chapter2 汇编语言

汇编语言的表示

万分感谢 tonycrane 和 hobbitqia 学长的网页

本文大量参考了 鹤翔万里 RISC-V 非特权级 ISA

寄存器

寄存器	ABI 名称	用途描述	修改前是否要保存
x0	zero	硬件 0	不得修改
x1	ra	返回地址（return address）	需要
x2	sp	栈指针（stack pointer）	需要
x3	gp	全局指针（global pointer）	需要
x4	tp	线程指针（thread pointer）	需要
x5	t0	临时变量 / 备用链接寄存器（alternate link reg）	不需要
x6-7	t1-2	临时变量	不需要
x8	s0/fp	需要保存的寄存器 / 帧指针（frame pointer）	需要
x9	s1	需要保存的寄存器	需要
x10-11	a0-1	函数参数 / 返回值	不需要
x12-17	a2-7	函数参数	不需要
x18-27	s2-11	需要保存的寄存器	需要
x28-31	t3-6	临时变量	不需要

指令与机器码互换

R 型指令

31		25	24		20	19		15	14		12	11		7	6		0
funct7			rs2			rs1			funct3			rd			opcode		

使用寄存器进行数字逻辑运算的指令格式，运算由 opcode funct3 funct7 决定，rd = rs1 op rs2（shift 类例外，它们用 rs2 位置表示移位数的立即数）

I 型指令

31		20	19		15	14		12	11		7	6		0
imm[11:0]			rs1			funct3			rd			opcode		

使用寄存器和立即数进行数字逻辑运算，以及 load 类指令等的指令格式，运算类型等由 opcode funct3 决定，如果是 ALU 运算，则 rd = rs1 op imm

立即数是 {{20{inst[31]}}, inst[31:20]}，也就是对 imm[11:0] 进行符号位扩展到 32 位

S 型指令

31		25	24		20	19		15	14		12	11		7	6		0
imm[11:5]			rs2			rs1			funct3			imm[4:0]			opcode		

store 类指令，store 的大小由 funct3 决定，以变址模式进行寻址，即 rs1 = [rs2+imm]

立即数是 {{20{inst[31]}}, inst[31:25], inst[11:7]}

B 型指令

31		25	24		20	19		15	14		12	11		7	6		0
imm[12,10:5]			rs2			rs1			funct3			imm[4:1,11]			opcode		

由 S 型指令分来，与之区别是立即数读取顺序不同，是所有分支类指令。是否分支由 funct3 rs1 rs2 决定

立即数是 {{19{inst[31]}}, inst[31], inst[7], inst[30:25], inst[11:8], 1'b0}

U 型指令

31		12	11		7	6		0
imm[31:12]			rd			opcode		

LUI 和 AUIPC，立即数都是在高 20 位，而且没有源操作数

立即数是 {inst[31:12], 12'b0}

J 型指令

31		12	11		7	6		0
imm[20,10:1,11,19:12]			rd			opcode		

由 U 型指令分来，区别也是立即数读取不同，仅有 JAL 一个指令

立即数是 {{11{inst[31]}}, inst[31], inst[19:12], inst[20], inst[30:21], 1'b0}

RV32I 指令

加减法指令

add (R-type)

31		25	24		20	19		15	14		12	11		7	6		0
0000000			rs2			rs1			000			rd			0110011		

- **指令格式**：add rd, rs1, rs2
- **指令作用**：rd = rs1 + rs2
- **注意**：溢出会被忽略

sub(R-type)

31		25	24		20	19		15	14		12	11		7	6		0
0100000			rs2			rs1			000			rd			0110011		

- **指令格式**：sub rd, rs1, rs2
- **指令作用**：rd = rs1 - rs2
- **注意**：溢出会被忽略

addi(R-type)

31		20	19		15	14		12	11		7	6		0
imm[11:0]			rs1			000			rd			0010011		

- **指令格式**：addi rd, rs1, imm
- **指令作用**：rd = rs1 + imm
- **注意**：溢出会被忽略，imm 在 [-2048, 2047] 范围内

比较运算指令

slt(R-type)

31		25	24		20	19		15	14		12	11		7	6		0
0000000			rs2			rs1			010			rd			0110011		

- **指令格式**：slt rd, rs1, rs2
- **指令作用**：（set less than）如果 rs1 < rs2 则 rd = 1，否则 rd = 0
- **注意**：rs1 rs2 会被视为有符号数进行比较

sltu(R-type)

31		25	24		20	19		15	14		12	11		7	6		0
0000000			rs2			rs1			011			rd			0110011		

- **指令格式**：sltu rd, rs1, rs2
- **指令作用**：（set less than unsigned）如果 rs1 < rs2 则 rd = 1，否则 rd = 0
- **注意**：rs1 rs2 会被视为无符号数进行比较

slti(I-type)

31		20	19		15	14		12	11		7	6		0
imm[11:0]			rs1			010			rd			0010011		

- **指令格式**：slti rd, rs1, imm
- **指令作用**：（set less than immediate）如果 rs1 < imm 则 rd = 1，否则 rd = 0
- **注意**：imm 在 [-2048, 2047] 范围内，被视为有符号数进行比较

sltiu(I-type)

31		20	19		15	14		12	11		7	6		0
imm[11:0]			rs1			011			rd			0010011		

- **指令格式**：sltiu rd, rs1, imm
- **指令作用**：（set less than immediate unsigned）如果 rs1 < imm 则 rd = 1，否则 rd = 0
- **注意**：imm 在 [-2048, 2047] 范围内，rs1 imm 被视为无符号数进行比较

二进制位运算指令

and(R-type)

31		25	24		20	19		15	14		12	11		7	6		0
0000000			rs2			rs1			111			rd			0110011		

- **指令格式**：and rd, rs1, rs2
- **指令作用**：rd = rs1 & rs2 按位与

or(R-type)

31		25	24		20	19		15	14		12	11		7	6		0
0000000			rs2			rs1			110			rd			0110011		

- **指令格式**：or rd, rs1, rs2
- **指令作用**：rd = rs1 \| rs2 按位或

xor(R-type)

31		25	24		20	19		15	14		12	11		7	6		0
0000000			rs2			rs1			100			rd			0110011		

- **指令格式**：xor rd, rs1, rs2
- **指令作用**：rd = rs1 ^ rs2 按位异或

andi(I-type)

31		20	19		15	14		12	11		7	6		0
imm[11:0]			rs1			111			rd			0010011		

- **指令格式**：andi rd, rs1, imm
- **指令作用**：rd = rs1 & imm 按位与

- **注意**：imm 在 [-2048, 2047] 范围内，会扩展符号位

ori I 型

31		20	19		15	14		12	11		7	6		0
imm[11:0]			rs1			110			rd			0010011		

- **指令格式**：ori rd, rs1, imm
- **指令作用**：rd = rs1 | imm 按位或
- **注意**：imm 在 [-2048, 2047] 范围内，会扩展符号位

xori I 型

31		20	19		15	14		12	11		7	6		0
imm[11:0]			rs1			100			rd			0010011		

- **指令格式**：xori rd, rs1, imm
- **指令作用**：rd = rs1 ^ imm 按位异或
- **注意**：imm 在 [-2048, 2047] 范围内，会扩展符号位（xori rd, rs1, -1 相当于 rd = ~rs1）

移位运算指令

sll r 型

31		25	24		20	19		15	14		12	11		7	6		0
0000000			rs2			rs1			001			rd			0110011		

- **指令格式**：sll rd, rs1, rs2
- **指令作用**：rd = rs1 << rs2[4:0] 左移（左侧丢掉，右侧补 0）
- **注意**：会取 rs2 内数值的低 5 位进行运算

srl r 型

31		25	24		20	19		15	14		12	11		7	6		0
0000000			rs2			rs1			101			rd			0110011		

- **指令格式**：srl rd, rs1, rs2
- **指令作用**：rd = rs1 >> rs2[4:0] 逻辑右移（左侧补 0，右侧丢掉）
- **注意**：会取 rs2 内容的低 5 位

sra r 型

31		25	24		20	19		15	14		12	11		7	6		0
0100000			rs2			rs1			101			rd			0110011		

- **指令格式**：sra rd, rs1, rs2

- **指令作用**：rd = rs1 >>> rs2[4:0] 算数右移（左侧补符号位，右侧丢掉）
- **注意**：会取 rs2 内容的低 5 位进行运算

slli　i 型（改）

31		25	24		20	19		15	14		12	11		7	6		0
0000000			shamt			rs1			001			rd			0010011		

- **指令格式**：slli rd, rs1, shamt
- **指令作用**：rd = rs1 << shamt 左移（左侧丢掉，右侧补 0）
- **注意**：shamt（shift amount）会编码到原来 rs2 的位置，它是一个立即数，正好有 5 位

srli　i 型（改）

31		25	24		20	19		15	14		12	11		7	6		0
0000000			shamt			rs1			101			rd			0010011		

- **指令格式**：srli rd, rs1, shamt
- **指令作用**：rd = rs1 >> shamt 逻辑右移（左侧补 0，右侧丢掉）
- **注意**：shamt（shift amount）会编码到原来 rs2 的位置，它是一个立即数，正好有 5 位

srai　i 型（改）

31		25	24		20	19		15	14		12	11		7	6		0
0100000			shamt			rs1			101			rd			0010011		

- **指令格式**：srai rd, rs1, shamt
- **指令作用**：rd = rs1 >>> shamt 算数右移（左侧补符号位，右侧丢掉）
- **注意**：shamt（shift amount）会编码到原来 rs2 的位置，它是一个立即数，正好有 5 位

数据加载指令

lui　U 型

31		12	11		7	6		0
imm[31:12]			rd			0110111		

- **指令格式**：lui imm
- **指令作用**：（load upper immediate）rd = imm << 12 将 imm 加载到 rd 的高 20 位
- **注意**：imm 不能超过 20 位，rd 以十六进制表示就是 imm 后接三个 0

auipc　U 型

31		12	11		7	6		0
imm[31:12]			rd			0010111		

- **指令格式**：auipc rd
- **指令作用**：（add upper immediate with pc）rd = pc + imm << 12 将 imm 加载到高 20 位，然后加上 pc 值
- **注意**：常用来构建 pc 相对寻址的地址，imm 不能超过 20 位

jump 类无条件跳转指令

jal　　J 型

31		12	11		7	6		0
imm[20,10:1,11,19:12]			rd			1101111		

- **指令格式**：jal rd, imm
- **指令作用**：（jump and link）rd = pc+4, pc = pc+imm 即将当前指令下一条指令的地址存入 rd，然后相对跳转到 imm 处
- **注意**：imm 在汇编程序中一般用标号来指定，jal 可以跳到 ±1MiB 范围内的代码

jalr　　I 型

31		20	19		15	14		12	11		7	6		0
imm[11:0]			rs1			000			rd			1100111		

- **指令格式**：jalr rd, imm(rs1)
- **指令作用**：rd = pc+4, pc = (imm+rs1) & 0xFFFFFFFE 即最低位会被设为 0
- **注意**：可以实现任意位置跳转

branch 类条件跳转指令

beq　　B 型

31		25	24		20	19		15	14		12	11		7	6		0
imm[12,10:5]			rs2			rs1			000			imm[4:1,11]			1100011		

- **指令格式**：beq rs1, rs2, imm
- **指令作用**：（branch if equal）如果 rs1 == rs2，则 pc = pc+imm
- **注意**：可以跳转到 ±4KiB 范围内

bne　　B 型

31		25	24		20	19		15	14		12	11		7	6		0
imm[12,10:5]			rs2			rs1			001			imm[4:1,11]			1100011		

- **指令格式**：bne rs1, rs2, imm
- **指令作用**：（branch if not equal）如果 rs1 != rs2，则 pc = pc+imm
- **注意**：可以跳转到 ±4KiB 范围内

blt B 型

31		25	24		20	19		15	14		12	11		7	6		0
imm[12,10:5]			rs2			rs1			100			imm[4:1,11]			1100011		

- **指令格式**：blt rs1, rs2, imm
- **指令作用**：（branch if less than）如果 rs1 < rs2 则 pc = pc+imm
- **注意**：rs1 rs2 视为有符号数进行比较，可以跳转到 ±4KiB 范围内

bge B 型

31		25	24		20	19		15	14		12	11		7	6		0
imm[12,10:5]			rs2			rs1			101			imm[4:1,11]			1100011		

- **指令格式**：bge rs1, rs2, imm
- **指令作用**：（branch if greater than or equal）如果 rs1 >= rs2 则 pc = pc+imm
- **注意**：rs1 rs2 视为有符号数进行比较，可以跳转到 ±4KiB 范围内

bltu B 型

31		25	24		20	19		15	14		12	11		7	6		0
imm[12,10:5]			rs2			rs1			110			imm[4:1,11]			1100011		

- **指令格式**：bltu rs1, rs2, imm
- **指令作用**：（blt unsigned）如果 rs1 < rs2 则 pc = pc+imm
- **注意**：rs1 rs2 视为无符号数进行比较，可以跳转到 ±4KiB 范围内

bgeu B 型

31		25	24		20	19		15	14		12	11		7	6		0
imm[12,10:5]			rs2			rs1			111			imm[4:1,11]			1100011		

- **指令格式**：bgeu rs1, rs2, imm
- **指令作用**：（bge unsigned）如果 rs1 >= rs2 则 pc = pc+imm
- **注意**：rs1 rs2 视为无符号数进行比较，可以跳转到 ±4KiB 范围内

装载存储指令

load 类装载指令

lb I 型

31		20	19		15	14		12	11		7	6		0
imm[11:0]			rs1			000			rd			0000011		

- **指令格式**：lb rd, imm(rs1)
- **指令作用**：从 rs1 + imm 处内存读取一个字节到 rd 低八位，再进行符号扩展

lh　　　　　　　　　　I 型

31		20	19		15	14		12	11		7	6		0
imm[11:0]			rs1			001			rd			0000011		

- **指令格式**：lh rd, imm(rs1)
- **指令作用**：从 rs1 + imm 处内存读取一个 16 位数到 rd 低 16 位，然后进行符号扩展

lw　　　　　　　　　　I 型

31		20	19		15	14		12	11		7	6		0
imm[11:0]			rs1			010			rd			0000011		

- **指令格式**：lw rd, imm(rs1)
- **指令作用**：从 rs1 + imm 处内存读取一个 32 位数到 rd 中

lbu　　　　　　　　　　I 型

31		20	19		15	14		12	11		7	6		0
imm[11:0]			rs1			100			rd			0000011		

- **指令格式**：lbu rd, imm(rs1)
- **指令作用**：从 rs1 + imm 处内存读取一个字节放到 rd 低 8 位，然后进行零扩展（高位全补 0）

lhu　　　　　　　　　　I 型

31		20	19		15	14		12	11		7	6		0
imm[11:0]			rs1			101			rd			0000011		

- **指令格式**：lhu rd, imm(rs1)
- **指令作用**：从 rs1 + imm 处内存读取 16 位数存入 rd，并进行零扩展（高 16 位全为 0）

store 类存储指令

sb　　　　　　　　　　S 型

31		25	24		20	19		15	14		12	11		7	6		0
imm[11:5]			rs2			rs1			000			imm[4:0]			0100011		

- **指令格式**：sb rs2, imm(rs1)
- **指令作用**：将 rs2 的低 8 位拷贝到 rs1 + imm 处内存中

sh　　S 型

31		25	24		20	19		15	14		12	11		7	6		0
imm[11:5]			rs2			rs1			001			imm[4:0]			0100011		

- **指令格式**：sh rs2, imm(rs1)
- **指令作用**：将 rs2 的低 16 位拷贝到 rs1 + imm 处内存中

sw　　S 型

31		25	24		20	19		15	14		12	11		7	6		0
imm[11:5]			rs2			rs1			010			imm[4:0]			0100011		

- **指令格式**：sw rs2, imm(rs1)
- **指令作用**：将 rs2（32 位）拷贝到 rs1 + imm 处内存中

环境调用和断点指令

ecall　　I 型

31		20	19		15	14		12	11		7	6		0
000000000000			00000			000			00000			1110011		

- **指令格式**：ecall
- **指令作用**：请求环境调用（类似 syscall），EEI 会定义相关参数规范（一般通过寄存器传参）

ebreak　　I 型

31		20	19		15	14		12	11		7	6		0
000000000001			00000			000			00000			1110011		

- **指令格式**：ebreak
- **指令作用**：将控制流转到调试环境

推荐一个小工具https://luplab.gitlab.io/rvcodecjs

寻址方式

寻址方式就是指令结果在哪里

- **立即数寻址** `addi x5, x6, 4`
- **寄存器寻址** `add x5, x6, x7`
- **基址寻址** `ld x5,100(x6)`

- **PC 相对寻址** `beq x5,x6,L1`