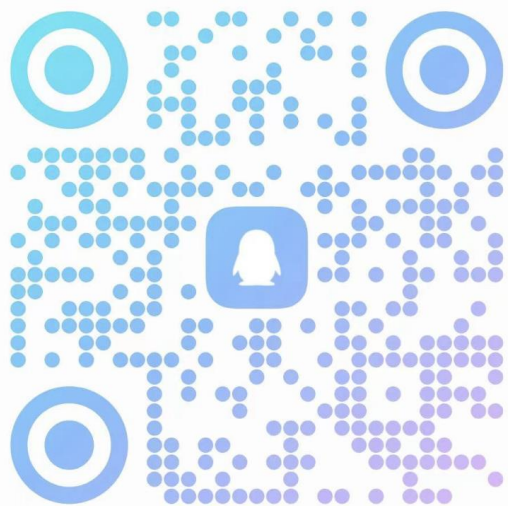


计算机组成原理

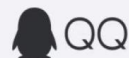


2025春计算机...

群号: 892932264



扫一扫二维码，入群聊



花忠云

<https://huazhongyun.github.io/>

<http://faculty.hitsz.edu.cn/huazhongyun>

计算机科学与技术学院

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

- 我们处理的大多数数据都是字（32-bit）：
 - lw和sw一次访问一个字的内存
- 指令字长、机器字长、存储字长
 - 指令字长：指令中二进制代码的总位数
 - 机器字长：直接处理的二进制位数，一般等于CPU寄存器的总位数
 - 存储字长：一个存储单元存储的二进制代码的长度

RISC-V指令表示

- 那么我们如何表示指令呢？
 - 记住：计算机只懂1和0，所以汇编字符串“add x10, x11, x0”对硬件来说毫无意义
 - RISC-V追求简单性：因为数据是以字为单位的，所以指令也应该是固定大小的32位字
 - 用于RV32、RV64、RV128的32位指令相同

RISC-V指令表示

指令如何表示？

简单源于规整 RISC-V的指令都是32位长（RV32、RV64）

add x9, x20, x21

指令32位，将指令字分成“字段”

每个字段有不同的含义

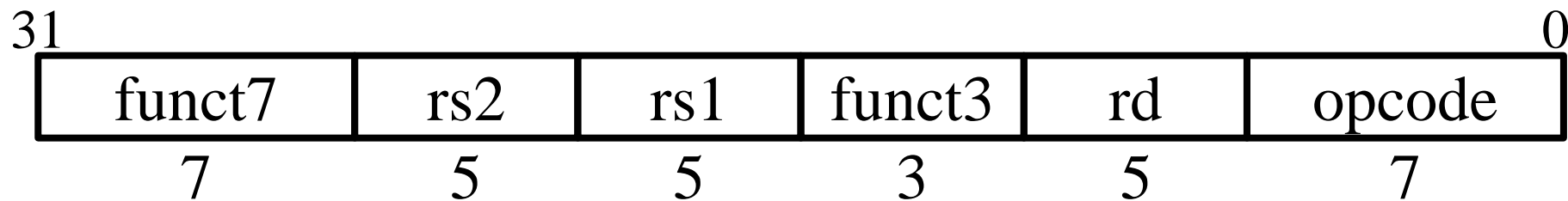
RISC-V指令表示

- 一个字是32位，所以把指令字分成“**字段**”
- 每个字段都告诉处理器一些关于指令的信息
- 理论上可以为**每条指令定义不同的字段**
- RISC-V定义了六种基本类型的**指令格式**:
 - **R型**指令 用于**寄存器 - 寄存器**操作
 - **I型**指令 用于**短立即数**和**取数 load** 操作
 - **S型**指令 用于**存数 store** 操作
 - **B型**指令 用于**条件跳转**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指令格式

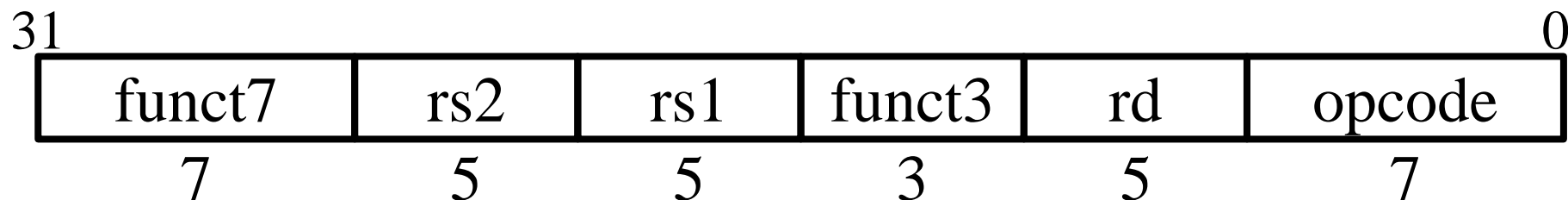
R 型	funct7	rs2	rs1	funct3	rd	opcode
I 型	imm[11:0]		rs1	funct3	rd	opcode
S 型	Imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
B 型	Imm[12,10:5]	rs2	rs1	funct3	imm[4:1,11]	opcode
J 型	Imm[20,10:1,11,19:12]				rd	opcode
U 型	Imm[31:12]				rd	opcode

R型指令



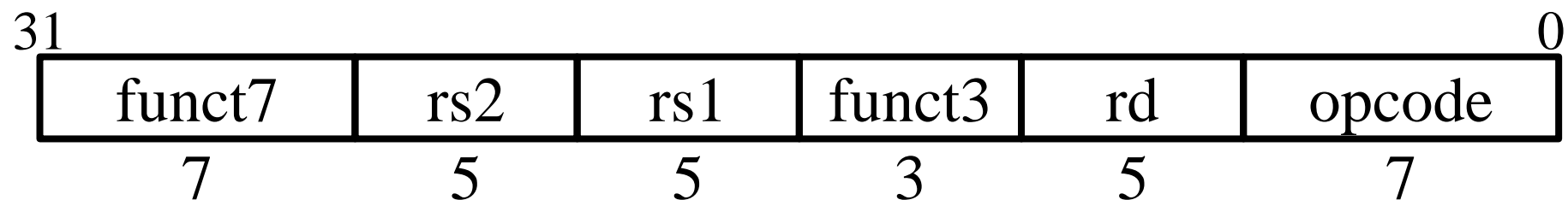
- 32位指令字，分为6个不同位数的字段
- **opcode(操作码)**: 指定它是什么指令
- **funct7+funct3**: 这两个字段结合操作码描述要执行的操作

R型指令



- **rs1**（源寄存器1）：指定第一个寄存器操作数
- **rs2**（源寄存器2）：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）（寄存器约定详见P75图2-14）

add x18, x19, x10



opcode='0110011', funct7='0000000', funct3='000'

对应的机器指令码为: [填空1]

正常使用填空题需3.0以上版本雨课堂

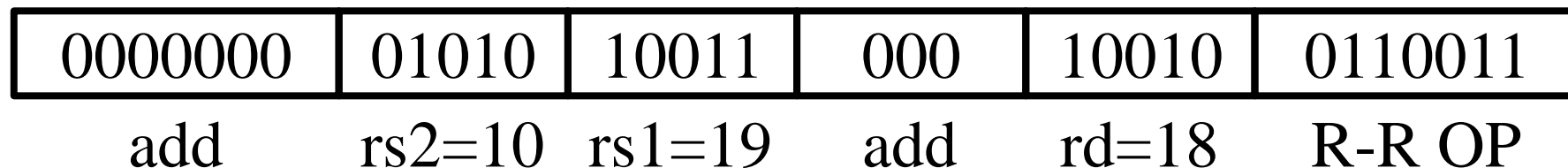
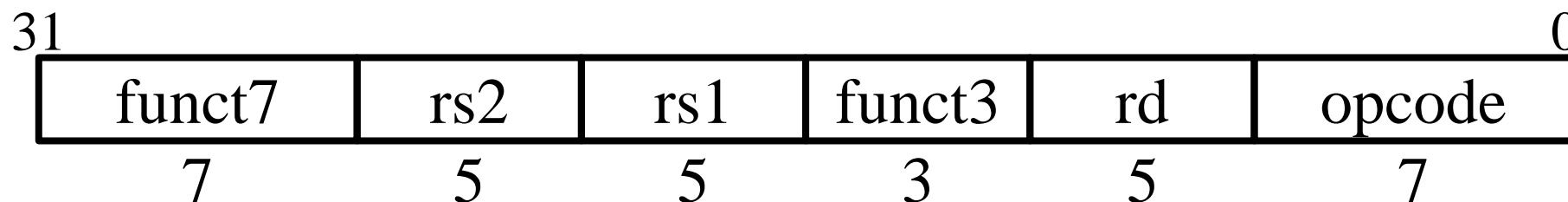
作答

R型指令

- RISC-V汇编指令:

- add x18, x19, x10

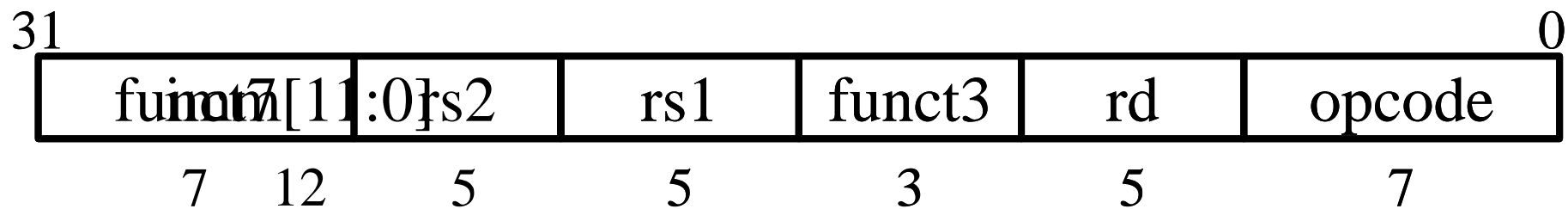
opcode='0110011', funct7='0000000',
funct3='000'



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

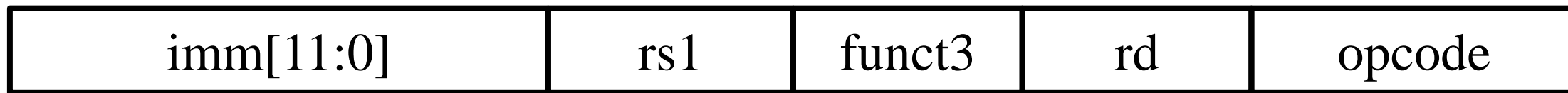


- 只有一个字段与R型指令不同，rs2和funct7被12位有符号立即数imm[11:0]替换
- 其余字段（rs1、funct3、rd、opcode）与之前相同
- 在算术运算前，立即数需符号扩展到32位（RV32）
- 如何处理大于12位的立即数？

addi x15, x1, -50

31

0



opcode = '0010011', funct3 = '000'

对应的机器指令码为: [填空1]

正常使用填空题需3.0以上版本雨课堂

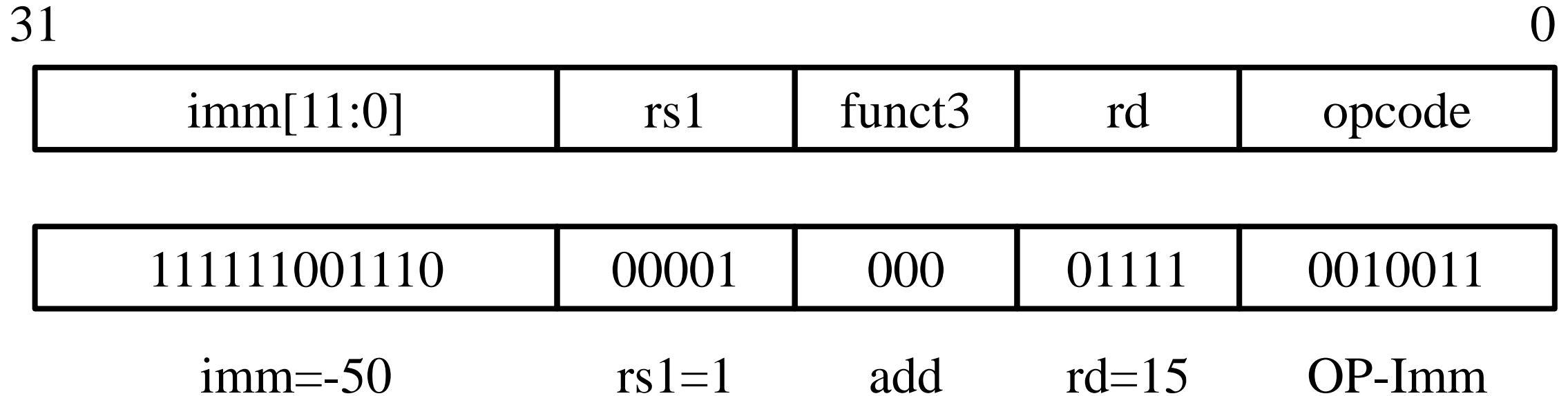
作答

I型指令

- RISC-V汇编指令:

- addi x15, x1, -50

opcode = '0010011', funct3 = '000'



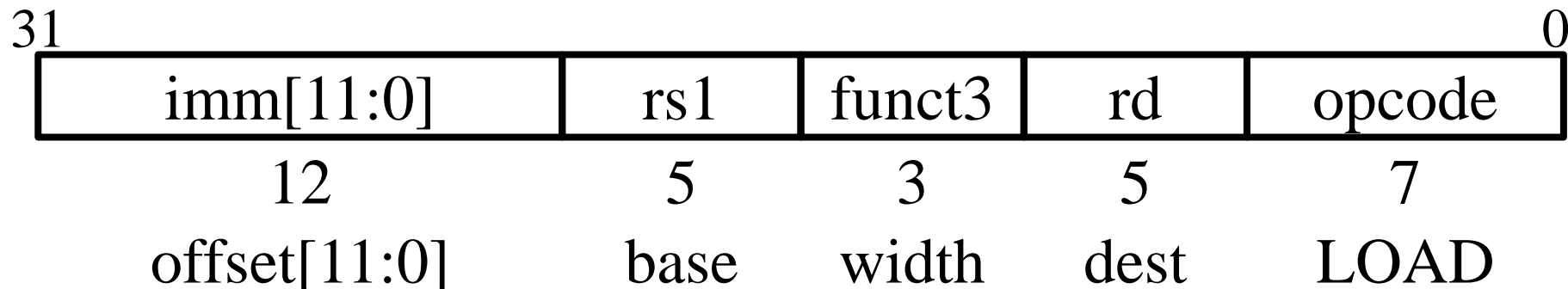
I型指令

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）

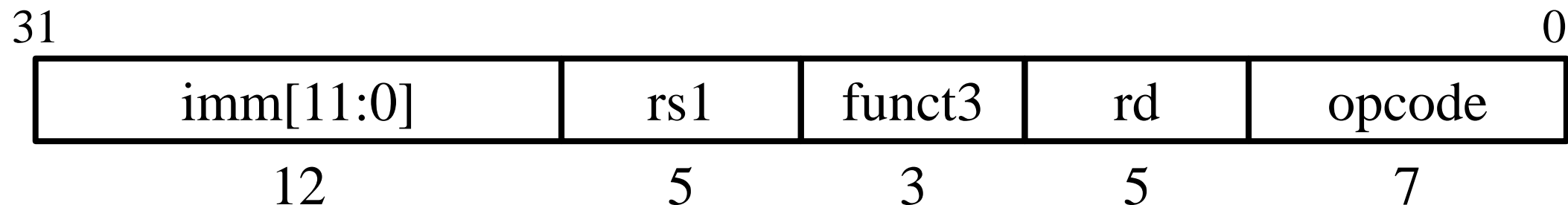
“按立即数移位”指令仅将立
即数的低位6位用作移位量（只
能移位0-63位位置）

I型指令



- **Load**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**
- 从**内存读取**到的值**存储**在寄存器rd中

lw x14, 8(x2)



opcode = '0000011', funct3 = '010'

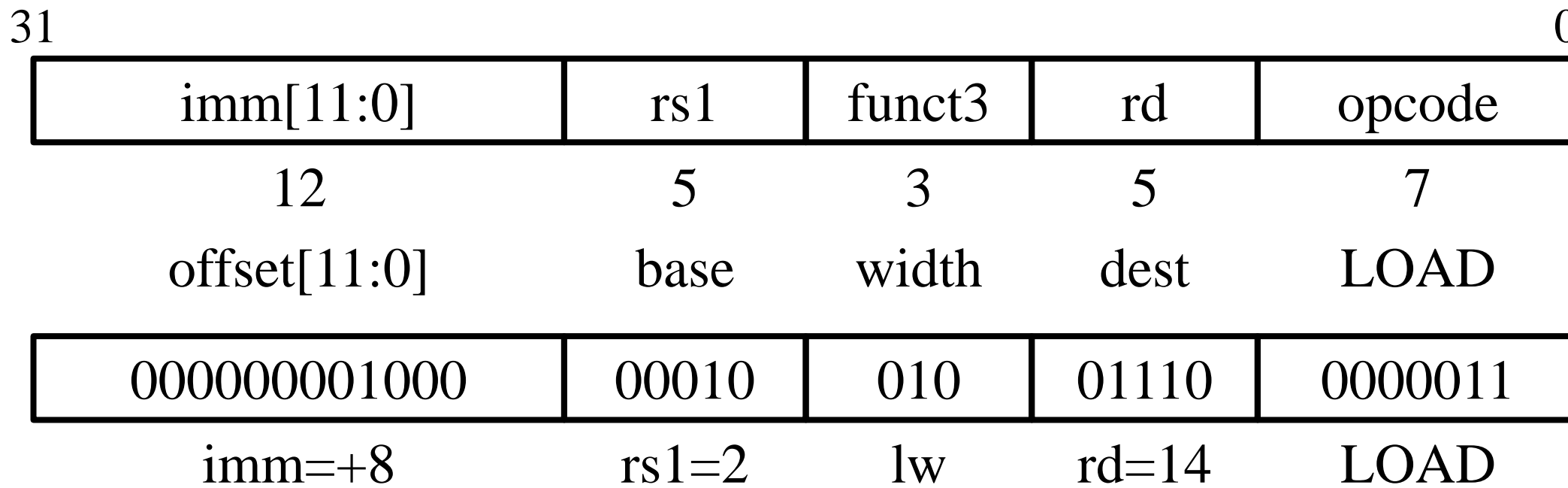
对应的机器指令码为: [填空1]

I型指令—Load

- RISC-V汇编指令:

- lw x14, 8(x2)

opcode = '0000011', funct3 = '010'



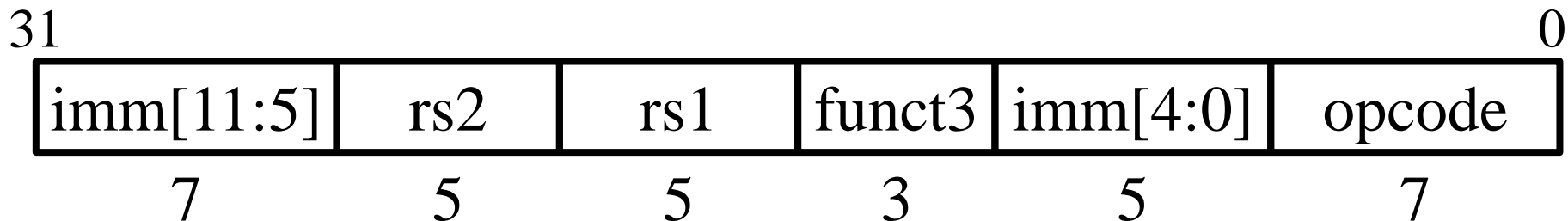
I型指令—Load

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

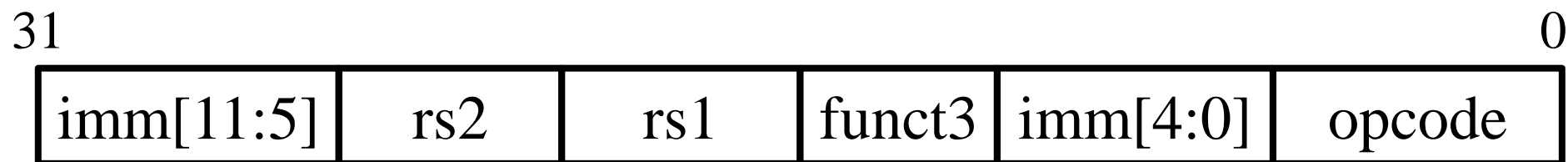
- lbu是“读取无符号字节”
- lh是“读取半字”，读取16位（2字节）并扩展以填充32(或64)位寄存器
- lhu是“读取无符号半字”，零扩展为16位以填充32 (或64)位寄存器

S型指令



- 存储需要读取两个寄存器，**rs1**为**内存地址**，**rs2**为要**写入的数据**，以及立即数**偏移量offset**
- 12位immediate是分开的，要保证rs2**位置不变**。
- S型指令不会将值写入寄存器，所以没有rd
- 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**

sw x14, 8(x2)



opcode = '0100011', funct3 = '010'

对应的机器指令码为: [填空1]

正常使用填空题需3.0以上版本雨课堂

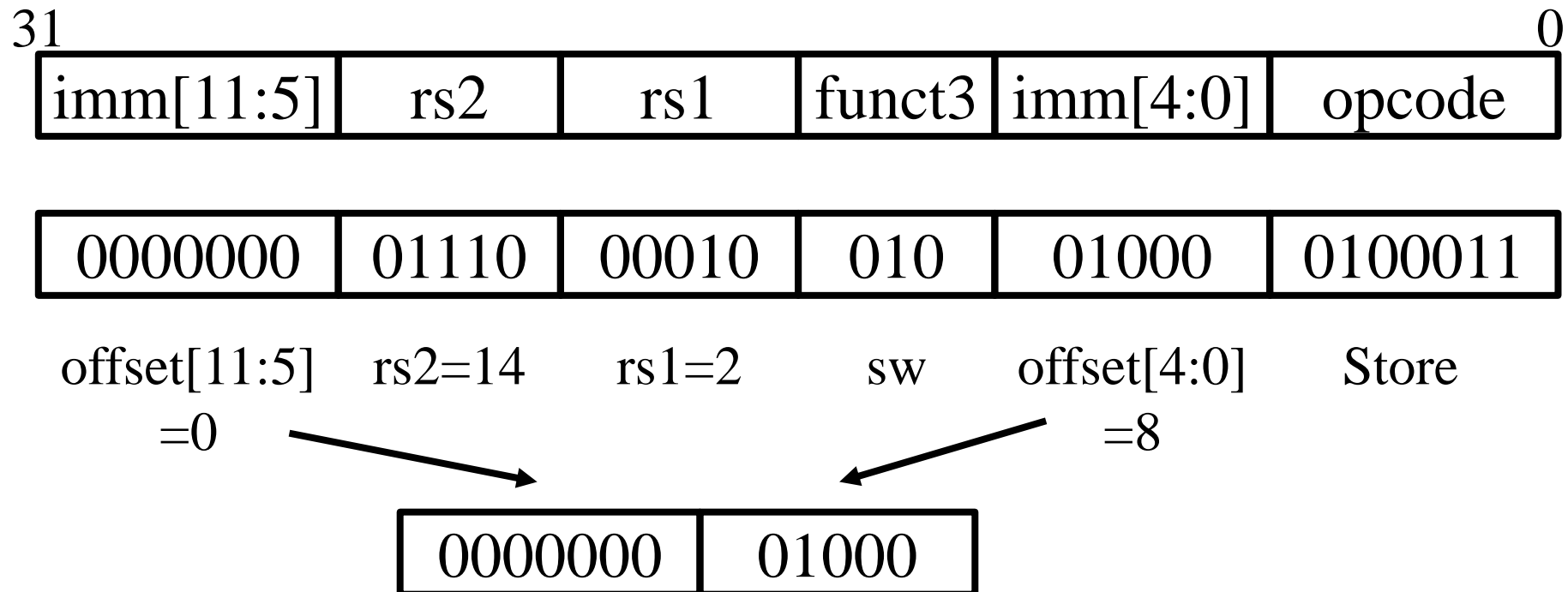
作答

S型指令

- RISC-V汇编S型指令

sw x14, 8(x2)

opcode = '0100011', funct3 = '010'



S型指令

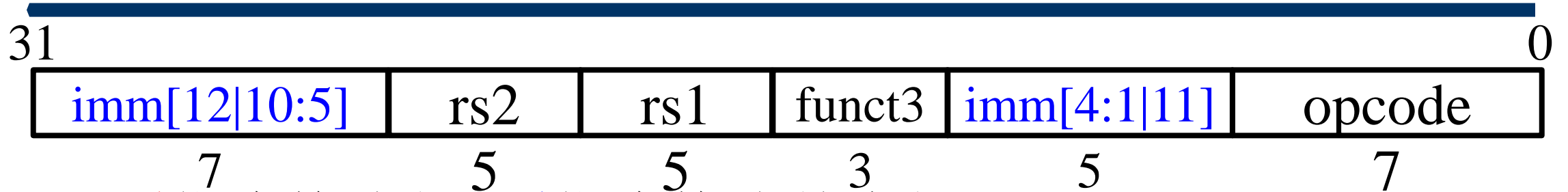
- 所有 S 型指令

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(RV64 才有)

B型指令

- 用于条件语句或循环语句（if-else、while、for）
 - 条件或循环体通常很小（< 50条指令）
- 指令存储在比较集中的区域
 - 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
- PC相对寻址：将立即数字段作为相对PC的**补码偏移量**
- 立即数字段以**2字节**为增量表示**-4096**到**+4094**的偏移量
- 用**12位**立即数字段表示**13位有符号字节地址**的偏移量
 - 偏移量的**最低位**始终为**零**，因此无需存储

B型指令

- 为什么不使用字节作为PC的偏移量单位？
 - 因为指令是32位（4字节）
- 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
 - 可以访问到相对PC地址前后 $2^{12} \times 32$ 位范围内的所有指令
 - 比使用字节偏移量大四倍的转移范围

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
- RISC-V条件分支指令实际上只能访问相对PC地址前后 $2^{12} \times 16$ 位范围内的指令

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
```

imm	rs2	rs1	BEQ	imm	Branch
			000		1100011

对应的机器指令码为 [填空1]

正常使用填空题需3.0以上版本雨课堂

作答

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

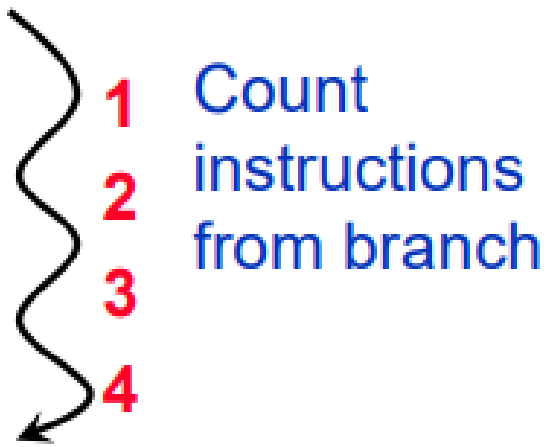
- End: # 目标指令

???????	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



Count instructions from branch

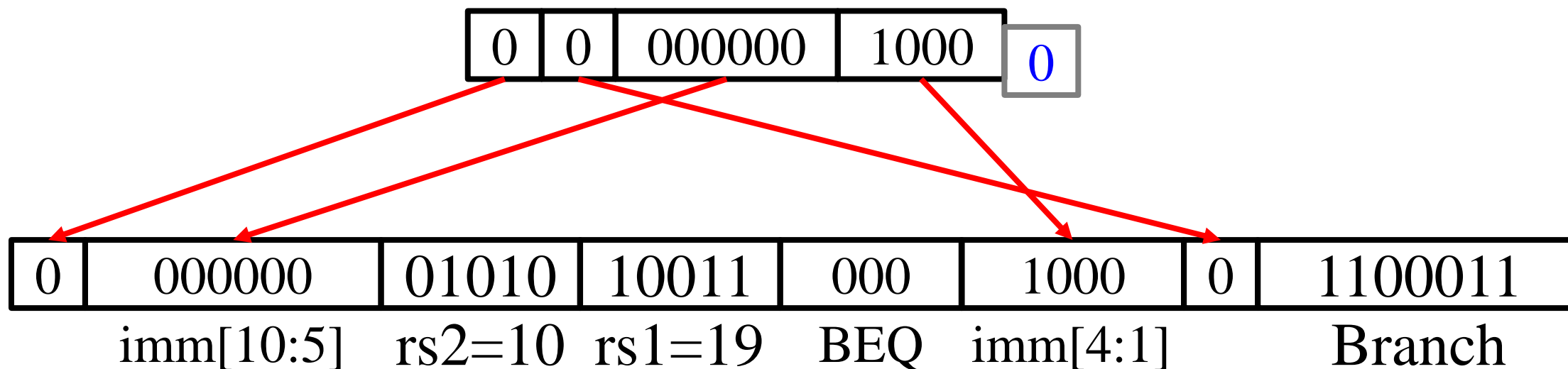
End: # 目标指令

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

B型指令

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

beq x19, x10, offset = 8×2 bytes



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

U型指令



U-immediate[31:12]

dest

LUI

AUIPC

- ADDI指令中的立即数**最大**是多少？
- U型指令进行长立即数操作
 - **高20**位的立即数字段
 - 5位的目的地寄存器字段
- 两个指令
 - LUI(Load Upper Immediate)——将长立即数写入目的寄存器
 - AUIPC——将PC与长立即数相加结果写入目的寄存器

U型指令LUI的应用

- LUI将长立即数写入目的寄存器的高20位，并清除低12位。
- LUI与ADDI一起可在寄存器中创建任何32位值

LUI x10, 0x87654 # x10 = 0x87654000

ADDI x10, x10, 0x321 # x10 = 0x87654321

U型指令LUI的应用

- 如何创建 0xDEADBEEF?

LUI x10, 0xDEAD**B** # x10 = 0xDEADB000

ADDI x10, x10, -0x111 (0xFFFFFEEF) # x10 = 0xDEAD**A**EEF?

- 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, 0xDEAD**C** # x10 = 0xDEADC000

ADDI x10, x10, -0x111 (0xFFFFFEEF) # x10 = 0xDEAD**B**EEF

U型指令LUI的应用

• 伪指令 `li x10, 0xDEADBEEF` # 创建两条指令

➤ LUI `x10, 0xDEADC` # `x10 = 0xDEADC000`

➤ ADDI `x10, x10, -0x111 (0xFFFFFEEF)` # `x10 = 0xDEADBEEF`

```
1
2  li x9, 0x87654321
3  li x10, 0xDEADBEEF
```

Edit

Execute

Text Segment

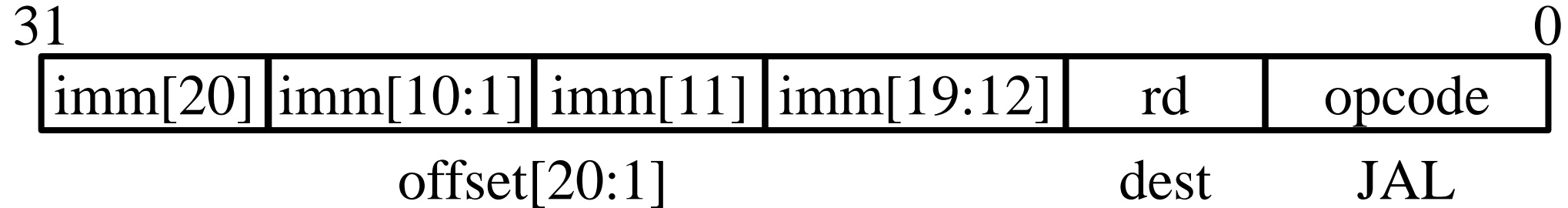
Bkpt	Address	Code	Basic	
<input type="checkbox"/>	4194304	0x876544b7	lui x9, 0xffff87654	2: li x9, 0x87654321
<input type="checkbox"/>	4194308	0x3214849b	addiw x9, x9, 0x00000321	
<input type="checkbox"/>	4194312	0xdeadc537	lui x10, 0xffffdeadc	3: li x10, 0xDEADBEEF
<input type="checkbox"/>	4194316	0xeef5051b	addiw x10, x10, 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 x10, 0 # 将Label地址放入x10

J型指令



- 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2字节为单位的 $\pm 2^{19}$ 范围内的地址空间
 - $\pm 2^{19}$ 16-bit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

J型指令

- j 伪指令

- j Label = jal x0, Label # 设置目的寄存器为x0

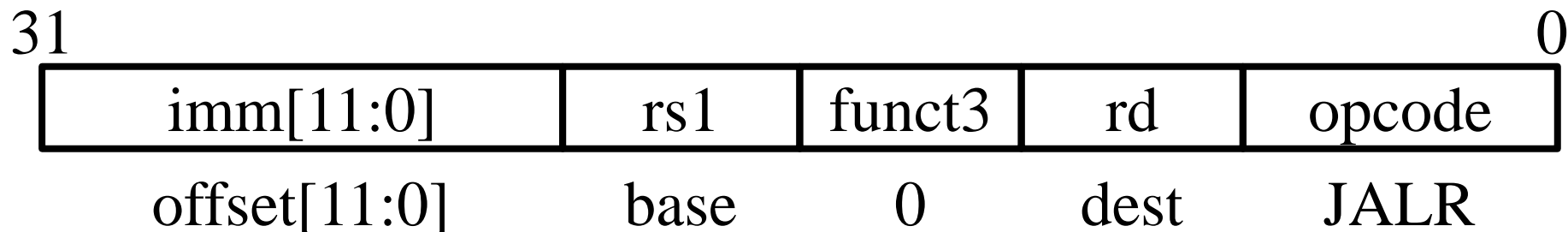
- jal 指令

- jal ra, FuncName

lui,auipc,jalr指令的应用

- ret 和 jr 伪指令
 - ret; **jr** ra; **jalr** x0, 0(ra) 三条指令完成同样功能
- 对任意32位绝对地址处的函数进行调用
 - **lui** x1, <hi20bits> #加载高20位
 - **jalr** ra, <lo12bits>(x1)
- 相对PC地址32位偏移量的相对寻址
 - **auipc** x1, <hi20bits> #高20位+PC
 - **jalr** x0, <lo12bits>(x1)

jalr指令



- JALR属于**I型**指令
- 将 $PC + 4$ 保存在rd中
- 设置 $PC = rs + immediate$
- 与load指令得到地址的方式类似
- 与B型指令和jal不同，**不**将立即数 $\times 2$ 字节

RISC-V指令格式

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
B 型	Imm[12,10:5]	rs2	rs1	funct3	imm[4:1,11]	opcode
J 型	Imm[20,10:1,11,19:12]				rd	opcode
U 型	Imm[31:12]				rd	opcode

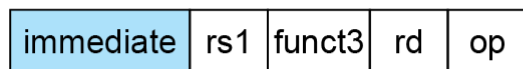
第三章 RISC-V汇编及其指令系统

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

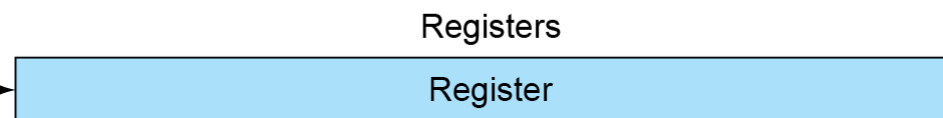
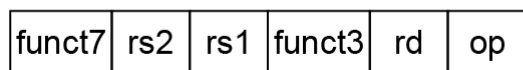


RISC-V的寻址模式

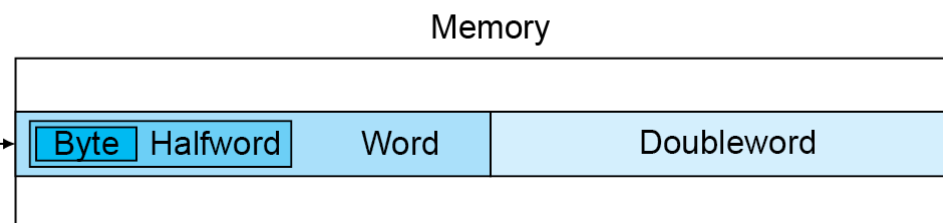
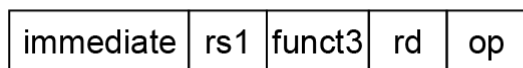
1. Immediate addressing



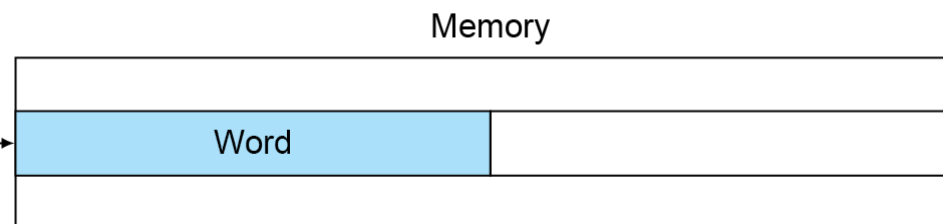
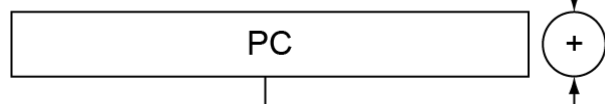
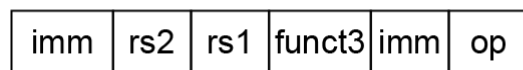
2. Register addressing



3. Base addressing



4. PC-relative addressing



立即数寻址

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

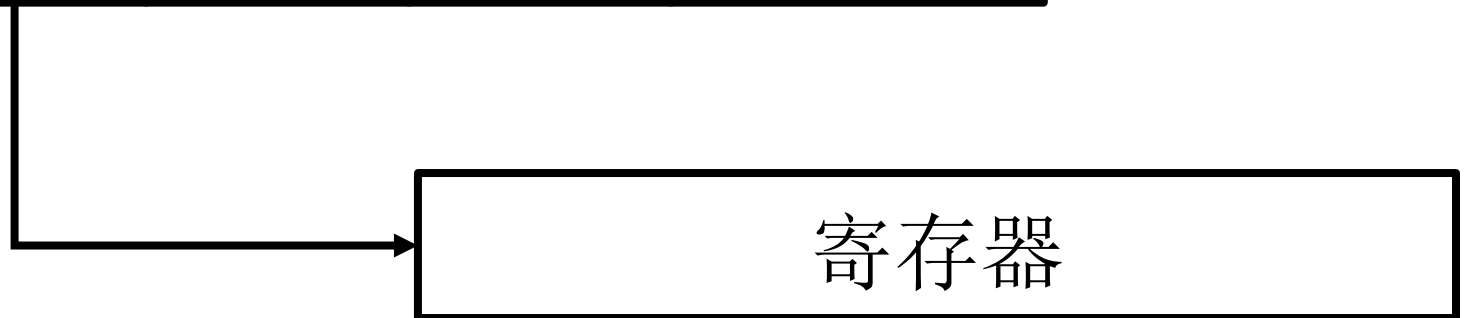


寄存器寻址

- 操作数在寄存器中

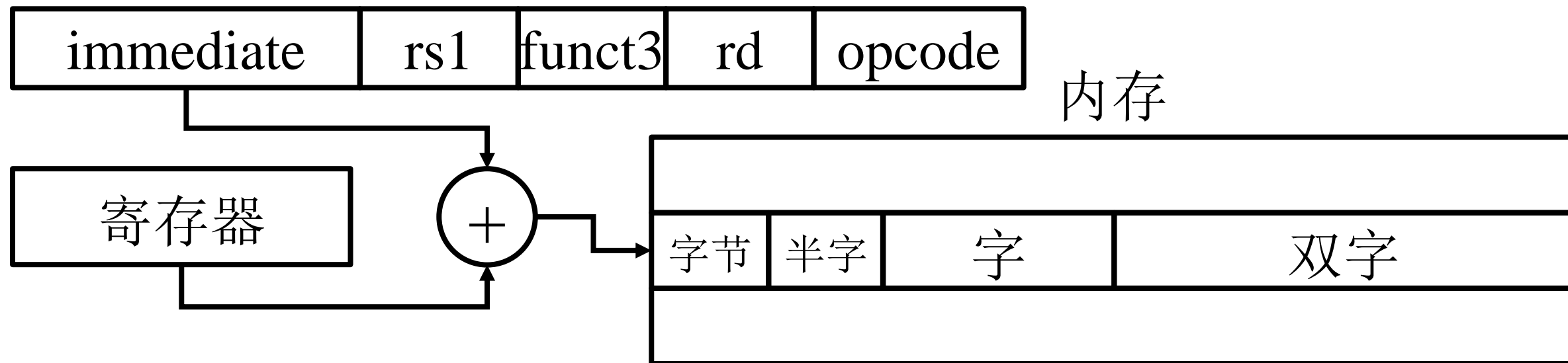
- add x1, x2, x3

- and x5, x6, x7



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和指令中常量之和

- Loop: beq x19, x10, End

add x18, x18, x10

addi x19, x19, -1

j Loop

