16}$	6 , d) $	移进
S_2 ;$_3$ id_4 :=$_6$ E_{11} +$_{16}$ (8 id_4 :=$_6$ E_{11} +$_{16}$ num_{10}	, d) $	归约$E \rightarrow$ num
S_2 ;$_3$ id_4 :=$_6$ E_{11} +$_{16}$ (8 id_4 :=$_6$ E_{11} +$_{16}$ E_{17}	, d) $	归约$E \rightarrow E + E$
S_2 ;$_3$ id_4 :=$_6$ E_{11} +$_{16}$ (8 id_4 :=$_6$ E_{11}	, d) $	归约$S \rightarrow$ id:=E

语法分析

- 分析器生成器

1 $P \rightarrow L$

2 $S \rightarrow$ id := id
3 $S \rightarrow$ while id do S
4 $S \rightarrow$ begin L end
5 $S \rightarrow$ if id then S
6 $S \rightarrow$ if id then S else S

7 $L \rightarrow S$
8 $L \rightarrow L$; S

```
%{
int yylex(void);
void yyerror(char *s) { EM_error(EM_tokPos, "%s", s); }
%}
%token ID WHILE BEGIN END DO IF THEN ELSE SEMI ASSIGN
%start prog
%%

prog: stmlist

stm : ID ASSIGN ID
    | WHILE ID DO stm
    | BEGIN stmlist END
    | IF ID THEN stm
    | IF ID THEN stm ELSE stm

stmlist : stm
        | stmlist SEMI stm
```

深圳大学 SHENZHEN UNIVERSITY

上下文相关分析

- 概述
  - 语法正确的输入程序仍然可能包含严重的错误，导致编译无法完成。为了检查这样的错误，编译器需要进行更深层的检查，其中设计每条语句放到实际的上下文中进行考虑。
  - 为积累进一步转换需要的上下文知识，编译器必须开发出一些方法，从语法之外的视角来考察程序。

深圳大学 SHENZHEN UNIVERSITY

上下文相关分析

- 类型系统
  - 大多数程序设计语言都将一组性质关联到每个数据值，这些性质的集合成为值的类型。与上下文无关语法相比，利用类型系统可以在更精确的层次上规定程序的行为。

1. 确保运行时的安全性
2. 提高表达力
3. 生成更好的代码
4. 类型检查

深圳大学 SHENZHEN UNIVERSITY

上下文相关分析

- 属性语法
  - 属性语法的是用于上下文相关分析的一种形式化机制。属性语法包 括一个上下文无关语法，外加一组规定了某些计算的规则。每个规则都 通过其他属性的值定义了一个值或属性。规则将属性关联到一个特定的语法符号，出现在语法分析树中的每个语法符号实例都有一个对应的属 性实例。规则是功能性的，没有蕴涵特定的求值次序，且唯一地定义了每个属性值。

深圳大学 SHENZHEN UNIVERSITY

上下文相关分析

- 属性语法

$$
P = \left\{
\begin{array}{lcl}
Number & \rightarrow & Sign\ List \\
Sign & \rightarrow & + \\
 & | & - \\
List & \rightarrow & List\ Bit \\
 & | & Bit \\
Bit & \rightarrow & 0 \\
 & | & 1
\end{array}
\right\}
$$

$$T = \{+, -, 0, 1\}$$

$$NT = \{Number,\ Sign,\ List,\ Bit\}$$

$$S = \{Number\}$$

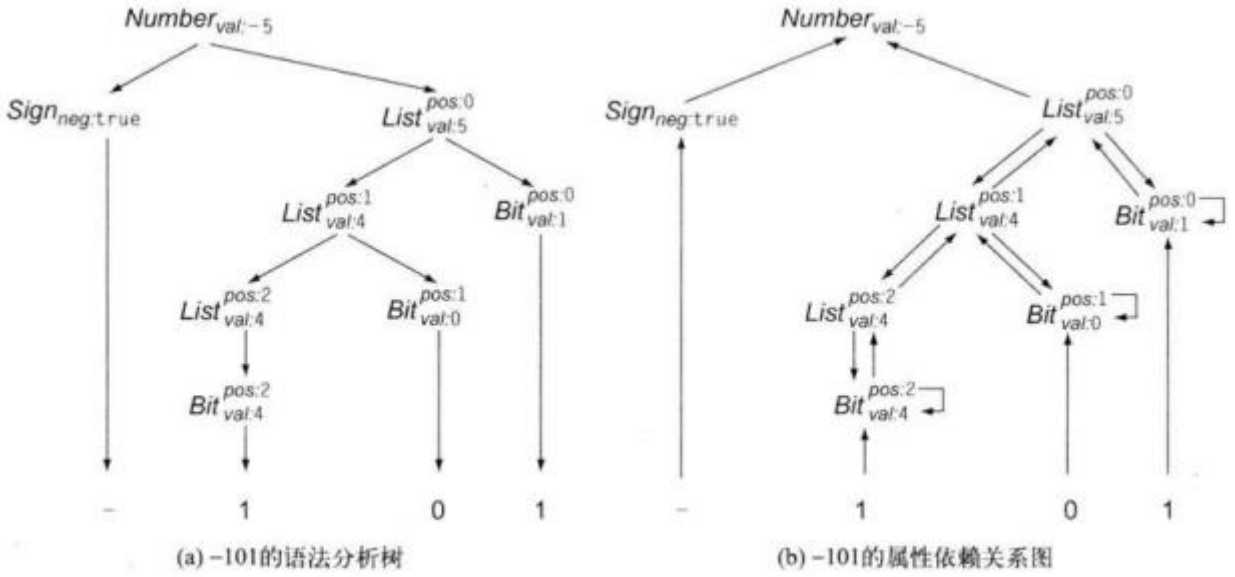
符号	属　性
Number	*value*
Sign	*negative*
List	*position, value*
Bit	*position, value*

	产 生 式	属性规则
1	*Number* → *Sign List*	*List.position* ← 0 *if Sign.negative* *then Number.value* ← −*List.value* *else Number.value* ← *List.value*
2	*Sign* → +	*Sign.negative* ← *false*
3	*Sign* → −	*Sign.negative* ← *true*
4	*List* → *Bit*	*Bit.position* ← *List.position* *List.value* ← *Bit.value*
5	$List_0 \rightarrow List_1\ Bit$	$List_1.position \leftarrow List_0.position + 1$ *Bit.position* ← $List_0.position$ $List_0.value \leftarrow List_1.value + Bit.value$
6	*Bit* → 0	*Bit.value* ← 0
7	*Bit* → 1	$Bit.value \leftarrow 2^{Bit.position}$

深圳大学
SHENZHEN UNIVERSITY

上下文相关分析

- 属性语法



(a) −101的语法分析树　　(b) −101的属性依赖关系图

图4-6　−101的属性化语法分析树

深圳大學
SHENZHEN UNIVERSITY

Q & A

深圳大学
SHENZHEN UNIVERSITY

4 RISC-V工具链的开发

潘志铭

深圳大学 SHENZHEN UNIVERSITY

为什么要进行工具链开发？

- RISC-V指令集的特点是什么？
  - 开源；模块化；可扩展性强
- 什么情况下需要设计自定义指令？
  - 特定应用场景：如PS5游戏加速指令
- 硬件实现自定义指令后，软件如何使用？
  - 修改GNU工具链，增加对自定义指令的支持

深圳大学
SHENZHEN UNIVERSITY

如何进行工具链开发？

- 1. 描述自定义指令
- 2. 确定指令码
- 3. 为binutils增加自定义指令
- 4. 为gcc增加自定义指令
- 5. 在spike上增加自定义指令
- 6. 重新构建toolchain和spike
- 7. 测试toolchain和spike

深圳大学
SHENZHEN UNIVERSITY

描述自定义指令

- 增加一条乘加指令mac，具体描述如下：

汇编指令	mac rd, rs1, rs2
软件模型	rd = rd + rs1 * rs2
例子	mac a2, a0, a1

确定指令码

- 依据 RISC-V SPEC， mac 属于 R-Type 指令

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

```
 31        25 24       20 19       15 14    12 11       7 6         0
 ------------------------------------------------------------------------
|    funct7  |    rs2    |    rs1    | funct3 |    rd    |    OPCODE   |
 ------------------------------------------------------------------------
       7          5          5         3         5           7
```

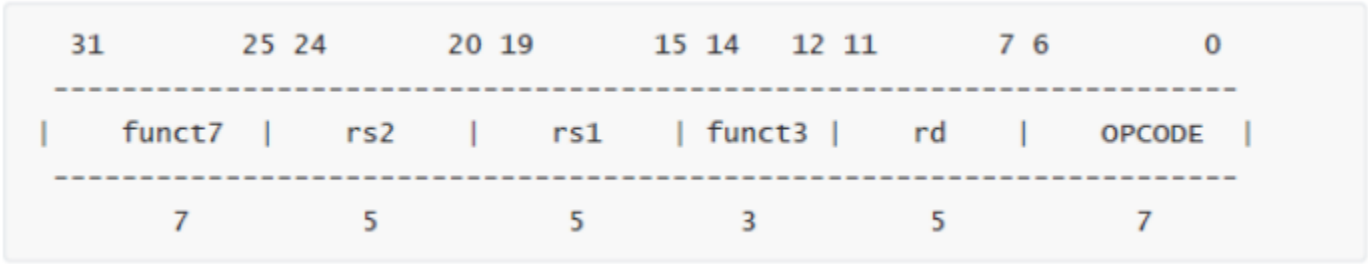
深圳大学 SHENZHEN UNIVERSITY

确定指令码

- 确定OPCODE

RISC-V SPEC规定了四种自定义指令的OPCODE，分别是7'b000_1011，7'b010_1011，7'b101_1011，7'b111_1011，这里我们选用CUSTOM0编码7'b000_1011

inst[4:2] / inst[6:5]	000	001	010	011	100	101	110	111 (> 32b)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	48b
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	64b
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	48b
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	≥ 80b



深圳大学 SHENZHEN UNIVERSITY

确定指令码

- 确定funct3

为了进一步规范自定义指令码格式，我们可以约定funct3的编码格式：3’b010表示只使用了RS1寄存器，3’b011表示只使用了RS1和RS2 寄存器，3’b100表示只使用了RD寄存器，3’b110表示只使用RD和RS1寄存器，而3’b111则表示RD、RS1和RS2寄存器都用到了，所以这里的funct3应该是3’b111.

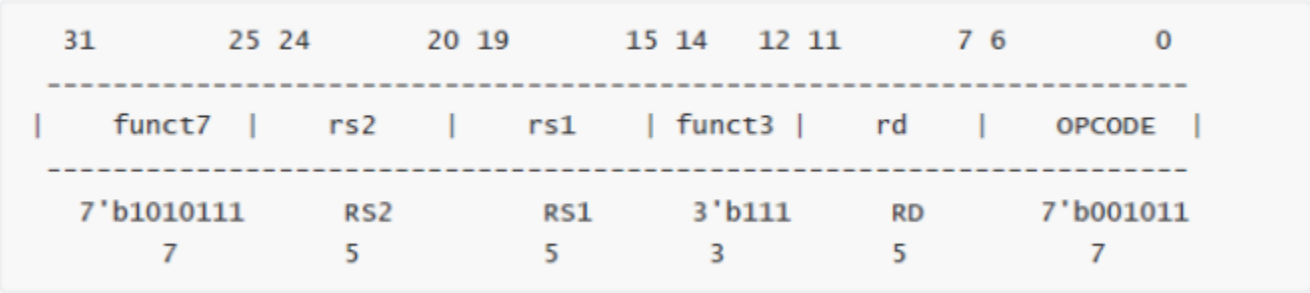
funct3	rd	rs1	rs2	Symbol
3'b000	0	0	0	CUSTOMX
3'b001	0	0	1	保留
3'b010	0	1	0	CUSTOMX_RS1
3'b011	0	1	1	CUSTOMX_RS1_RS2
3'b100	1	0	0	CUSTOMX_RD
3'b101	1	0	1	保留
3'b110	1	1	0	CUSTOMX_RD_RS1
3'b111	1	1	1	CUSTOMX_RD_RS1_RS2

深圳大学 SHENZHEN UNIVERSITY

确定指令码

- 最终确定的指令码

由于funct7没有特别的约束，我们这里选定为7’b101_0111.



深圳大学 SHENZHEN UNIVERSITY

为binutils增加自定义指令

- 修改riscv-opc.c文件和riscv-opc.h文件：

```
diff --git a/include/opcode/riscv-opc.h b/include/opcode/riscv-opc.h
index 7bdc7e4..c68d1c7 100644
--- a/include/opcode/riscv-opc.h
+++ b/include/opcode/riscv-opc.h
@@ -644,6 +644,9 @@
 #define MATCH_VGHASH_V 0x4a0fa057
 #define MASK_VGHASH_V  0xfe0ff07f

+#define MATCH_MAC 0xae00700b
+#define MASK_MAC  0xfe00707f
 /* Temporary Load/store encoding info
 MOP load
 00 unit-stride VLE<EEW>, VLE<EEW>FF, VL<nf>RE<EEW> (nf = 1, 2, 4, 8)
diff --git a/opcodes/riscv-opc.c b/opcodes/riscv-opc.c
index 43714eb..26a3db2 100644
--- a/opcodes/riscv-opc.c
+++ b/opcodes/riscv-opc.c
@@ -897,6 +897,7 @@ const struct riscv_opcode riscv_opcodes[] =
 {"divuw",    64, INSN_CLASS_M, "d,s,t",  MATCH_DIVUW, MASK_DIVUW,
match_opcode, 0 },
 {"remw",     64, INSN_CLASS_M, "d,s,t",  MATCH_REMW, MASK_REMW,
match_opcode, 0 },
 {"remuw",    64, INSN_CLASS_M, "d,s,t",  MATCH_REMUW, MASK_REMUW,
match_opcode, 0 },
+{"mac",       0, INSN_CLASS_M, "d,s,t",  MATCH_MAC, MASK_MAC, match_opcode,
0 },

 /* Half-precision floating-point instruction subset */
 {"flh",        0, INSN_CLASS_F_AND_ZFH,   "D,o(s)",  MATCH_FLH, MASK_FLH,
match_opcode, INSN_DREF|INSN_2_BYTE },
```

汇编器需要定义一些宏来确定指令编码，其中有两个宏是一定要定义的，分别是 MATCH 和 MASK，其中MATCH宏定义用来识别指令码，MASK宏定义用来获取指令码，在生成指令码时，汇编器会这样使用这些宏定义：

((insn ^ MATCH) & MASK) == 0.

为gcc增加自定义指令

- 修改riscv.md文件：

```
diff --git a/gcc/config/riscv/riscv.md b/gcc/config/riscv/riscv.md
index 8cfac79..c1b72ee 100644
--- a/gcc/config/riscv/riscv.md
+++ b/gcc/config/riscv/riscv.md
@@ -750,6 +750,25 @@
   [(set_attr "type" "imul")
    (set_attr "mode" "SI")])

+(define_insn "macsi3"
+  [(set (match_operand:SI          0 "register_operand" "=r")
+        (plus: SI (mult:SI (match_operand:SI 2 "register_operand" " r")
+                           (match_operand:SI 3 "register_operand" " r"))
+                  (match_operand: SI 1 "register_operand" "0")))]
+  "TARGET_MUL"
+  "mac\t%0,%2,%3"
+  [(set_attr "type" "imul")
+   (set_attr "mode" "SI")])
+
+(define_insn "macdi3"
+  [(set (match_operand:DI          0 "register_operand" "=r")
+        (plus: DI (mult:DI (match_operand:DI 2 "register_operand" " r")
+                           (match_operand:DI 3 "register_operand" " r"))
+                  (match_operand: DI 1 "register_operand" "0")))]
+  "TARGET_MUL && TARGET_64BIT"
+  "mac\t%0,%2,%3"
+  [(set_attr "type" "imul")
+   (set_attr "mode" "DI")])
 ;;
 ;;  ....................
 ;;
```

为了让 GCC 识别 mac 指令，我们需要在 GCC 后端添加指令模板，由于 mac 指令可以描述成乘、加两种指令的混合体，所以我们可以直接使用 GCC 中自带的 SPN 来描述指令模板（对于 AES、SHA 这种加密指令，是无法直接使用 GCC 中自带的 SPN 来描述的，只能用 intrinsic，然后通过 intrinsic 与 GCC 后端的对应指令模板做匹配），经过修改后的 riscv.md 文件如左图所示。

深圳大学 SHENZHEN UNIVERSITY

在spike上增加自定义指令

- 修改encoding.h文件

```
diff --git a/riscv/encoding.h b/riscv/encoding.h
index 9cbb271..42391d2 100644
--- a/riscv/encoding.h
+++ b/riscv/encoding.h
@@ -1844,6 +1844,10 @@
 #define MASK_VL4R_V  0xfff0707f
 #define MATCH_VL8R_V 0x1e807007
 #define MASK_VL8R_V  0xfff0707f
+#define MATCH_MAC 0xae00700b
+#define MASK_MAC  0xfe00707f
 #define CSR_FFLAGS 0x1
 #define CSR_FRM 0x2
 #define CSR_FCSR 0x3
@@ -2921,6 +2925,7 @@ DECLARE_INSN(vl1r_v, MATCH_VL1R_V, MASK_VL1R_V)
 DECLARE_INSN(vl2r_v, MATCH_VL2R_V, MASK_VL2R_V)
 DECLARE_INSN(vl4r_v, MATCH_VL4R_V, MASK_VL4R_V)
 DECLARE_INSN(vl8r_v, MATCH_VL8R_V, MASK_VL8R_V)
+DECLARE_INSN(mac, MATCH_MAC, MASK_MAC)
 #endif
 #ifdef DECLARE_CSR
 DECLARE_CSR(fflags, CSR_FFLAGS)
```

spike 是一款 RISC-V 指令模拟器，我们可以在 spike 上增加自定义指令，运行包含自定 义指令 mac 的应用程序，从而验证修改后的 GNU 工具链是否能够正常工作。

深圳大学 SHENZHEN UNIVERSITY

在spike上增加自定义指令

- 增加指令描述：
  - 在riscv/insns目录下新建文件mac.h，内容如下：

```
require_extension('M');
WRITE_RD(sext_xlen(RD + RS1 * RS2));
```

  - 修改riscv/riscv.mk.in文件：

```
diff --git a/riscv/riscv.mk.in b/riscv/riscv.mk.in
index d4422fe..aa6f68b 100644
--- a/riscv/riscv.mk.in
+++ b/riscv/riscv.mk.in
@@ -195,6 +195,7 @@ riscv_insn_ext_m = \
        remu \
        remuw \
        remw \
+       mac \

 riscv_insn_ext_f = \
        fadd_s \
```

深圳大学 SHENZHEN UNIVERSITY

重新构建toolchain和spike

- 重新构建toolchain

```
$ cd riscv-gnu-toolchain
$ mkdir build && cd build
$ ../configure --prefix=/opt/riscv --enable-multilib
$ make && make install
```

- 重新构建spike

```
$ cd riscv-isa-sim
$ mkdir build && cd build
$ ../configure –prefix=/opt/riscv -enable-histogram
$ make && make install
```

深圳大學 SHENZHEN UNIVERSITY

测试toolchain和spike

- 编写测试文件main.c

```c
#include <stdio.h>

int32_t op1 = 6;
int32_t op2 = 7;
int32_t res = 1;

int main(void)
{
 res += op1 * op2;
 printf("res = %d\n", res);

 return 0;
}
```

测试toolchain和spike

- 测试步骤

$ riscv64-unknown-elf-gcc -O2 main.c -o main.elf

$ riscv64-unknown-elf-objdump -D main.elf > main.asm

$ spike pk main.elf

```
00000000000100b0 <main>:
   100b0:       7501a603                lw      a2,1872(gp) # 1f588 <op1>
   100b4:       7481a783                lw      a5,1864(gp) # 1f580 <res>
   100b8:       74c1a683                lw      a3,1868(gp) # 1f584 <op2>
   100bc:       6571                    lui     a0,0x1c
   100be:       1141                    addi    sp,sp,-16
   100c0:       aed6778b                mac     a5,a2,a3
   100c4:       65050513                addi    a0,a0,1616 # 1c650 <__clzdi2+0
   100c8:       e406                    sd      ra,8(sp)
   100ca:       0007859b                sext.w  a1,a5
   100ce:       74f1a423                sw      a5,1864(gp) # 1f580 <res>
   100d2:       200000ef                jal     ra,102d2 <printf>
   100d6:       60a2                    ld      ra,8(sp)
   100d8:       4501                    li      a0,0
   100da:       0141                    addi    sp,sp,16
   100dc:       8082                    ret
```

深圳大學
SHENZHEN UNIVERSITY

Q & A