

第三章 RISC-V汇编及其指令系统

- RISC-V概述
- RISC-V汇编语言
- **RISC-V指令表示**
- 案例分析



第三章 RISC-V汇编及其指令系统

- **RISC-V指令表示**
 - RISC-V六种指令格式
 - RISC-V寻址模式介绍



RISC-V指令表示

- 处理的大多数数据都是字或双字（32bit或64bit）：
 - 例如：lw和sw一次访问一个字（4字节）的内存是比较常见的操作
- 指令字长、机器字长、存储字长(二进制位数)
 - 指令字长：指令中二进制代码的总位数
 - 机器字长：CPU一次能处理的二进制位数，一般等于通用寄存器位数
 - 存储字长：一个存储单元存储的二进制代码的长度
- 如何表示指令？
 - 计算机只懂1和0（汇编语言只是助记符）
 - 表示一个操作的一串二进制数，称为指令字，简称指令
 - RISC-V追求简单性，用于RV32、RV64、RV128的32位基础指令相同

RISC-V指令表示

- 指令字分成多个“字段”（一部分位）
- 每个字段有特定的含义和作用
- 理论上可以为每条指令定义不同的字段
- RISC-V定义了**六种**基本指令类型：
 - R型指令 用于**寄存器** - 寄存器操作
 - I型指令 用于短**立即数**、访存 load 操作、jalr
 - S型指令 用于访存 **store** 操作
 - B型指令 用于条件**跳转/分支/转移**操作
 - U型指令 用于**长**立即数操作
 - J型指令 用于**无条件跳转**操作

RISC-V六种基本指令类型

- **R-type**: **R**egister-register arithmetic/logical operations
- **I-type**: register-**I**mmEDIATE arith/logical operations & loads
- **S-type** (S型) : for **S**tore
- **B-type** (B型) : for **B**ranches(**SB**型?)
- **U-type** (U型) : for 20-bit **U**pper immediate instructions
- **J-type** (J型) : for **J**umps (**UJ**型?)
- 其他: Used for OS & Synchronization

指令格式

- 指令格式：用二进制代码表示指令的结构形式
 - 要求计算机处理**什么数据**？
 - 要求计算机对数据执行**什么操作**？
 - 计算机**怎样**才能**得到**要处理的**数据**？



操作码(OP)与地址码(AC)

- 操作码字段长度决定指令系统规模
 - 每条指令对应一个操作码
 - 定长操作码 n 位操作码就有 2^n 个组合种操作
 - 变长操作码 操作码可以向不用的地址码字段扩展
- 地址码（有时候也叫操作数）字段可能有多个：
 - 寻址方式字段：长度与寻址方式总数有关，可能隐含在操作码字段

注意：RV里面没有寻址方式字段

 - 地址码字段：长度与寻址方式有关（**比如寄存器号**）

RISC-V指令格式

- 会查表（基础结构（比如R型）分6个字段：7、5、5、3、5、7）

R 型	funct7	rs2	rs1	funct3	rd	opcode
I 型	imm[11:0]		rs1	funct3	rd	opcode
S 型	Imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB/ B 型	Imm[12,10:5]	rs2	rs1	funct3	imm[4:1,11]	opcode
UJ/ J 型	Imm[20,10:1,11,19:12]				rd	opcode
U 型	Imm[31:12]				rd	opcode

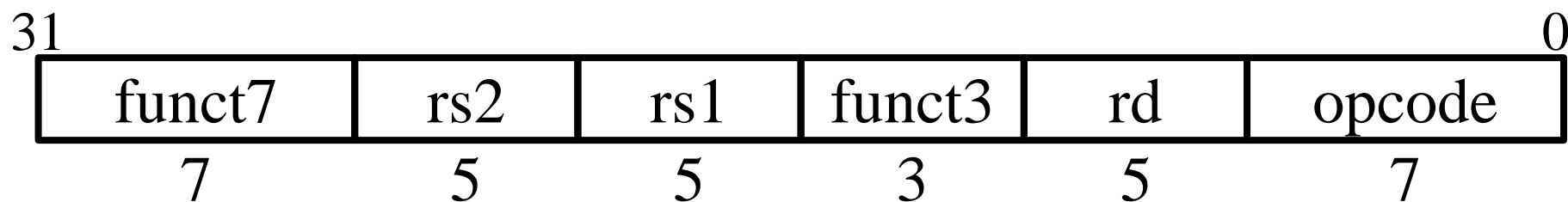
RISC-V指令类型

- **R-type**: **R**egister-register arithmetic/logical operations
- **I-type**: register-**I**mmEDIATE arith/logical operations & loads
- **S-type** (S型) : for **S**tore
- **B-type** (B型) : for **B**ranches(**SB**型?)
- **U-type** (U型) : for 20-bit **U**pper immediate instructions
- **J-type** (J型) : for **J**umps (**UJ**型?)
- 其他: Used for OS & Synchronization

指令语法（特别重要）

类型	原始语法	示例
R	op rd, rs1, rs2	add x5, x6, x7
I	op rd, rs1, imm (寄存器跳转) op rd, offset(rs1)	addi x5, x6, -10 jalr x1, 100(x5)
	(load载入类) op rd, offset(rs1)	ld x5, 40(x6)
S	(store存储类) op rs2, offset(rs1)	sd x5, 40(x6)
B	op rs1, rs2, offset	beq, x5, x6, 100
J	op rd, offset (注意: offset在实际汇编编程中用的是Label)	jal x1, 100
U	op rd, imm	lui x10, 0x87654 # x10 = 0x87654000

R型指令



- 32位指令字，分为6个不同位数的字段
- opcode(操作码): 是哪类指令
- funct7+funct3: 这两个字段结合操作码描述要执行的操作

RISC-V基础opcode对应指令类型分布图

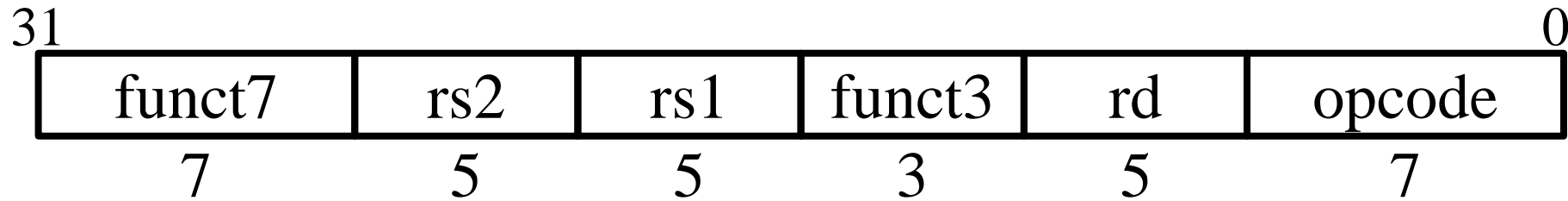
inst[4:2]	000	001	010	011	100	101	110	111
inst[6:5]								(> 32b)
00	LOAD	LOAD-FP	<i>custom-0</i>	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	48b
01	STORE	STORE-FP	<i>custom-1</i>	AMO	OP	LUI	OP-32	64b
10	MADD	MSUB	NMSUB	NMADD	OP-FP	<i>reserved</i>	<i>custom-2/rv128</i>	48b
11	BRANCH	JALR	<i>reserved</i>	JAL	SYSTEM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

Table 24.1: RISC-V base opcode map, inst[1:0]=11

Opcode共7位:

- 最低两位inst[1:0]始终为11
- 中间三位inst[4:2]根据不同指令类型取不同的值
- inst[6:5] 根据不同的操作类型会取不同的值。

R型指令



- rs1（源寄存器1）：指定第一个寄存器操作数
- rs2：指定第二个寄存器操作数
- rd（目的寄存器）：指定接收计算结果的寄存器
- 每个字段保存一个5位无符号整数（0-31），对应于一个寄存器号（x0-x31）（寄存器使用约定详见P75图2-14）

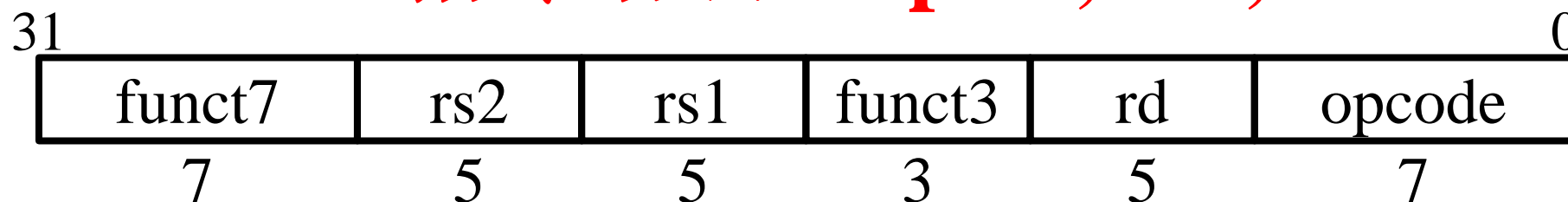
指令语法

类型	原始语法	示例
R	op rd, rs1, rs2	add x5, x6, x7
I	op rd, rs1, imm (寄存器跳转) op rd, offset(rs1) (load载入类) op rd, offset(rs1)	addi x5, x6, -10 jalr x1, 100(x5) ld x5, 40(x6)
S	(store存储类) op rs2, offset(rs1)	sd x5, 40(x6)
B	op rs1, rs2, offset	beq, x5, x6, 100
J	op rd, offset (注意: offset在实际汇编编程中用的是Label)	jal x1, 100
U	op rd, imm	lui x10, 0x87654 # x10 = 0x87654000

R型指令格式示例

- RISC-V汇编指令 add x18, x19, x10

R型指令语法: op rd, rs1, rs2



rs2=10 rs1=19

rd=18



add x8, sp, zero 对应的机器指令码为_____。

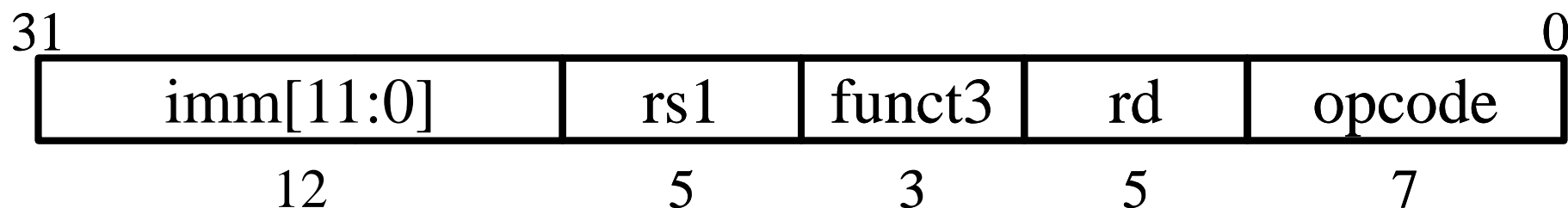
- ☐ A 0000 0000 1000 0001 0000 0000 0011 0011
- ☒ B 0000 0000 0000 0001 0000 0100 0011 0011
- ☐ C 0000 0000 1000 0001 0000 0000 0011 0011
- ☐ D 0000 0000 1000 0010 0000 0000 0011 0011

提交

R型指令

00000000	rs2	rs1	000	rd	0110011	add
01000000	rs2	rs1	000	rd	0110011	sub
00000000	rs2	rs1	001	rd	0110011	sll
00000000	rs2	rs1	010	rd	0110011	slt
00000000	rs2	rs1	011	rd	0110011	sltu
00000000	rs2	rs1	100	rd	0110011	xor
00000000	rs2	rs1	101	rd	0110011	srl
01000000	rs2	rs1	101	rd	0110011	sra
00000000	rs2	rs1	110	rd	0110011	or
00000000	rs2	rs1	111	rd	0110011	and

I型指令



- rs1、funct3、rd、opcode字段与R型相同
- rs2和funct7被**12位有符号**立即数imm[11:0]替换
- 在算术运算前，立即数总是进行符号扩展
- 立即数的表示范围（12位有符号数补码表示）
- I型指令包括3小类指令：**算数逻辑运算类、load类、jalr**

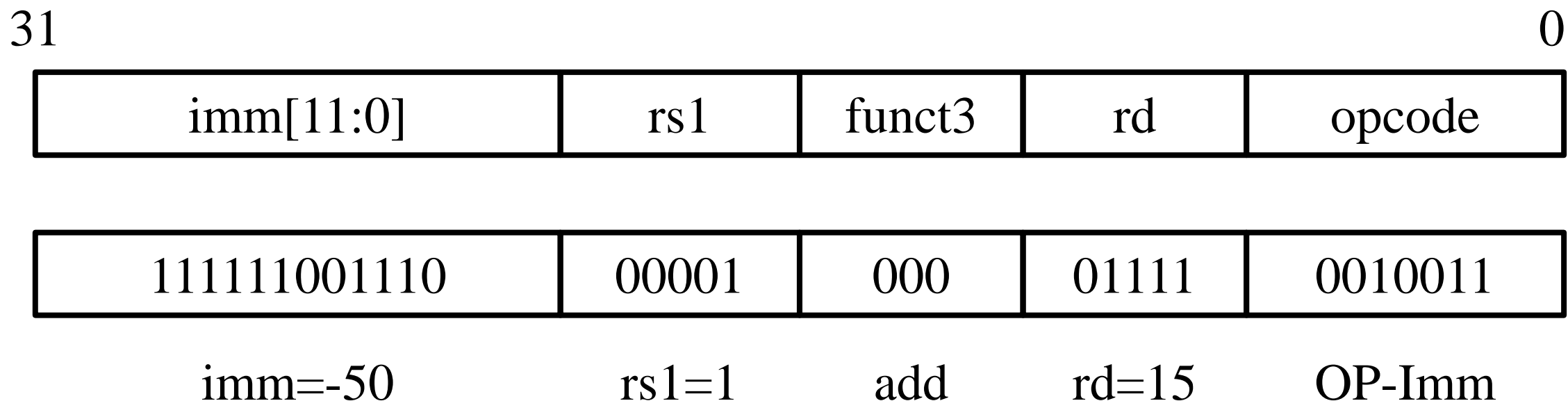
指令语法

类型	原始语法	示例
R	op rd, rs1, rs2	add x5, x6, x7
I	op rd, rs1, imm (寄存器跳转) op rd, offset(rs1) (load载入类) op rd, offset(rs1)	addi x5, x6, -10 jalr x1, 100(x5) ld x5, 40(x6)
S	(store存储类) op rs2, offset(rs1)	sd x5, 40(x6)
B	op rs1, rs2, offset	beq, x5, x6, 100
J	op rd, offset (注意: offset在实际汇编编程中用的是Label)	jal x1, 100
U	op rd, imm	lui x10, 0x87654 # x10 = 0x87654000

I型指令示例

addi x15, x1, -50

I型指令语法: op rd, rs1, imm



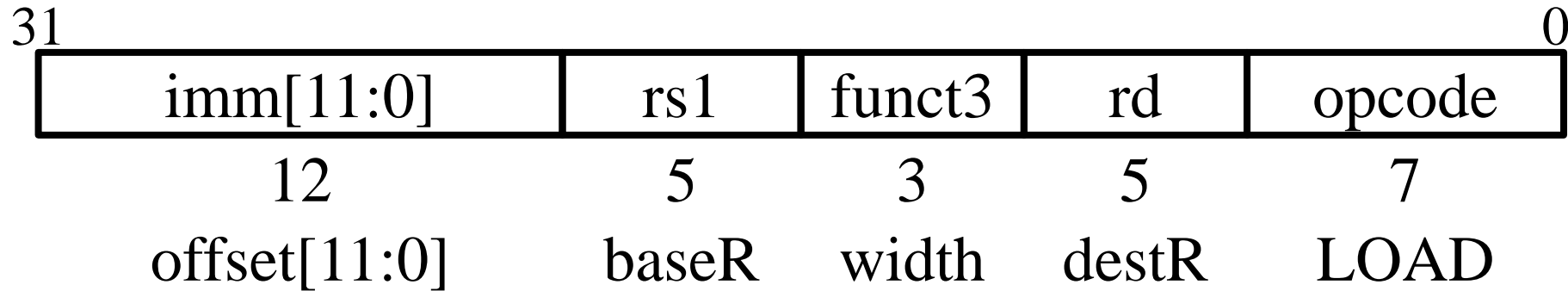
I型指令

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	100	rd	0010011	xori
imm[11:0]		rs1	110	rd	0010011	ori
imm[11:0]		rs1	111	rd	0010011	andi
000000	shamt	rs1	001	rd	0010011	slli
000000	shamt	rs1	101	rd	0010011	srli
010000	shamt	rs1	101	rd	0010011	srai

其中一个高阶立即数位用于区分
“逻辑右移”（SRLI）和“算术
右移”（SRAI）

“按立即数移位”指令仅将立
即数的低位6位用作移位量（最
多只能移位63次）

I型——Load类指令



- Load类指令也是**I型**指令
- 12位有符号立即数加寄存器rs1的基址，形成内存地址
 - 这与add immediate操作非常相似，但用于创建地址
- 将该内存地址读取到的值存储在寄存器rd中
- width表示装载的数据位数宽度和是否考虑符号

I型指令

imm[11:0]	rs1	000	rd	0000011	lb
imm[11:0]	rs1	001	rd	0000011	lh
imm[11:0]	rs1	010	rd	0000011	lw
imm[11:0]	rs1	011	rd	0000011	ld
imm[11:0]	rs1	100	rd	0000011	lbu
imm[11:0]	rs1	101	rd	0000011	lhu
imm[11:0]	rs1	110	rd	0000011	lwu

funct3字段对读取数据的大小
和“有符号性”进行编码

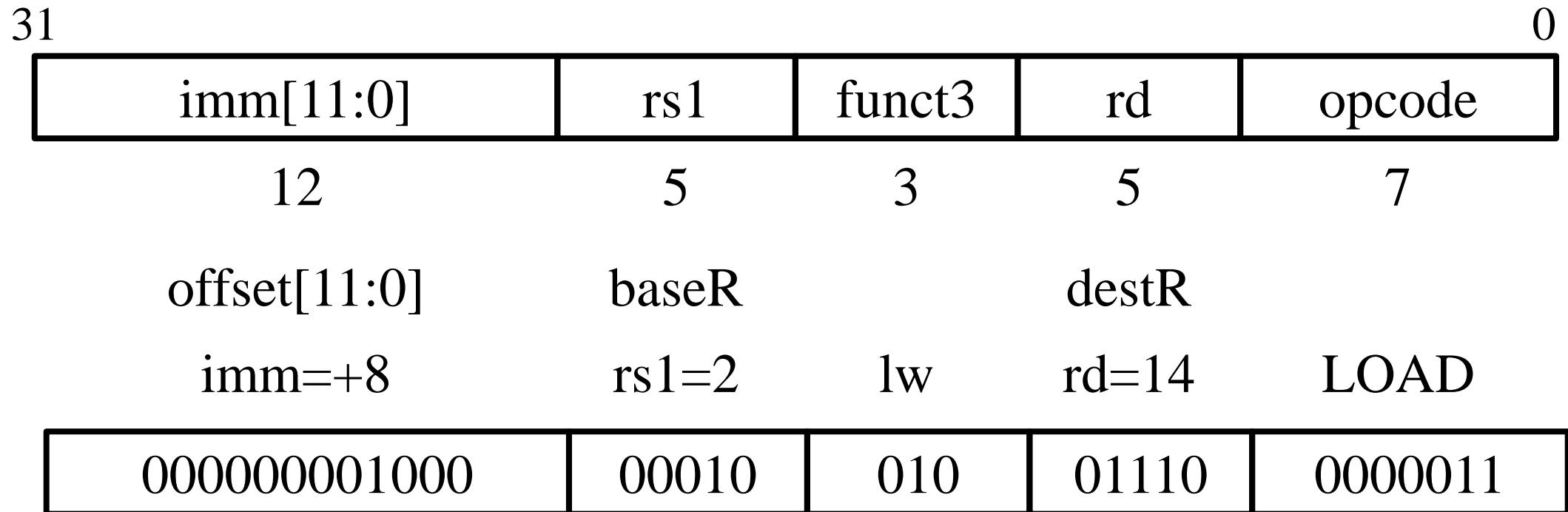
- LBU是“读取无符号字节”
- LH是“读取半字”，符号扩展以填充32/64位寄存器
- LHU是“读取无符号半字”，零扩展以填充32/64位寄存器

指令语法

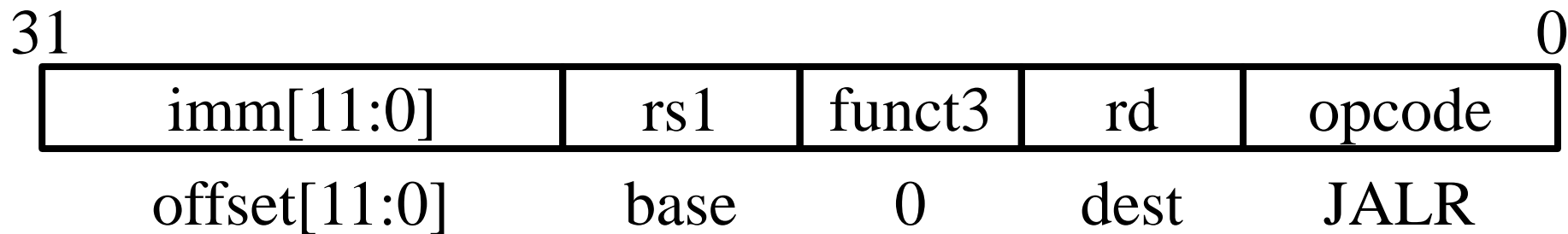
类型	原始语法	示例
R	op rd, rs1, rs2	add x5, x6, x7
I	op rd, rs1, imm (寄存器跳转) op rd, offset(rs1) (load载入类) op rd, offset(rs1)	addi x5, x6, -10 jalr x1, 100(x5) ld x5, 40(x6)
S	(store存储类) op rs2, offset(rs1)	sd x5, 40(x6)
B	op rs1, rs2, offset	beq, x5, x6, 100
J	op rd, offset (注意: offset在实际汇编编程中用的是Label)	jal x1, 100
U	op rd, imm	lui x10, 0x87654 # x10 = 0x87654000

I型Load类指令示例

lw x14, 8(x2) **I型Load类语法: op rd, offset(rs1)**



I型指令——jalr指令



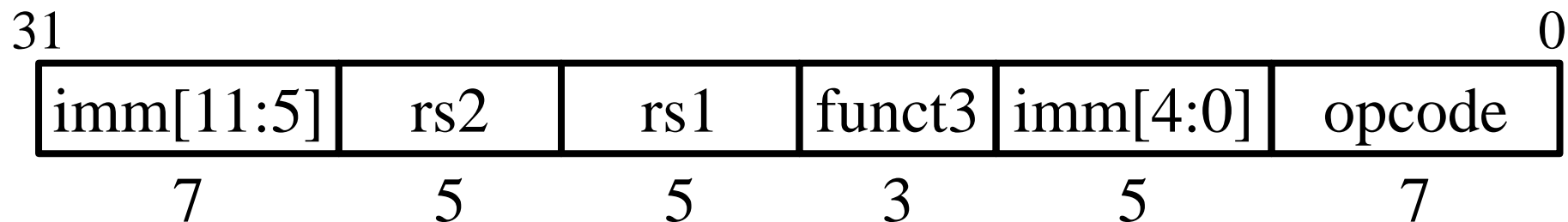
jalr rd, offset(rs1) #(jump and link register)

- $rd \leftarrow PC + 4$
- $PC \leftarrow rs1 + \text{offset}$ (即imm立即数字段)
- 与load指令得到地址的方式类似

指令语法

类型	原始语法	示例
R	op rd, rs1, rs2	add x5, x6, x7
I	op rd, rs1, imm (寄存器跳转) op rd, offset(rs1) (load载入类) op rd, offset(rs1)	addi x5, x6, -10 jalr x1, 100(x5) ld x5, 40(x6)
S	(store存储类) op rs2, offset(rs1)	sd x5, 40(x6)
B	op rs1, rs2, offset	beq, x5, x6, 100
J	op rd, offset (注意: offset在实际汇编编程中用的是Label)	jal x1, 100
U	op rd, imm	lui x10, 0x87654 # x10 = 0x87654000

S型指令



- 存储指令需要读取两个寄存器，rs1为内存地址，rs2为要写入的数据，以及立即偏移量offset
- **为什么不能像其他指令一样放置rs2和immediate?**
 - 因为S型指令没有写入目的寄存器操作，所以没有rd字段
 - RISC-V的设计决定是将低位的5位立即数移到其他指令中的rd字段所在的位置——保持rs1/rs2字段在同一位置

RISC-V指令格式

- 会查表（基础结构（比如R型）分6个字段：7、5、5、3、5、7）

R 型	funct7	rs2	rs1	funct3	rd	opcode
I 型	imm[11:0]		rs1	funct3	rd	opcode
S 型	Imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB/ B 型	Imm[12,10:5]	rs2	rs1	funct3	imm[4:1,11]	opcode
UJ/ J 型	Imm[20,10:1,11,19:12]				rd	opcode
U 型	Imm[31:12]				rd	opcode

指令语法

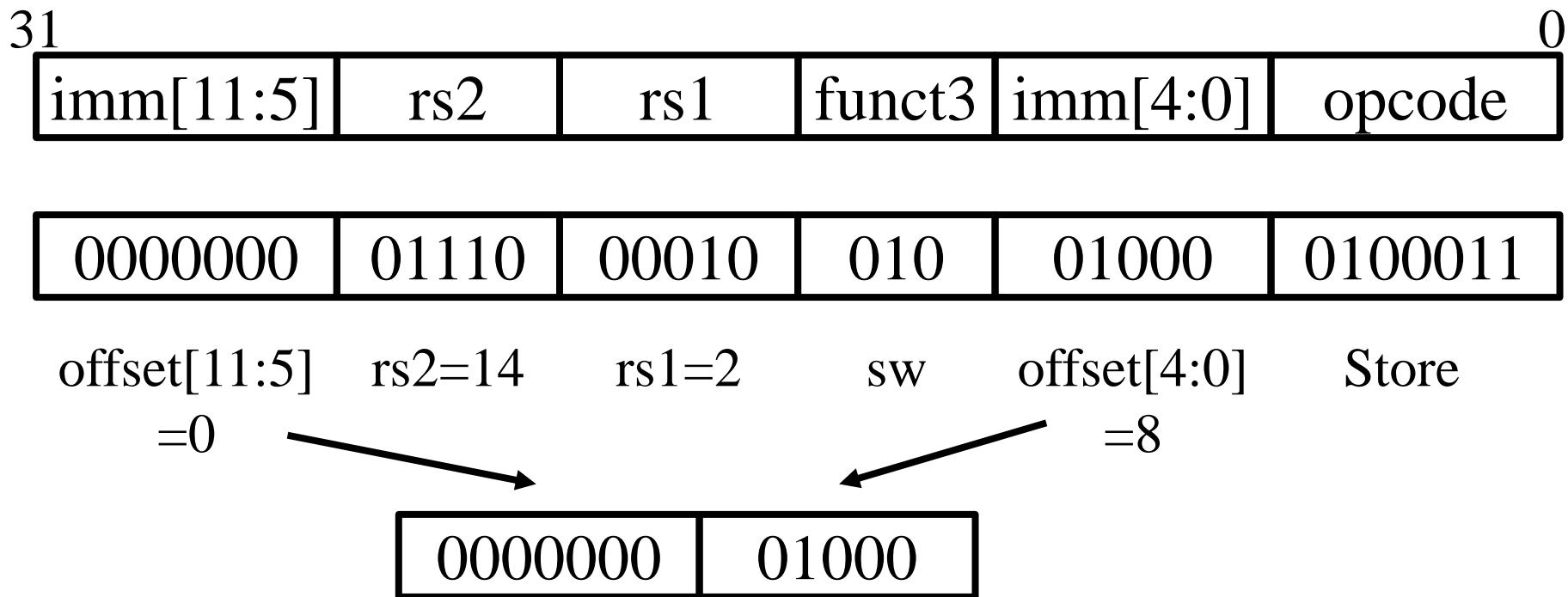
类型	原始语法	示例
R	op rd, rs1, rs2	add x5, x6, x7
I	op rd, rs1, imm (寄存器跳转) op rd, offset(rs1) (load载入类) op rd, offset(rs1)	addi x5, x6, -10 jalr x1, 100(x5) ld x5, 40(x6)
S	(store存储类) op rs2, offset(rs1)	sd x5, 40(x6)
B	op rs1, rs2, offset	beq, x5, x6, 100
J	op rd, offset (注意: offset在实际汇编编程中用的是Label)	jal x1, 100
U	op rd, imm	lui x10, 0x87654 # x10 = 0x87654000

S型指令

- ## • RISC-V汇编S型指令

sw x14, 8(x2)

S型指令语法: `op rs2, offset(rs1)`



S型指令举例

- 所有S型指令

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
imm[11:5]	rs2	rs1	011	imm[4:0]	0100011	sd

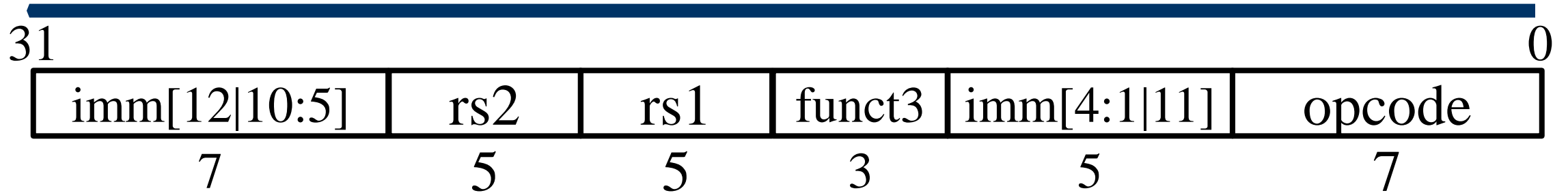
指令语法

类型	原始语法	示例
R	op rd, rs1, rs2	add x5, x6, x7
I	op rd, rs1, imm (寄存器跳转) op rd, offset(rs1) (load载入类) op rd, offset(rs1)	addi x5, x6, -10 jalr x1, 100(x5) ld x5, 40(x6)
S	(store存储类) op rs2, offset(rs1)	sd x5, 40(x6)
B	op rs1, rs2, offset	beq, x5, x6, 100
J	op rd, offset (注意: offset在实际汇编编程中用的是Label)	jal x1, 100
U	op rd, imm	lui x10, 0x87654 # x10 = 0x87654000

B型指令

- 主要用于条件分支或循环等
 - beq, bne, blt, bge, bltu, bgeu
- B型指令读取两个寄存器，但不写回寄存器
 - 类似于S型指令
- 指令格式中如何对Label进行编码？（用哪些字段）
 - 条件或循环体通常很小（<50条指令）
 - SPEC基准测试中约有一半条件分支跳转距离小于16条指令
 - 最大分支转移距离与代码量有关
 - 当前指令的地址存储在程序计数器（PC）中

B型指令



- B型指令格式与S型指令格式基本相同（也叫SB型）
- PC相对寻址：将立即数字段作为相对PC的补码偏移量
 - 立即数字段隐含以2字节为增量，偏移量范围**约**为 $\pm 2^{11} \times \text{2字节}$ （即：-4096 ~ 4094**字节**）
- **用12位立即数字段表示13位（含隐藏0）有符号字节地址的偏移量**

B型指令

问题1：为什么不只使用字节作为PC的偏移量单位？

- 因为指令是32位（4字节），压缩指令也是2字节。

问题2：若将偏移量视为以字为单位，在相对寻址前，先将偏移量乘以4，得到字节偏移量，再加上PC中的基地址，可否这样设计？

- 虽然可以访问到相对PC地址前后**共** $2^{12} \times 4$ 字节范围内的所有指令，比使用字节偏移量大四倍的转移范围，**但不能支持16位压缩扩展指令寻址。**

B型指令

- 基于RISC-V的扩展指令集支持16位压缩编码指令，也支持16位长度倍数的可变长度指令
- 为了实现对可能的扩展指令的支持，在实际的RISC-V指令设计中，分支转移指令的偏移量都是以2字节为单位

因此，立即数字段默认隐藏低位的0，无需存储

- RISC-V条件分支指令实际上只能访问相对PC地址前后

共 $2^{11} \times 4$ 字节（即 $\pm 2^{11} \times 2$ 字节或-4096 ~ 4094字节）范围内的指令

B型指令

- RISC-V汇编指令:

- Loop: beq x19, x10, End

- add x18, x18, x10

- addi x19, x19, -1

- j Loop

End: # 目标指令 如何对Label进行编码，转移位置如何得到？

???????	01010	10011	000	?????	1100011
imm	rs2=10	rs1=19	BEQ	imm	Branch

B型指令

- RISC-V汇编指令:

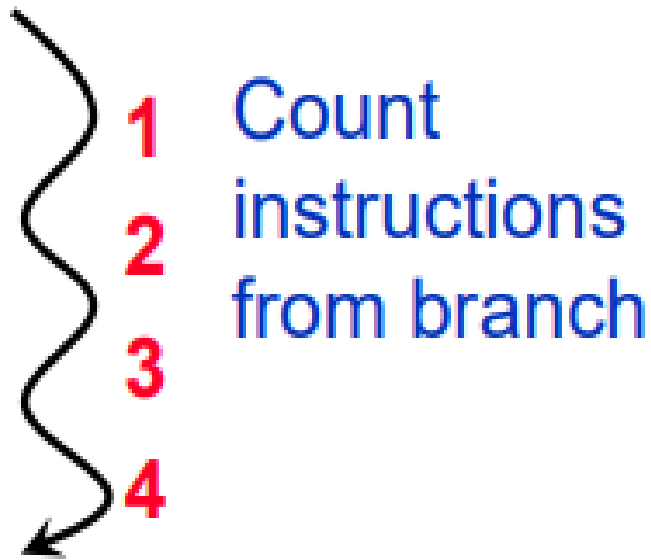
- Loop: **beq x19, x10, End**

- add x18, x18, x10

- addi x19, x19, -1

- j Loop

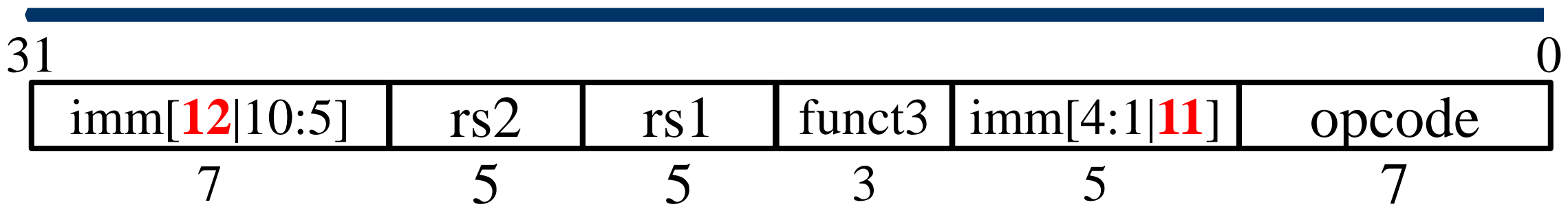
- End: # 目标指令



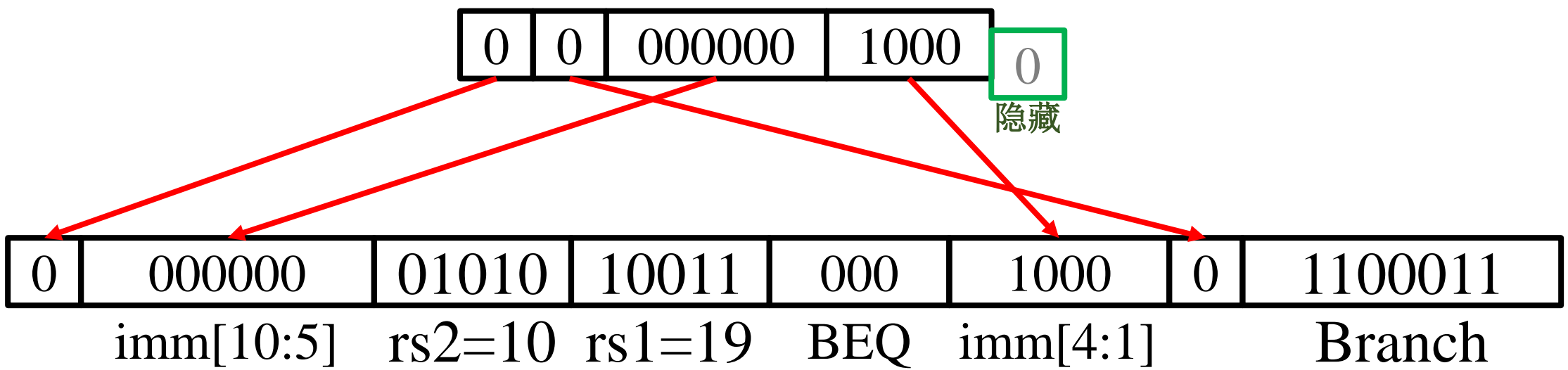
- $\text{offset} = 4 \times 32\text{-bit instructions} = 16 \text{ bytes} = 8 \times 2 \text{ bytes}$

B型指令举例

B型指令语法: **op rs1, rs2, offset**



beq x19, x10, offset = 8 × 2 bytes



RISC-V指令格式

- 会查表（基础结构（比如R型）分6个字段：7、5、5、3、5、7）

R 型	funct7	rs2	rs1	funct3	rd	opcode
I 型	imm[11:0]		rs1	funct3	rd	opcode
S 型	Imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB/ B 型	Imm[12,10:5]	rs2	rs1	funct3	imm[4:1,11]	opcode
UJ/ J 型	Imm[20,10:1,11,19:12]				rd	opcode
U 型	Imm[31:12]				rd	opcode

指令语法

类型	原始语法	示例
R	op rd, rs1, rs2	add x5, x6, x7
I	op rd, rs1, imm (寄存器跳转) op rd, offset(rs1) (load载入类) op rd, offset(rs1)	addi x5, x6, -10 jalr x1, 100(x5) ld x5, 40(x6)
S	(store存储类) op rs2, offset(rs1)	sd x5, 40(x6)
B	op rs1, rs2, offset	beq, x5, x6, 100
J	op rd, offset (注意: offset在实际汇编编程中用的是Label)	jal x1, 100
U	op rd, imm	lui x10, 0x87654 # x10 = 0x87654000

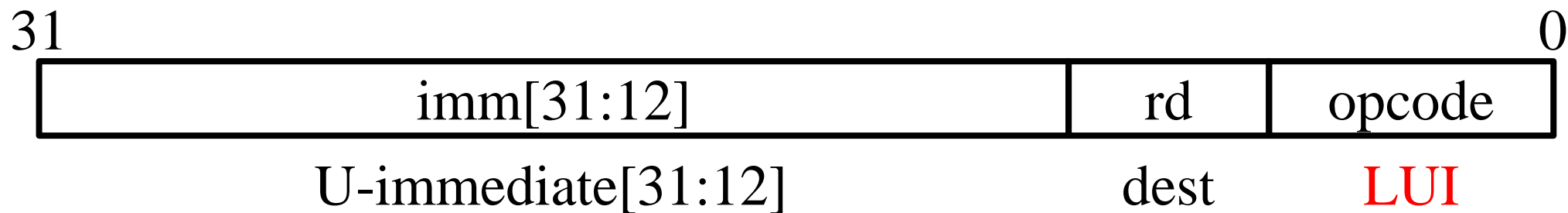
RISC-V所有B型指令

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


指令语法

类型	原始语法	示例
R	op rd, rs1, rs2	add x5, x6, x7
I	op rd, rs1, imm (寄存器跳转) op rd, offset(rs1) (load载入类) op rd, offset(rs1)	addi x5, x6, -10 jalr x1, 100(x5) ld x5, 40(x6)
S	(store存储类) op rs2, offset(rs1)	sd x5, 40(x6)
B	op rs1, rs2, offset	beq, x5, x6, 100
J	op rd, offset (注意: offset在实际汇编编程中用的是Label)	jal x1, 100
U	op rd, imm	lui x10, 0x87654 # x10 = 0x87654000

U型指令 语法: **op rd, imm**



AUIPC

- ADDI等I型指令中的立即数范围是多少? 
- U型指令进行**长**立即数操作
 - 高20位为长度为20位的立即数字段
 - 5位目的地寄存器字段
- LUI(**L**oad **U**pper **I**mmediate)—将长立即数写入目的寄存器
- AUIPC(**A**dd **U**pper **I**mmediate to **PC**)——将PC与长立即数相加结果写入目的寄存器

U型指令——LUI(Load Upper Immediate)

- LUI将长立即数写入目的寄存器的高20位，并清除低12位。
- 注意：在RV64里面还需要符号位扩展。

LUI rd, imm # R[rd]= {32b'imm<31>, imm, 12'b0}

- LUI与ADDI一起可在寄存器中创建任何32位值

LUI t0, 0x87654 #t0 = 0x87654000

ADDI t0, t0, 0x321 # 本条语句执行后t0 = 0x87654321

Code	Basic	Source		
0x876542b7	LUI x5, 0x00087654	2: LUI	t0, 0x87654	#t0 = 0x87654000
0x32128293	ADDI x5, x5, 0x00000321	3: ADDI	t0, t0, 0x321	# t0 = 0x87654321

U型指令LUI的应用

- **RV32中**，如何创建 0xDEADBEEF?

LUI x10, 0xDEADB # x10 = 0xDEADB000

ADDI x10, x10, -0x111 # x10 = 0xDEADAEEF? 出错

- ADDI 立即数总是进行**符号扩展**，如果高位为1，将从高位的20位中减去1

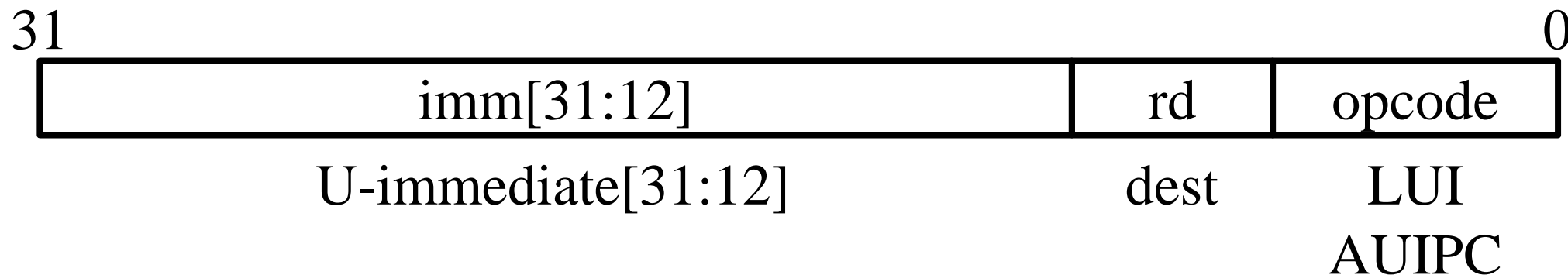
LUI x10, 0xDEADC # x10 = 0xDEADC000

ADDI x10, x10, -0x111 # x10 = 0xDEADBEEF

- 伪指令 li x10, 0xDEADBEEF # 创建两条指令

注意： -0x111在RV32通用寄存器里的补码表示是0xFFFFFEEF

U型指令—AUIPC(Add Upper Immediate to PC)



- 将长立即数值加到PC并写入目的寄存器
- 用于PC相对寻址

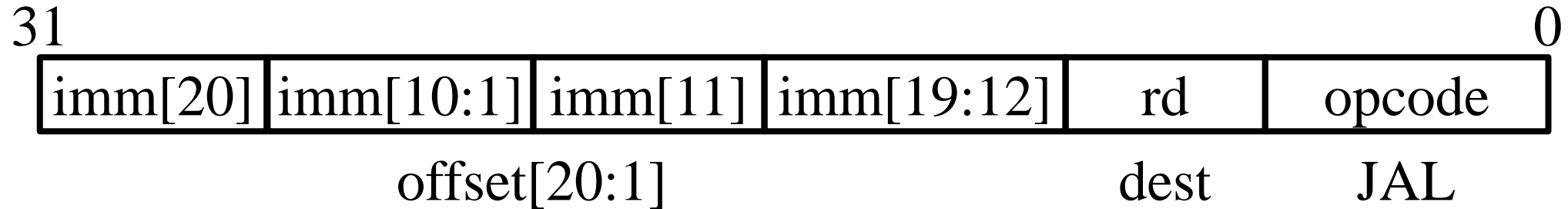
Label: AUIPC rd, imm **#** R[rd]=PC+{imm,12'b0}

例子: Label: AUIPC x10, 0 # 将当前指令地址 (PC) 放入x10,
即Label标签地址放入x10

指令语法

类型	原始语法	示例
R	op rd, rs1, rs2	add x5, x6, x7
I	op rd, rs1, imm (寄存器跳转) op rd, offset(rs1) (load载入类) op rd, offset(rs1)	addi x5, x6, -10 jalr x1, 100(x5) ld x5, 40(x6)
S	(store存储类) op rs2, offset(rs1)	sd x5, 40(x6)
B	op rs1, rs2, offset	beq, x5, x6, 100
J	op rd, offset (注意: offset在实际汇编编程中用的是Label)	jal x1, 100
U	op rd, imm	lui x10, 0x87654 # x10 = 0x87654000

J型指令 语法: **op rd, offset**



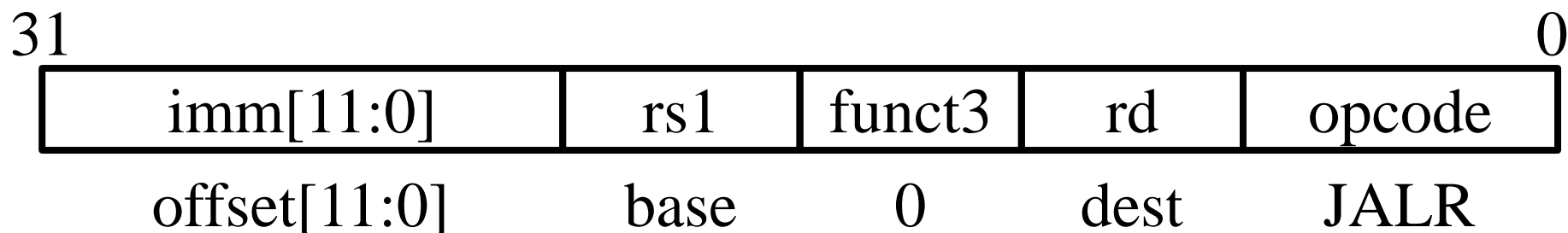
- JAL将PC+4写入目的寄存器rd中
 - J是伪指令，等同于JAL，但设置rd = x0不保存返回地址
- $PC = PC + offset$ （PC相对寻址）
- 访问相对PC，以大约 $\pm 2^{19} \times 2$ 字节范围内的地址空间
 - 大约 $\pm 2^{18} \times 4$ 字节（或 $\pm 2^{20}$ 字节 或 共 2^{21} 字节）指令空间
- 立即数字段编码的优化类似于B型指令，以降低硬件成本

RISC-V指令格式

- 会查表（基础结构（比如R型）分6个字段：7、5、5、3、5、7）

R 型	funct7	rs2	rs1	funct3	rd	opcode
I 型	imm[11:0]		rs1	funct3	rd	opcode
S 型	Imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB/ B 型	Imm[12,10:5]	rs2	rs1	funct3	imm[4:1,11]	opcode
UJ/ J 型	Imm[20,10:1,11,19:12]				rd	opcode
U 型	Imm[31:12]				rd	opcode

(回顾对比) jalr指令



- 注意: jalr属于I型指令, 但是jal是J型指令
- 将PC + 4保存在rd中
- 设置 $PC = rs1 + immediate$
- 与load指令得到地址的方式类似
- 与B型指令和jal不同, **jalr无需将立即数×2字节, 无隐藏位**

lui, auipc, jalr指令的应用

- ret 和 jr 伪指令
 - `ret` = `jr ra = jalr x0, 0(ra) == jalr x0, 0(x1)`
- 对任意32位绝对地址处的函数进行调用，原理（非语法格式）：
 - `lui x1, <hi20bits>`
 - `jalr ra, x1, <lo12bits>`
- 相对PC地址32位偏移量的相对寻址，原理（非语法格式）：
 - `auipc x1, <hi20bits>`
 - `jalr x0, x1, <lo12bits>`

第三章 RISC-V汇编及其指令系统

- **RISC-V指令表示**

- RISC-V六种指令格式

- **RISC-V寻址模式介绍**

- 寄存器寻址

- 立即数寻址

- 基址寻址（偏移寻址）

- PC相对寻址



寄存器寻址

- 操作数在寄存器中

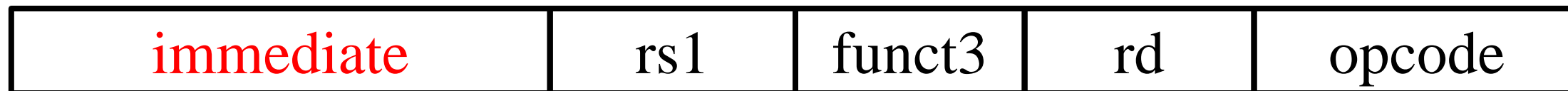
- add x1, x2, x3

- and x5, x6, x7



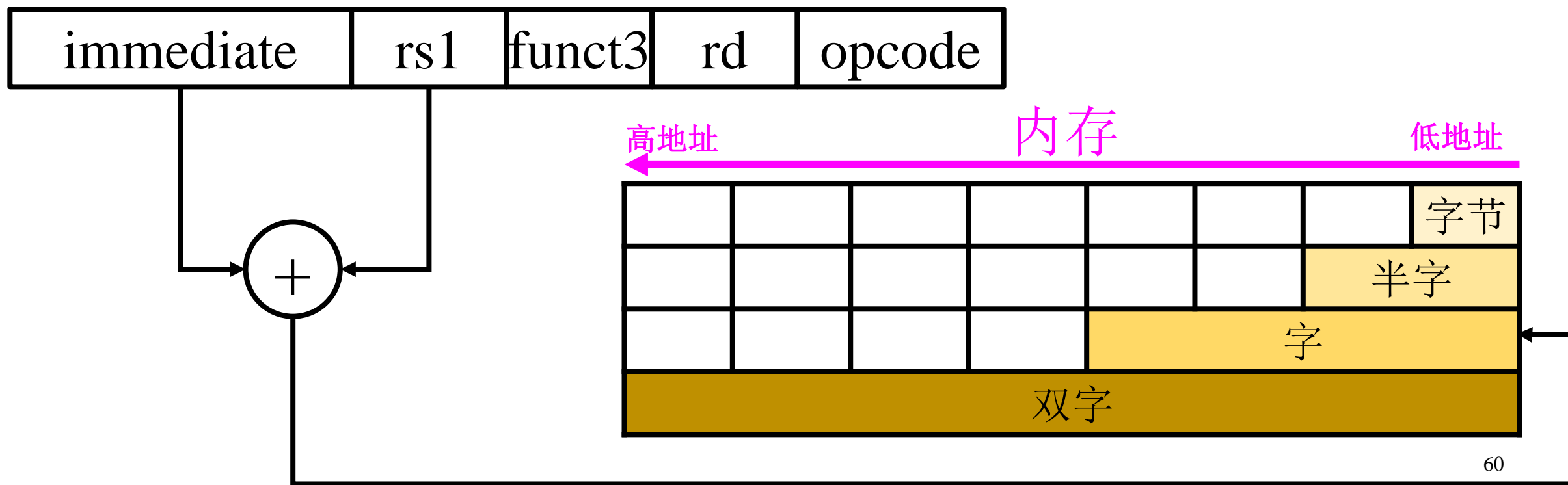
立即数寻址

- 操作数是指令本身的常量
 - addi x3, x4, 10
 - **andi x5, x6, 3**



基址或偏移寻址

- 操作数于内存中，其地址是寄存器和指令中的常量之和
 - `lw x10, 12(x15)`
 - 基址寄存器(x15) + 偏移量(12)



PC相对寻址

- 分支地址是PC与指令中立即数之和

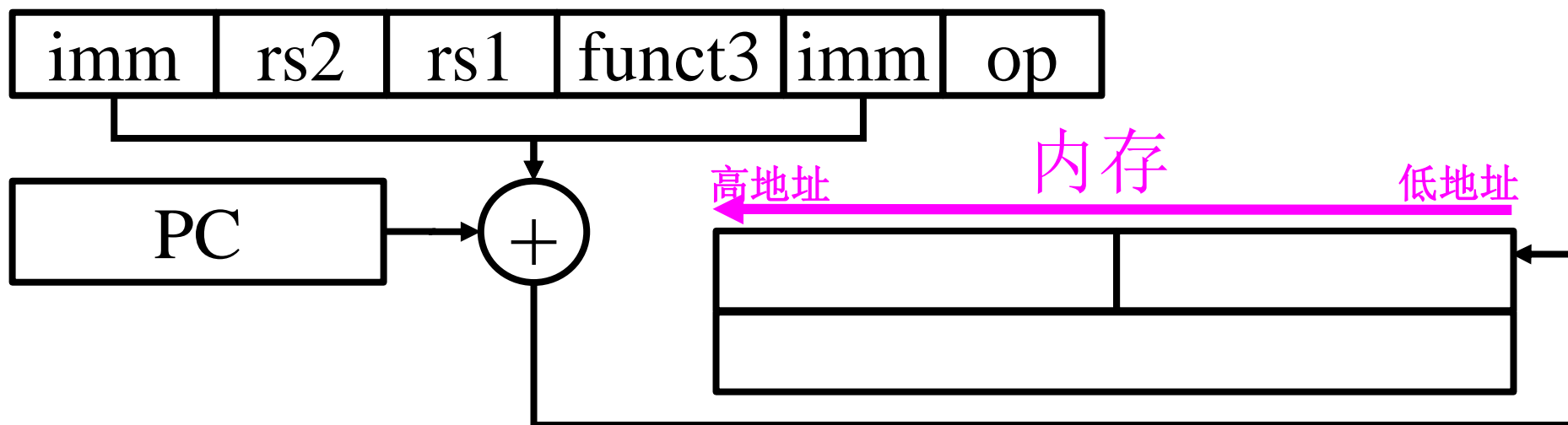
Loop: **beq x19, x10, End**

add x18, x18, x10

addi x19, x19, -1

jal x0, Loop

End:



RISC-V重点总结

- 熟练查找指令格式
- 记忆汇编指令语法

RISC-V指令格式（熟练查表）

- 基本指令（比如R型）划分为6个字段：7、5、5、3、5、7

R 型	funct7	rs2	rs1	funct3	rd	opcode
I 型	imm[11:0]		rs1	funct3	rd	opcode
S 型	Imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB/ B 型	Imm[12,10:5]	rs2	rs1	funct3	imm[4:1,11]	opcode
UJ/ J 型	Imm[20,10:1,11,19:12]				rd	opcode
U 型	Imm[31:12]				rd	opcode

指令语法（需要记忆）

类型	原始语法	示例
R	op rd, rs1, rs2	add x5, x6, x7
I	op rd, rs1, imm (寄存器跳转) op rd, offset(rs1)	addi x5, x6, -10 jalr x1, 100(x5)
	(load载入类) op rd, offset(rs1)	ld x5, 40(x6)
S	(store存储类) op rs2, offset(rs1)	sd x5, 40(x6)
B	op rs1, rs2, offset	beq, x5, x6, 100
J	op rd, offset (注意: offset在实际汇编编程中用的是Label)	jal x1, 100
U	op rd, imm	lui x10, 0x87654 # x10 = 0x87654000