

课程实验报告

RISC-V on T-Core

MaTrixV Team

目录

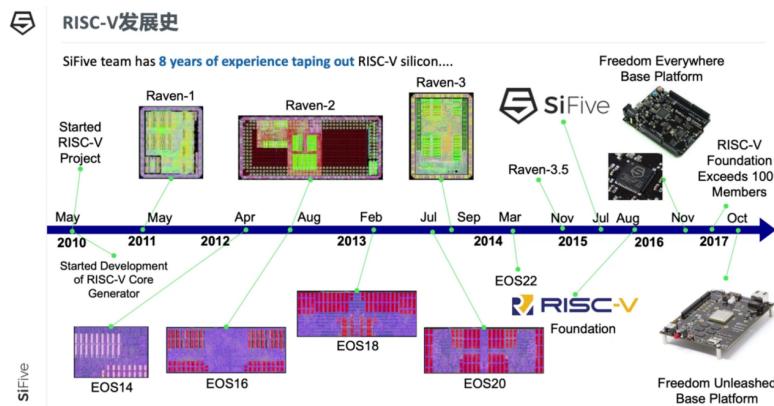
1.	基础篇	2
1.1	RISC-V 简介	2
1.2	蜂鸟 E203 简介	10
1.3	T-core 开发板介绍	11
2.	实践篇	12
3.	结果展示	12
4.	未来展望	12

1. 基础篇

1.1 RISC-V 简介

RISC-V 发展过程

1. RISC(精简指令集计算机) 和 CISC(复杂指令集计算机) 是当前 CPU 的两种架构。早些年，市面上只有 CISC 指令集，后来 IBM 的研究员通过统计的方法发现，传统 CISC 处理器中，五分之一的指令承担了五分之四的工作，而剩下五分之四的指令基本没有被使用，或者很少使用，这样，既浪费了 CPU 的核心面积，增大了功耗，还降低了效率。于是，RISC 应运而生。
2. RISC 的指令数目较 CISC 少，CISC 中的一些复杂指令，RISC 需要用多条简单指令来实现。但指令字等长，效率高，功耗低，并发性高。且内部寄存器丰富，更强调对寄存器的合理调用。但高性能 RISC 处理器成本高，性价比低，且不同公司的 RISC 芯片几乎无法通用，生态环境较 X86 的 CISC 而言更闭塞，通用性完全无法和 X86 相比，这就是 RISC 最大的弊端。
3. 20 世纪末和 21 世纪初，市面上绝大多数核心指令集都是不开源的。2010 年，加州大学伯克利分校的 David A. Patterson 教授团队在 3 个月内开发出完全开源指令集 RISC-V，RISC-V 指令集是基于精简指令集计算 (RISC) 原理建立的开放指令集架构 (ISA)，RISC-V 是在指令集不断发展和成熟的基础上建立的全新指令。RISC-V 指令集完全开源，设计简单，易于移植 Unix 系统，模块化设计，完整工具链，同时有大量的开源实现和流片案例，已在社区得到大力支持。
4. 它虽然不是第一个开源的的指令集 (ISA)，但它是第一个被设计成可以根据具体场景可以选择适合的指令集的指令集架构。基于 RISC-V 指令集架构可以设计服务器 CPU、家用电器 CPU、工控 CPU 和传感器中的 CPU 等。



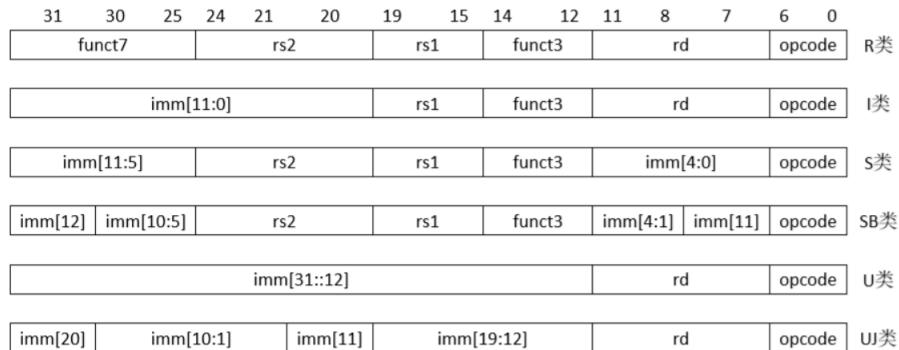
RISC-V 指令简述

1. RSICV 指令集分为基本指令集 I 和扩展指令集 M, A, F, D, C。基本指令集 I 是整数指令集，也是 RISC-V 中，对于任何处理器必须有的指令集，扩展指令集可有可无。

基本指令集	指令数	描述
RV32I	47	32位地址空间与整数指令，支持32个通用整数寄存器
RV32E	47	RV32I的子集，仅支持16个通用整数寄存器
RV64I	59	64位地址空间与整数指令及一部分32位的整数指令
RV128I	71	128位地址空间与整数指令及一部分64位和32位的指令
扩展指令集	指令数	描述
M	8	整数乘法与除法指令
A	11	存储器原子 (Atomic) 操作指令和Load-Reserved/Store-Conditional指令
F	26	单精度 (32比特) 浮点指令
D	26	双精度 (64比特) 浮点指令，必须支持F扩展指令

2. 基本指令集有六种格式：

- (a) R 类型指令：用于寄存器 - 寄存器操作；
- (b) I 类型指令：用于短立即数和访存 load 操作；
- (c) S 类型指令：用于访存 store 操作；
- (d) B 类型指令：用于条件跳转操作；
- (e) U 类型指令：用于长立即数操作；
- (f) J 类型指令：用于无条件操作；

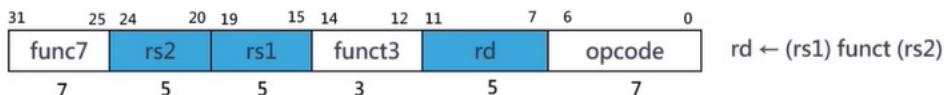


RISC-V 指令详述

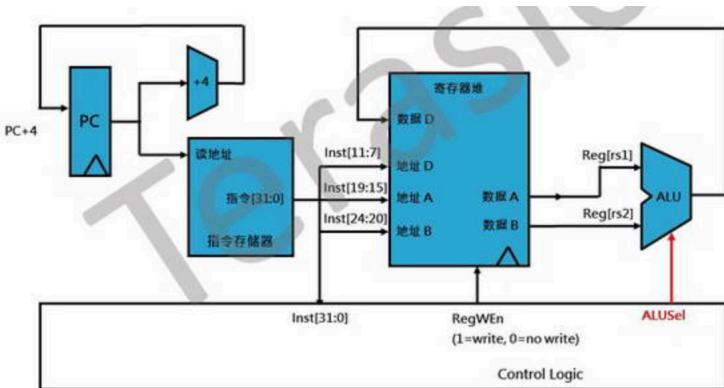
1. R 型指令：

(a) 简介：

R 型指令包含简单的运算指令，由两个 32 位源寄存器 (rs1, rs2) 进行运算，得到的结果存入一个 32 位目的寄存器 (rd)。R 型指令的 6 位操作码 (opcode) 是固定的，由功能码 (func7 和 func3) 确定是何种运算。



(b) 数据通路 & 指令执行流程：



- i. 首先按照 PC 中的地址，从指令存储器中读出 32 位指令，然后 $PC+4$ ；
- ii. 指令被送入控制器，控制器根据功能码和操作码确定这是 R 型指令中的某一条指令，并产生控制信号 **ALUSel** 和 **RegWEn**；

- iii. 将指令的 rs1、rs2、rd 字段分别送到寄存器堆的相应地址端，寄存器根据 rs1 和 rs2 字段的地址读出对应的寄存器的值 Reg[rs1] 和 Reg[rs2];
- iv. ALU 根据控制信号 ALUSel，对 Reg[rs1] 和 Reg[rs2] 进行相应的运算，得到运算结果；
- v. 将运算结果送到寄存器堆的数据 D 端口，控制器产生的 RegWEn 信号使寄存器堆允许写入，寄存器堆根据地址 D 端口的地址将数据 D 端口的值写入相应的寄存器。

(c) 指令格式：

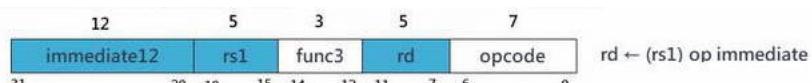
func7	rs2	rs1	func3	rd	opcode	指令
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

- 算数运算：加法 add、减法 sub
- 逻辑运算：与 and、或 or、异或 xor
- 比较运算：有符号小于比较 slt、无符号小于比较 sltu
- 移位运算：逻辑左移 sll、逻辑右移 srl、算数右移 sra

2. I 型指令：

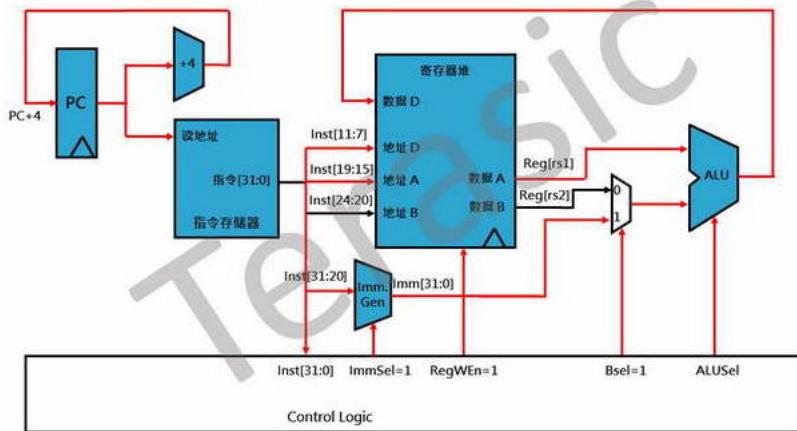
(a) 简介：

I 型指令包含简单的立即数运算指令和 load 指令。立即数运算指令是由一个 32 位源寄存器 (rs1) 和指令中包含的立即数（经扩充后的）进行运算，将得到的结果存入一个 32 位目的寄存器 (rd)。I 型指令中的立即数运算指令的 6 位操作码 (opcode) 固定，由功能码 (func7 和 func3) 确定是何种运算。



(b) 数据通路 & 指令执行流程：

- i. 首先按照 PC 中的地址，从指令存储器中读出 32 位指令，然后 PC+4;
- ii. 指令被送入控制器，控制器根据功能码和操作码确定这是 I 型指令中的某一条指令，并产生控制信号 ALUSel、ImmSel 和 RegWEn;
- iii. 将指令的 rs1、rs2、rd 字段分别送到寄存器堆的相应地址端，寄存器根据 rs1 和 rs2 字段的地址读出对应的寄存器的值 Reg[rs1] 和 Reg[rs2]（但此时 Reg[rs2] 没有用）；
- iv. 将指令的 12 位立即数字段送到立即数扩展电路，通过 ImmSel 信号的指示，将立即数扩展为 32 位，并送到多路选择器的一个端口；
- v. Bsel 信号，控制多路选择器选择立即数的那个端口，将立即数传送到 ALU 的一个输入端；
- vi. ALU 根据控制信号 ALUSel，对 Reg[rs1] 和扩展后的立即数进行相应运算，得到运算结果；



vii. 将运算结果送到寄存器堆的数据 D 端口，寄存器堆根据地址 D 端口的值将运算结果写入相应的寄存器。

(c) 指令格式：

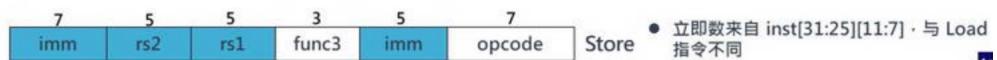
imm[11:0]	rs1	funct3	rd	opcode	指令
imm[11:0]	rs1	000	rd	0010011	ADDI
imm[11:0]	rs1	010	rd	0010011	SLTI
imm[11:0]	rs1	011	rd	0010011	SLTIU
imm[11:0]	rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	0010011	SLLI
0000000	shamt	rs1	101	0010011	SRLI
0100000	shamt	rs1	101	0010011	SRAI
imm[11:0]	rs1	000	rd	0000011	LB
imm[11:0]	rs1	001	rd	0000011	LH
imm[11:0]	rs1	101	rd	0000011	LHU

- 算数运算：加法 addi
- 逻辑运算：与 andi、或 ori、异或 xor
- 比较运算：有符号小于比较 slti、无符号小于比较 sltu
- 移位运算：逻辑左移 slli、逻辑右移 srli、算数右移 srai

3. S 型指令：

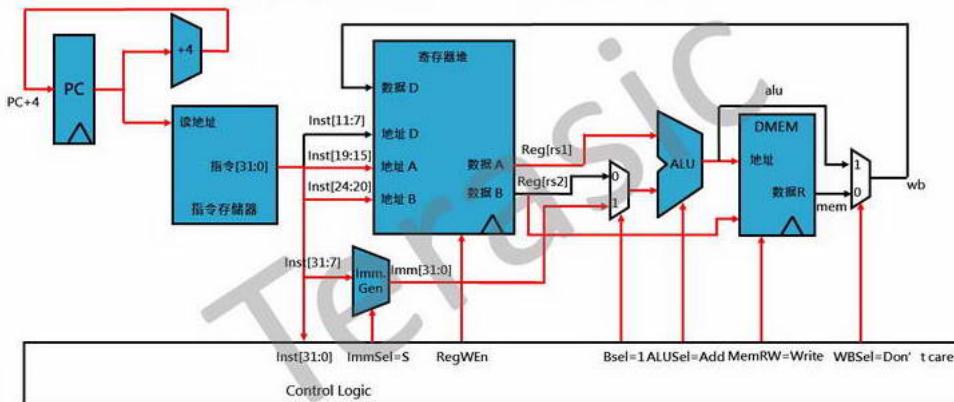
(a) 简介：

S - type 型指令包括 Store 指令，其功能是完成存储器的写操作。Store 使用 funct3 字段选择 Word, HalfWord, Byte。Store 指令将 rs1 中的数据与 12 位立即数字段符号扩展相加所得到的结果作为访问存储器的地址，即存储器地址 = Reg[rs1] + imm。接着，将 rs2 中的数据写入存储器相应地址中。



(b) 数据通路 & 指令执行流程:

- 首先按照 PC 中的地址，从指令存储器中读出 32 位指令，然后 $PC+4$ ；



- ii. 指令中的控制码和功能码送到控制器中产生相应的控制信号;
- iii. 将 rs1 和 rs2 的地址送入寄存器堆, 得到 rs1 寄存器的值 Reg[rs1], 作为 ALU 的第一个输入端; rs2 寄存器的值 Reg[rs2], 作为 DMEM 写入地址的数据;
- iv. 将 12 位的立即数进行符号扩展, 得到 32 位的立即数作为 ALU 的第二个输入端。此处控制信号 Bsel=1, 表示指令的立即数部分有效, 二选一选择器输出立即数的内容;
- v. 控制信号 ALUSel = Add, 指示 ALU 做加法运算, 得到的地址送入数据存储器 DMEM 的地址接口, 并注意此时控制信号 MemRW = Write, 使得数据存储器处于写状态;
- vi. DMEM 接收到地址和数据后, 将所选地址空间写入 Reg[rs2]。

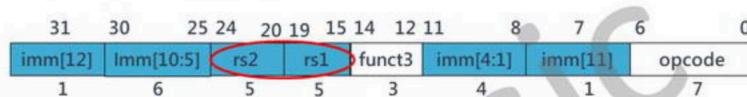
(c) 指令格式:

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	指令
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

4. B 型指令:

(a) 简介:

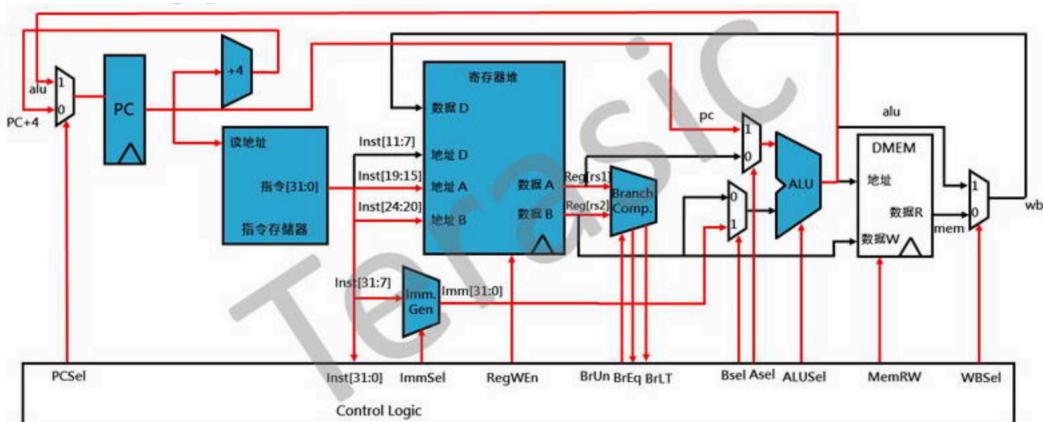
B 型指令的功能是比较寄存器 rs1、rs2 中的值, 并根据比较结果进行分支跳转。funct3 字段表示分支跳转的类型。B 型指令中 beq/bne 需进行相等比较的计算, blt/bltu 与 bge/bgeu 需进行量值比较的计算。



- B-type 指令与 S-type 指令类似, 需要两个源寄存器 rs1、rs2, 和一个 12-bit 的立即数
- 12-bit 的立即数实际上是 13-bit 的有符号数, 最低位为 0, 所以没有进行存储
- 比较操作数 rs1、rs2
- 计算目标地址
 - ✓ 首先进行符号位的扩展
 - ✓ 然后左移 1 位
 - ✓ 将得到的立即数与 PC 值相加

(b) 数据通路 & 指令执行流程:

- i. 首先按照 PC 中的地址, 从指令存储器中读出 32 位指令, 然后 PC+4;
- ii. 指令中的控制码和功能码送到控制器中产生相应的控制信号;



- iii. 将指令中的 rs1 和 rs2 的地址送入寄存器堆，得到对应的寄存器的值 Reg[rs1] 和 Reg[rs2]；
- iv. Reg[rs1] 和 Reg[rs2] 被送入比较运算单元，比较后的结果送至控制器用于判断是否进行目标地址的跳转；
- v. 将 PC 的值送至 ALU 的第一个输入端；
- vi. 将指令中的立即数字段送入立即数扩展电路，该电路输出一个 32 位的数，送至 ALU 的第二个输入端；
- vii. 若进行跳转，则由 ALU 计算出目标地址送至 PC 处。

(c) 指令格式：

imm[12 10:5]	rs2	rs1	funct3	imm[4:1 11]	opcode	指令
imm[12 10:5]	rs2	rs1	000	imm[4:1 11]	1100011	BEQ
imm[12 10:5]	rs2	rs1	001	imm[4:1 11]	1100011	BNE
imm[12 10:5]	rs2	rs1	100	imm[4:1 11]	1100011	BLT
imm[12 10:5]	rs2	rs1	101	imm[4:1 11]	1100011	BGE
imm[12 10:5]	rs2	rs1	110	imm[4:1 11]	1100011	BLTU
imm[12 10:5]	rs2	rs1	111	imm[4:1 11]	1100011	BGEU

5. U 型指令：

(a) 简介：

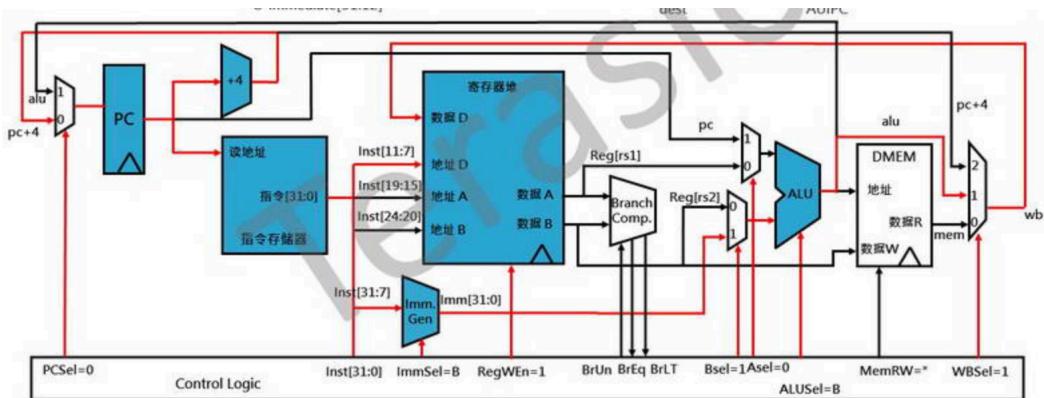
U 型指令包含 LUI 和 AUIPC，用于构建 32 位常数，由 20 位立即数，rd 和操作码构成。



(b) 数据通路 & 指令执行流程：

- i. 首先按照 PC 中的地址，从指令存储器中读出 32 位指令，然后 PC+4；
- ii. 指令中的控制码和功能码送到控制器中产生相应的控制信号；
- iii. Bsel = 1 使得 ALU B 端口输入扩展后的立即数，Asel = 1 使得 ALU A 端口输入 PC；
- iv. 将 20 位的立即数进行扩展（低 12 位填 0），输入进 ALU B 端口；
- v. 若是 LUI 指令，ALUsel = B(输出等于 B 端口输入)；若是 AUIPC 指令，ALUsel = Add(将扩展后的立即数与 PC 相加)；
- vi. WBSel = 1 使得 ALU 运算结果直接存入寄存器堆中 rd 所指寄存器；

(c) 指令格式：

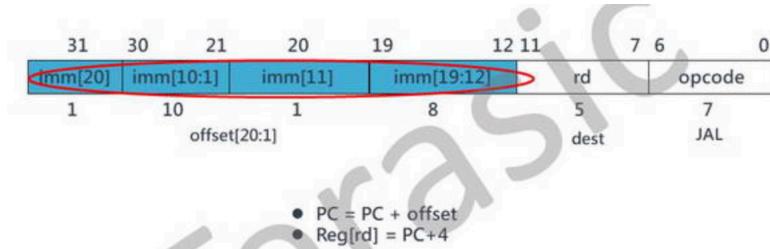


imm[32: 12]	rd	opcode	指令
imm[32: 12]	rd	0110111	LUI
imm[32: 12]	rd	0010111	AUIPC

6. J 型指令:

(a) 简介:

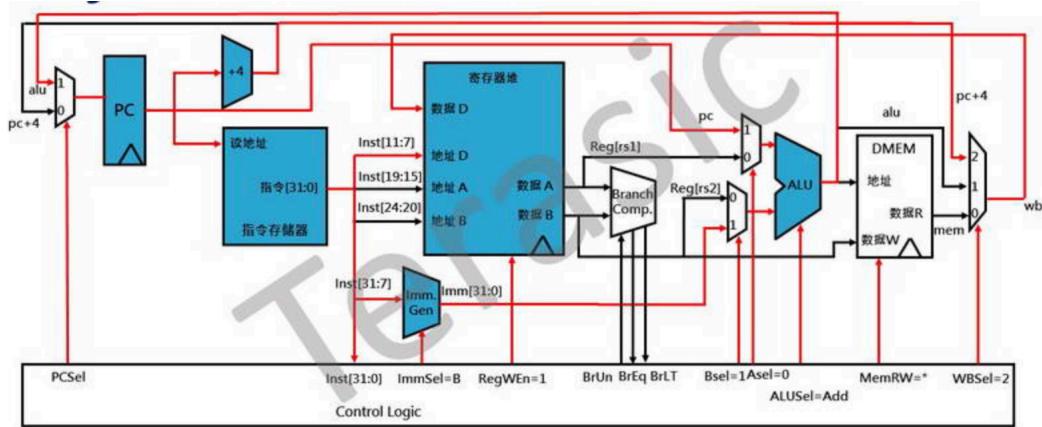
JAL/JALR 的功能是无条件跳转到目标地址。其中 JAL 指令中目标地址为 PC 的值加上 20 位的立即数所表示的偏移量，JALR 指令中目标地址为寄存器 rs1 的值加上 12 位的立即数所表示的偏移量。JAL/JALR 所需的计算类型只有加法运算。



(b) 数据通路 & 指令执行流程:

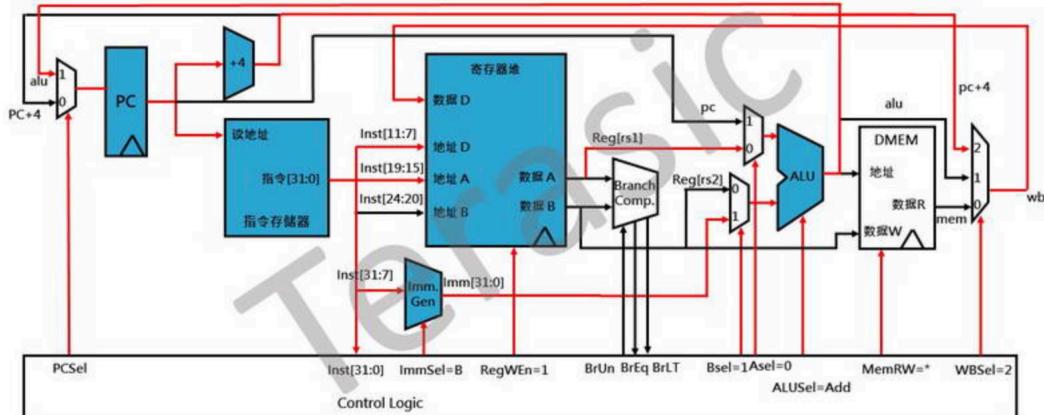
JAL

- 首先按照 PC 中的地址，从指令存储器中读出 32 位指令，然后 PC+4；
- 指令中的控制码和功能码送到控制器中产生相应的控制信号；
- 将 PC 的值送至 ALU 的第一个输入端；
- 将指令中的立即数字段送入立即数扩展电路，该电路输出一个 32 位的数，送至 ALU 的第二个输入端；
- 控制信号 ALUSel = Add，指示 ALU 做加法运算，得到的结果即为目标地址，将其送至 PC；



vi. 将原来 PC+4 的值写回到寄存器 rd 中；

JALR



- i. 首先按照 PC 中的地址，从指令存储器中读出 32 位指令，然后 PC+4；
- ii. 指令中的控制码和功能码送到控制器中产生相应的控制信号；
- iii. 将指令中的 rs1,rd 的地址送入寄存器堆，得到 rs1 对应的寄存器的值 Reg[rs1]，并送入 ALU 的第一个端口；
- iv. 将指令中的立即数字段送入立即数扩展电路，该电路输出一个 32 位的数，送至 ALU 的第二个输入端；
- v. 控制信号 ALUSel = Add，指示 ALU 做加法运算，得到的结果即为目标地址，将其送至 PC；
- vi. 将原来 PC+4 的值写回到寄存器 rd 中；

(c) 指令格式：

Imm[20][10:1][11][19:12]	rd	opcode	指令
Imm[20][10:1][11][19:12]	rd	110111	JAL

imm[11:0]	rs1	funct3	rd	opcode	指令
imm[11:0]	rs1	000	rd	1100111	JALR

1.2 蜂鸟 E203 简介

E203

- 蜂鸟 E203 系列处理器由作者所在的公司开发，是一款开源的 RISC-V 处理器。蜂鸟是世界上最小的鸟类，其体积虽小，却有着极高的速度与敏锐度，可以说是“能效比”最高的鸟类。E203 系列以蜂鸟命名便寓意于此，旨在将其打造成为一款世界上最高能效比的 RISC 处理器。



E203 核心数据通路的模块划分

- IFU 取址单元
- EXU 执行单元
- LSU 访存单元
- BIU 总线

E203 Core 的代码架构

```

e200_opensource
|----rtl           // 存放 RTL 的目录
|   |----e203      // E203 核和 SoC 的 RTL 目录
|   |----general   // 存放一些公用的通用 RTL 代码
|   |----core       // 存放 e203 Core 的 RTL 代码,
|   |                // 列举主要文件如下, 全部的文件列表请参见 GitHub
|   |----config.v   // 参数配置文件, 参见第 4.6 节了解具体配置信息
|   |----e203_biu.v // BIU 模块
|   |----e203_reset_ctrl.v // Core 的复位控制 (Reset Control) 模块
|   |----e203_clk_ctrl.v // Core 的时钟控制 (Clock Control) 模块
|   |----e203_cpu_top.v // Core 的顶层模块
|   |----e203_cpu.v   // Core 去除了 SRAM 之后的逻辑顶层模块
|   |----e203_core.v  // Core 的主体逻辑模块
|   |----e203_dtcn_ctrl.v // DTCM 的控制模块
|   |----e203_itcm_ctrl.v // ITCM 的控制模块
|   |----e203_exu.v   // Core 内部执行单元顶层模块
|   |----e203_ifu.v   // Core 内部取指令单元顶层模块
|   |----e203_lsu.v   // Core 内部存储器访问单元顶层模块
|   |----e203_srams.v // Core 的所有 SRAM 的顶层模块
|   |----e203_itcm_ram.v // ITCM 的 SRAM 模块
|   |----e203_dtcn_ram.v // DTCM 的 SRAM 模块

```

E203 数据通路的两级流水水线

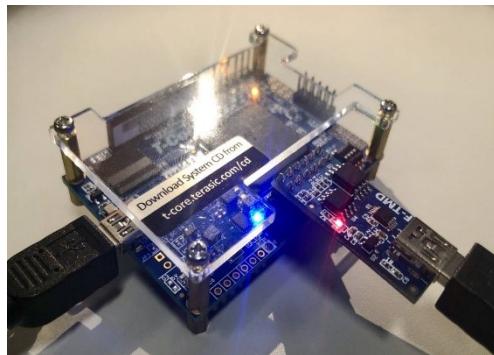
1. 第一级是 IFU，包括，取址、分支预测、生成 PC。
2. 第二级是译码、派遣、执行、访存、写回。

E203 的特点

1. 蜂鸟 E203 处理器研发团队拥有在国际一流公司多年开发处理器的经验，使用稳健的。
2. 蜂鸟 E203 的代码为人工编写，添加丰富的注释且可读性强，非常易于理解。
3. 蜂鸟 E203 专为 IoT 领域量身定做，其具有 2 级流水线深度，功耗和性能指标均优于目前主流商用的 ARM Cortex-M 系列处理器，且免费开源，能够在 IoT 领域完美替代 ARM Cortex-M 处理器。

1.3 T-core 开发板介绍

1. T-core 开发板是友晶科技公司的基于 RISC-V 的新款开发板。T-Core 提供了围绕 Intel MAX 10 FPGA 构建的强大的硬件设计平台。它配备完善，可在控制平面或数据路径应用中提供具有成本效益的单芯片解决方案，并提供行业领先的可编程逻辑，以实现最终的设计灵活性。
2. 借助 MAX 10 FPGA，可以获得比上一代更低的功耗/成本和更高的性能。可支持大量应用，包括协议桥接，电机控制驱动，模数转换和手持设备。T-Core 开发板包括硬件，例如板载 USB-Blaster II, QSPI Flash, ADC 接头连接器，WS2812B RGB LED 和 2x6 TMD 扩展接头连接器。通过利用所有这些功能，T-Core 是展示，评估和原型化 Intel MAX 10 FPGA 真正潜力的理想解决方案。T-Core 还通过板载 JTAG 调试支持 RISC-V CPU。它是学习 RISC-V CPU 设计或嵌入式系统设计的理想平台。



2. 实践篇

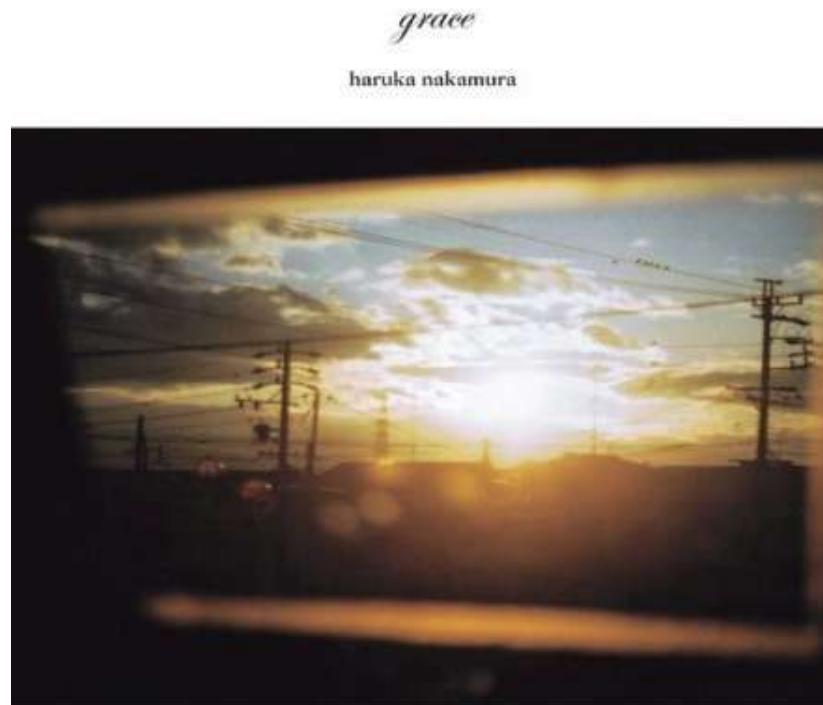


图 1: DHCP Request 包字节形式

1.

•

```
1 class Pokemon : public QObject //代码
```

字段	值	含义
包头长度	45	数据分组首部长度 20 字节
服务类型	00	正常时延、正常吞吐量、正常可靠性
总长度	003c	数据分组长度 60 字节
标识	5c6e	标识为 23662
标志	00	MF=0: 此片为最后一片, DF=0: 允许分片
片偏移	00	偏移量 =0
TTL	40	每跳生存周期为 64
协议	01	来自 ICMP 协议
头部校验和	0000	IP 头部校验和为 0000
源地址	c0a86386	源地址为 192.168.99.134
目的地址	72ff28a6	目的地址为 114.255.40.166

3. 结果展示

4. 未来展望