Lecture 05 指令系统OK

主要内容：

指令分类、寻址方式、指令格式、指令编码、RISC-V 基本指令集

重点内容：

指令格式、寻址方式、RISC-V 基本指令集

参考教材：第 2 章

一、指令分类

指令：控制计算机完成一些基本操作的命令

机器指令：0、1 形式的指令

汇编指令：文字符号形式的指令

指令系统：所有指令的集合

机器语言：所有机器指令的集合

汇编语言：所有汇编指令的集合

1.按功能分类

数据运算类指令：算术、逻辑、移位、扩展

算术运算类：加、减、乘、除

逻辑运算类：与、或、非、异或、同或等

移位运算类：左移、右移等；算术移位、逻辑移位等

扩展运算类：无符号扩展、有符号扩展

数据传送类指令：常数→寄存器、常数→存储器、寄存器←→寄存器、寄存器←→存储器

流程控制类指令：无条件转移、有条件转移、调用与返回

其它

2.按长度分类

单字长指令、双字长指令、三字长指令等

本课程介绍的 RISC-V 基本指令集的所有指令全部为 32 位长度！

3.按操作数

无操作数、单操作数、双操作数、三操作数等

操作数：数据处理指令中的数据、流程控制指令中的地址！

操作数的类型：常量、变量

变量：寄存器变量、内存变量

源操作数：数据来源（可以是常量、变量）

目的操作数：数据去向（一定是变量）

二、寻址方式

问题：数据处理指令中的数据从哪来，到哪去？流程控制指令中的目标地址从哪来？

寻址方式：指令执行过程中根据有关信息寻找操作数的方式

为此：指令有责任提供操作数的位置信息以及根据位置信息寻找操作数的方式

1.立即数（常量）

指令中直接写出立即数本身。立即数本身就是指令的一部分，当指令从存储器取到 CPU 中后，立即数就可以直接从指令中找到。此种寻址方式称为**立即寻址**。

例如：add EAX,**-1000**（Intel X86）

addi $s0, $s0, **-4**（MIPS）

2.寄存器型操作数（寄存器变量）

指令中直接写出操作数所在的寄存器的编号（或名字），此种寻址方式称为**寄存器寻址**。

例如：add **EAX**,-1000（Intel X86）

addi **$s0**, **$s0**, -4（MIPS）

3.存储器型操作数（内存变量）

当操作数位于内存储器中时，指令中必须提

供其所在的位置信息，即**地址信息**，以及**地址计算方式**——根据地址信息计算地址的方式。

例如：add EAX,**X-1000**（Intel X86）

add EAX,**[EBX]**（Intel X86）

add EAX,**[EBX-1000]**（Intel X86）

add EAX,**[EBX] [ESI]**（Intel X86）

add EAX,**[EBX] [ESI-1000]**（Intel X86）

lw $s0, **0($sp)**（MIPS）

由于不同**程序结构**和**数据结构**的需要，内存地址的提供方式可以有多种。

说明：为了说明指令的寻址方式，可以在指令中设置专门的寻址方式字段；也可以将寻址方式问题和操作问题合并处理，即操作码兼有寻址方式的作用！

4.RISC-V 的寻址方式

RISC-V 支持 4 种寻址方式，前 3 种用于**数据寻址**，第 4 种用于**指令寻址**。

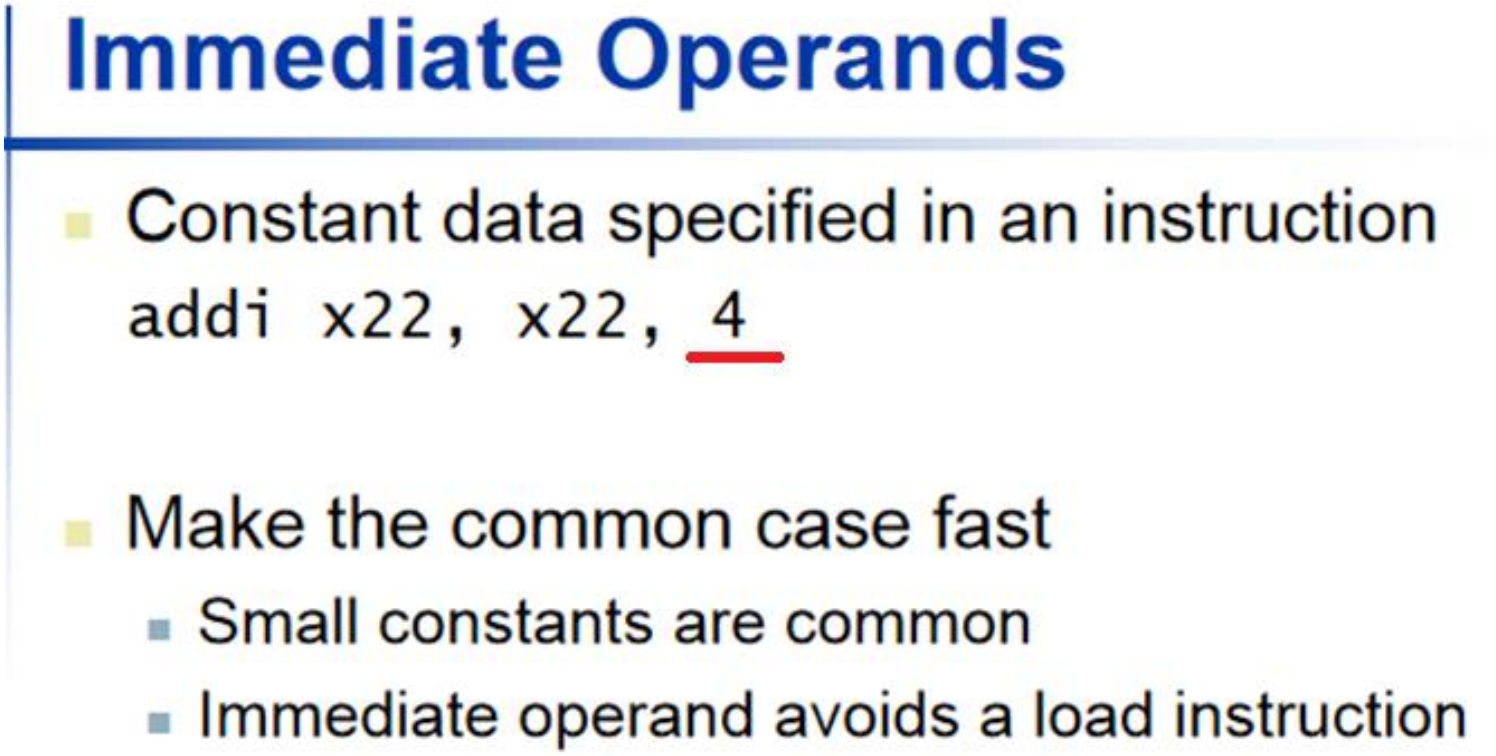
指令寻址：解决程序执行过程中，下一条指令的地址问题，分**顺序寻址**和**跳跃寻址**

顺序寻址：用于顺序结构，程序顺序执行

跳跃寻址：用于分支结构、循环结构以及子程

序调用与返回等，程序从当前位置转移到目标位置执行，用于流程控制指令。

1）立即寻址：用于常量（立即数）



由于指令长度的限制，RISC-V 指令支持的立即数一般不能太大。

问题：如果需要用到比较大的立即数怎么办？

办法：可以利用多条指令进行“拼接”处理，或者：利用**内存“常量”**

2）寄存器寻址：用于寄存器变量

RISC-V 提供了 32 个 64 位的寄存器

Register Operands

- Arithmetic instructions use register operands
- RISC-V has a 32 × 64-bit register file
  - Use for frequently accessed data
  - 64-bit data is called a “doubleword”
    - 32 x 64-bit general purpose registers x0 to x30
  - 32-bit data is called a “word”
- *Design Principle 2:* Smaller is faster
  - c.f. main memory: millions of locations

RISC-V Registers

- x0: the constant value 0
- x1: return address
- x2: stack pointer
- x3: global pointer
- x4: thread pointer
- x5 – x7, x28 – x31: temporaries
- x8: frame pointer
- x9, x18 – x27: saved registers
- x10 – x11: function arguments/results
- x12 – x17: function arguments

Register Operands

- Constant data specified in an instruction

```
addi x22, x22, 4
```

- Make the common case fast
  - Small constants are common
  - Immediate operand avoids a load instruction

3）基址寻址：用于内存变量

内存地址=基址+偏移量

基址：目标数据所在区域的起始地址，由**基址寄存器**提供，

偏移量：目标地址与基址的差值，由立即数提供，采用补码

指令中需要说明**基址寄存器**和**偏移量**！

RISC-V 的内存变量可以是字节类型（8 位）、半字类型（16 位）、字类型（32 位）、双字类型（64

位），不要求对齐，指令一般要求对齐。

RISC-V 系统内存**以字节编址**，地址（物理地址）长度为 64 位；采用**小端模式**存储数据

注意：访问内存（取指令、取数据、存数据）需要的物理地址只是起始地址，起始地址以及数据类型（数据长度）决定了要访问的数据。

对齐：物理地址的低 1 位、低 2 位、低 3 位

为 0，分别为半字对齐、字对齐、双字对齐。

对齐的好处：对齐的半字、字、双字可以一次性访问完成。

这种寻址方式只用于内存访问指令，包括 Load 指令和 Store 指令。 jalr x0, 0(x1)

RISC-V 的运算类指令不能对内存变量进行运算。需要对内存变量进行运算时，必须先将它们读

到寄存器，然后做运算，然后将结果写到内存。

Memory Operand Example

- C code:

```
A[12] = h + A[8];
```

  - h in x21, base address of A in x22
- Compiled RISC-V code:
  - Index 8 requires offset of 64
    - 8 bytes per doubleword

```
ld      x9, 64(x22)
add     x9, x21, x9
sd      x9, 96(x22)
```

4）PC-相对寻址：用于流程控制

PC：**程序计数器**，专门负责为取指令提供地址。

在程序执行过程中，PC 会不断修改。程序顺序执行时，用物理顺序上的下一条指令的地址去修改 PC；碰到流程控制指令时，如果条件成立，用目标指令的地址（**目标地址**）去修改 PC；否则，还是用物理顺序上的下一条指令的地址去修改 PC。

流程控制指令有责任提供目标地址信息，并且在执行时会利用地址信息计算出目标地址，条件成立时，用计算出来的目标地址去修改 PC。

目标地址=当前地址+偏移量*2

当前地址：PC 当前值，即当前正在执行的流程控制指令的地址

字节地址

偏移量：（目标地址-当前地址）/2，补码

半字

字节地址：以字节为单位的地址

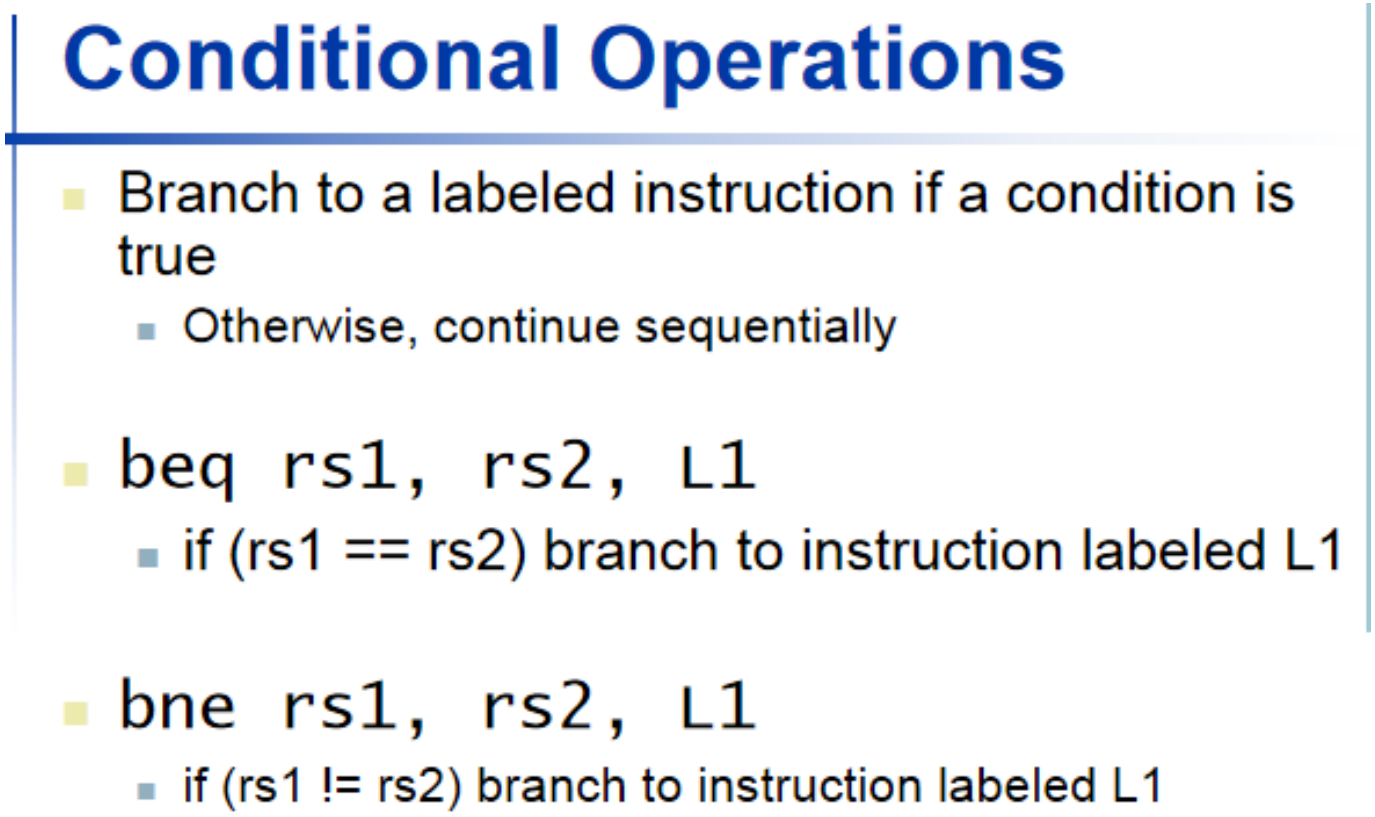
半字地址：以半字为单位的地址

字地址：以字为单位的地址

双字地址：以双字为单位的地址

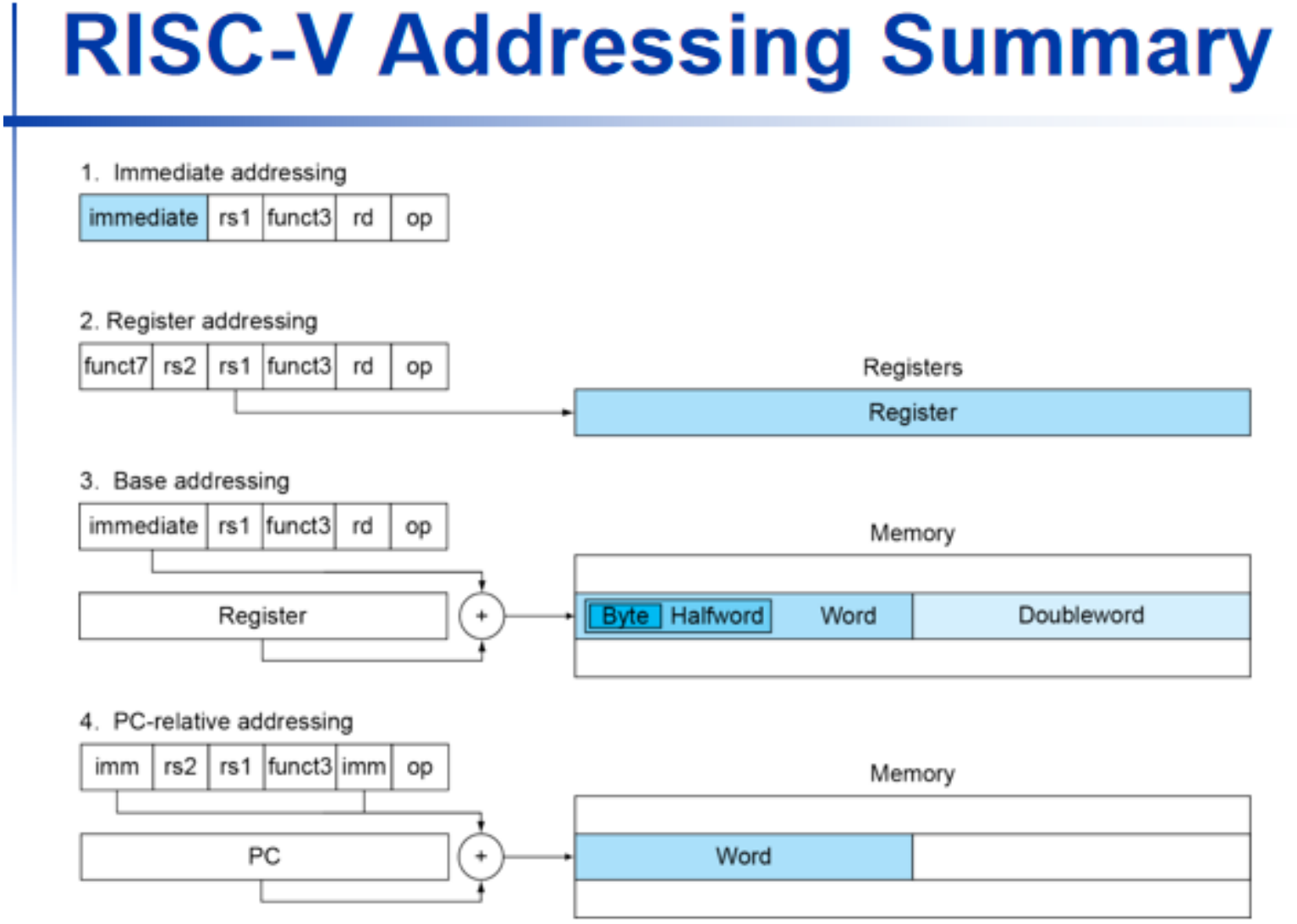
访问内存需要的是字节地址！

流程控制指令有责任提供偏移量（以半字为单位）。



标号 L1 表示目标地址，翻译转换时，【目标地址-当前地址（beq 指令的地址）】/2 就是对应的机器指令中的偏移量，由翻译转换程序（汇编器）自

动计算。


RISC-V Addressing Summary
1. Immediate addressing
immediate
rs1
funct3
rd
op
2. Register addressing
funct7
rs2
rs1
funct3
rd
op
Registers
Register
3. Base addressing
immediate
rs1
funct3
rd
op
Memory
Register
+
Byte
Halfword
Word
Doubleword
4. PC-relative addressing
imm
rs2
rs1
funct3
imm
op
Memory
PC
+
Word

三、指令格式

1.指令构成

完成指令功能所需要的各种信息：**操作码、操作数、地址码、寻址方式**等

2.指令格式

完成指令功能所需要的各种信息在指令中的位置和宽度

3.基本格式

操作码 操作数/地址码

操作码 寻址方式 操作数/地址码

基本操作码 扩展操作码 操作数/地址码

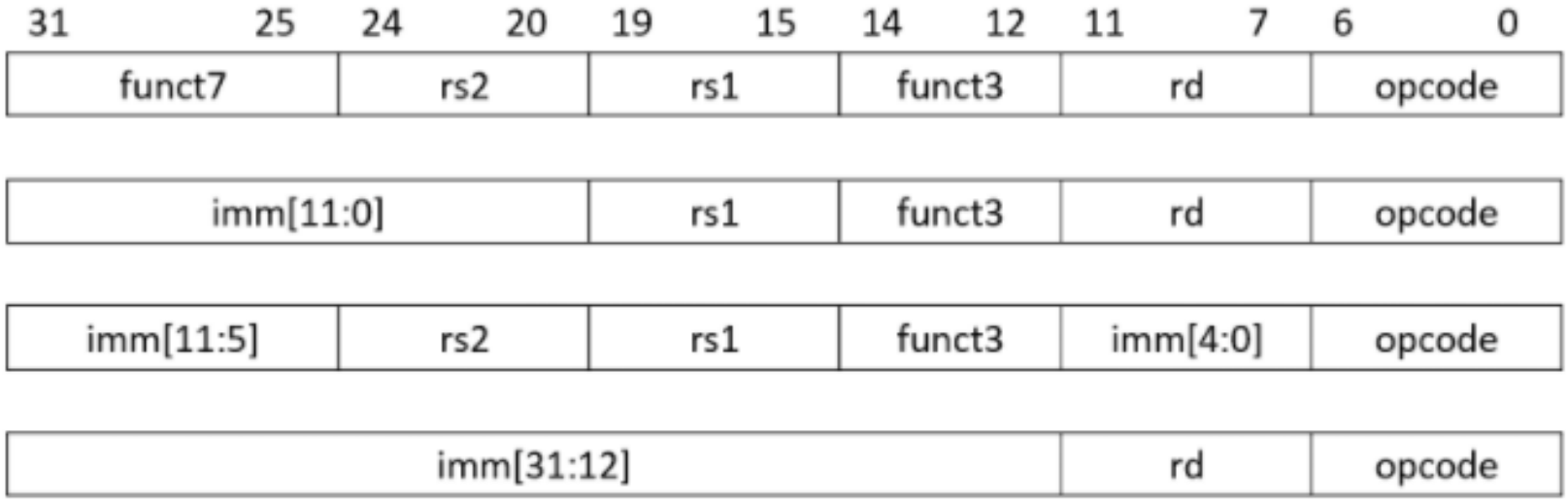
基本操作码 扩展操作码 寻址方式 操作数/地址码

特别提示：不同系统的 ISA 有不同处理方式，

具体的指令格式通常和指令的功能有关以及操作数的个数、寻址方式等有关！

4.RISC-V 指令格式

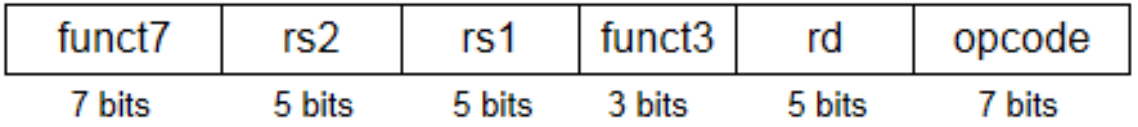
RISC-V 基本指令集有以下几种格式。



1）R 型格式：用于运算类指令，支持寄存器和寄存器之间的算术、逻辑、移位运算

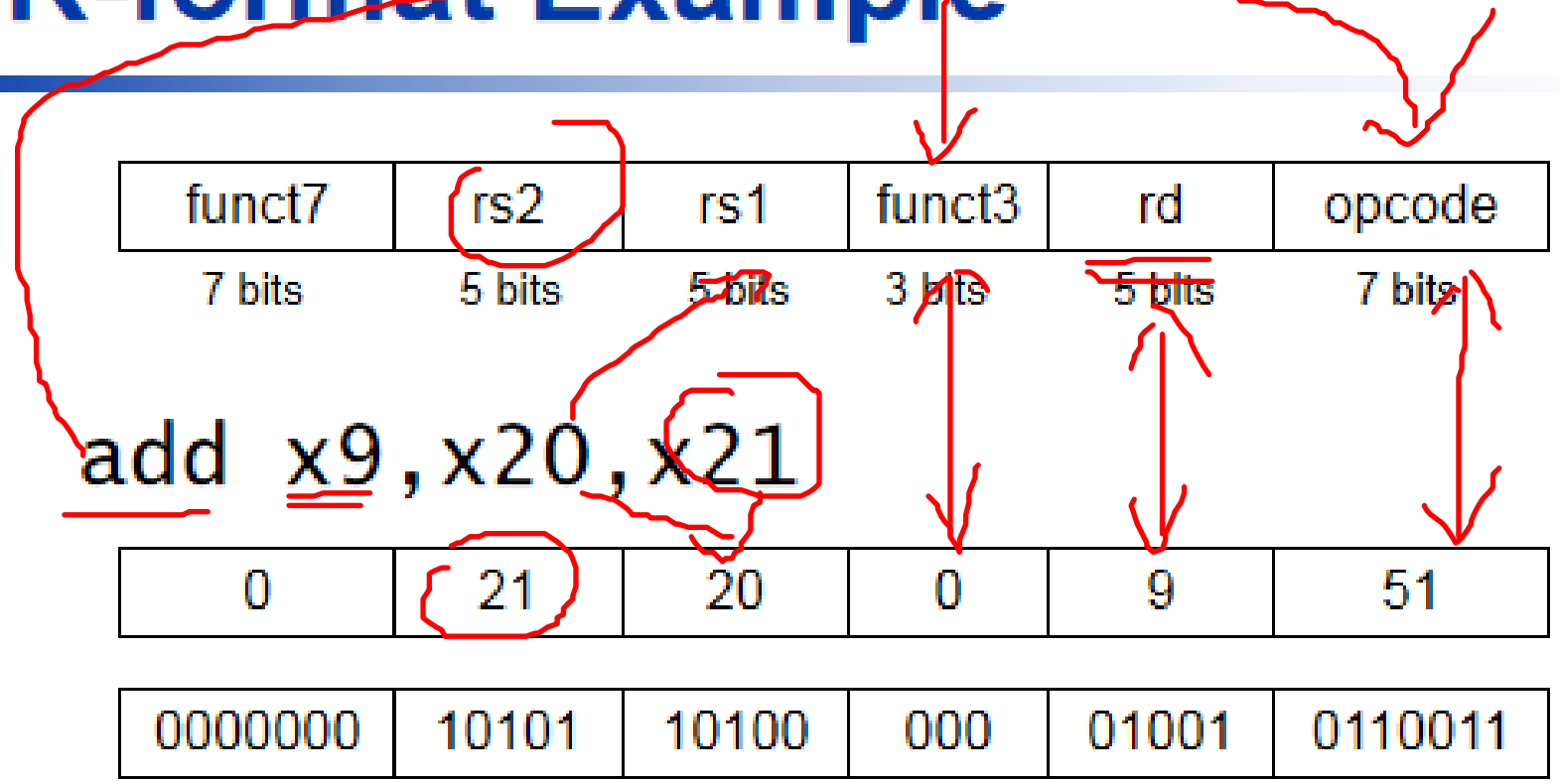
add/and/sll rd, rs1, rs2

RISC-V R-format Instructions



- Instruction fields
  - opcode: operation code
  - rd: destination register number
  - funct3: 3-bit function code (additional opcode)
  - rs1: the first source register number
  - rs2: the second source register number
  - funct7: 7-bit function code (additional opcode)

R-format Example



funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

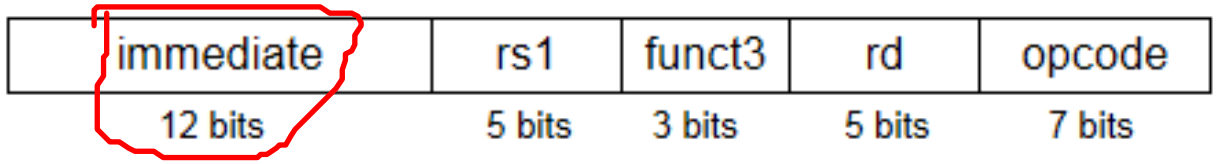
add x9,x20,x21

0	21	20	0	9	51
0000000	10101	10100	000	01001	0110011

$0000\ 0001\ 0101\ 1010\ 0000\ 0100\ 1011\ 0011_{two} =$
$015A04B3_{16}$

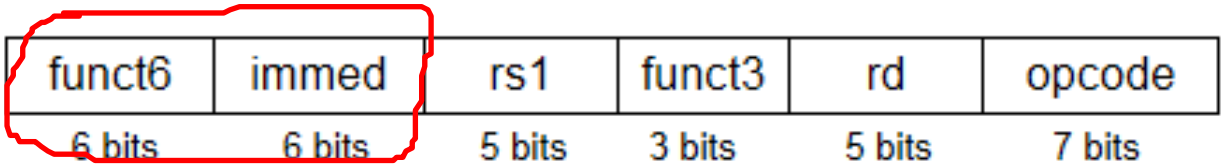
2）I 型格式：用于运算类指令和 Load 指令，支持常数和寄存器之间的算术、逻辑、移位运算

RISC-V I-format Instructions



- Immediate arithmetic and load instructions
  - rs1: source or base address register number
  - immediate: constant operand, or offset added to base address
    - 2s-complement, sign extended
- *Design Principle 3:* Good design demands good compromises
  - Different formats complicate decoding, but allow 32-bit instructions uniformly
  - Keep formats as similar as possible

Shift Operations

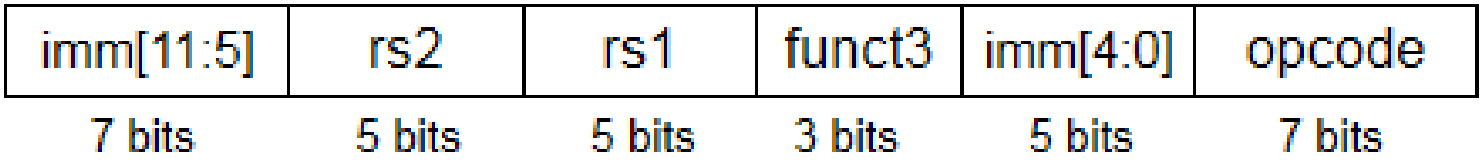


- immed: how many positions to shift
- Shift left logical
  - Shift left and fill with 0 bits
  - `slli` by *i* bits multiplies by 2^i
- Shift right logical
  - Shift right and fill with 0 bits
  - `srli` by *i* bits divides by 2^i (unsigned only)

slli rd, rs1, imm

3）S 型格式：用于 Store 指令

RISC-V S-format Instructions



- Different immediate format for store instructions
  - rs1: base address register number
  - rs2: source operand register number
  - immediate: offset added to base address
    - Split so that rs1 and rs2 fields always in the same place

sw rs2, imm(rs1)

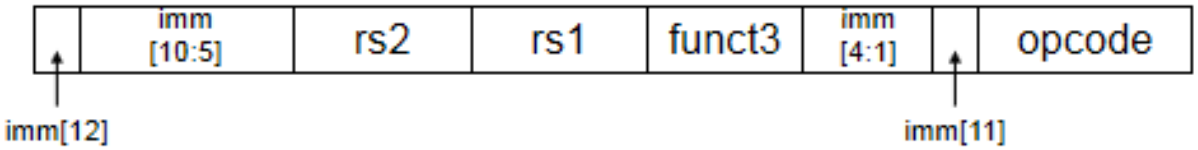
offset

SB 型格式: 用于分支指令(有条件转移指令；条件恒成立时相当于无条件转移)

Branch Addressing

beq rs1, rs2, imm (offset)

- Branch instructions specify
  - Opcode, two registers, target address
- Most branch targets are near branch
  - Forward or backward
- SB format:



- PC-relative addressing
  - Target address = PC + immediate × 2

4）U 型格式：用于大常量处理指令 lui

20 bits

imm[31:12]	rd	opcode

lui rd, imm

32-bit Constants

- Most constants are small
  - 12-bit immediate is sufficient
- For the occasional 32-bit constant

```
lui rd, constant
```

  - Copies 20-bit constant to bits [31:12] of rd
  - Extends bit 31 to bits [63:32]
  - Clears bits [11:0] of rd to 0

```
lui x19, 976  // 0x003D0
```

0 0000 0000 0000	0000 0000 0000 0000	0000 0000 0011 1101 0000	0000 0000 0000

```
addi x19,x19,128  // 0x500
```

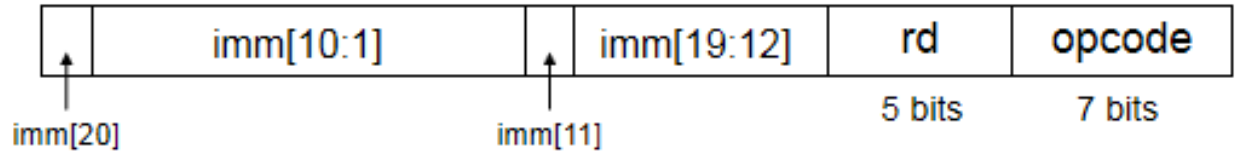
0 0000 0000 0000	0000 0000 0000 0000	0000 0000 0011 1101 0000	0101 0000 0000

x19

UJ 型格式：用于无条件转移指令

Jump Addressing

- Jump and link (jal) target uses 20-bit immediate for larger range
- UJ format:



- For long jumps, eg, to 32-bit absolute address
  - lui: load address[31:12] to temp register
  - jalr: add address[11:0] and jump to target

jal rd, imm (offset) 20bits

RISC-V Format Summary

Name (Field Size)	Field 7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	Comments
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
UJ-type	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
U-type	immediate[31:12]				rd	opcode	Upper immediate format

特别提示：汇编指令的格式和机器指令的格式大体一致，但未必严格按顺序一一对应！

四、指令编码

1.“编码”的两种含义

名词：指令的二进制形式

动词：将指令中的各种信息用恰当的二进制形式表示出来，设计指令系统时就要做

说明：指令编码主要是指指令中的操作码、寻址方式信息等无法**参数**化的部分，即相对固定的

部分，尤其是操作码！

比如，如果设计这样的一条加法指令：将某个寄存器的内容和另一个寄存器的内容做加法，结果放到第 3 个寄存器

那么，其中的寄存器信息就是可以参数化的部分，而其它的信息就必须固定下来！

2.RISC-V 基本指令集指令编码举例

格式	指令	操作码	Funct3	Funct6/7
R型	add	0110011	000	0000000
	sub	0110011	000	0100000
	sll	0110011	001	0000000
	xor	0110011	100	0000000
	srl	0110011	101	0000000
	sra	0110011	101	0000000
	or	0110011	110	0000000
	and	0110011	111	0000000
	lr.d	0110011	011	0001000
	sc.d	0110011	011	0001100
I型	lb	0000011	000	n.a.
	lh	0000011	001	n.a.
	lw	0000011	010	n.a.
	ld	0000011	011	n.a.
	lbu	0000011	100	n.a.
	lhu	0000011	101	n.a.
	lwu	0000011	110	n.a.
	addi	0010011	000	n.a.
	slli	0010011	001	000000
	xori	0010011	100	n.a.
	srli	0010011	101	000000
	srai	0010011	101	010000
	ori	0010011	110	n.a.
	andi	0010011	111	n.a.
	jalr	1100111	000	n.a.
S型	sb	0100011	000	n.a.
	sh	0100011	001	n.a.
	sw	0100011	010	n.a.
	sd	0100011	111	n.a.
SB型	beq	1100111	000	n.a.
	bne	1100111	001	n.a.
	blt	1100111	100	n.a.
	bge	1100111	101	n.a.
	bltu	1100111	110	n.a.
	bgeu	1100111	111	n.a.
U型	lui	0110111	n.a.	n.a.
UJ型		1101111	n.a.	n.a.

RISC-V Registers

- x0: the constant value 0
- x1: return address
- x2: stack pointer
- x3: global pointer
- x4: thread pointer
- x5 – x7, x28 – x31: temporaries
- x8: frame pointer
- x9, x18 – x27: saved registers
- x10 – x11: function arguments/results
- x12 – x17: function arguments

RISC-V Format Summary

Name (Field Size)	Field 7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	Comments
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
UJ-type	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
U-type	immediate[31:12]				rd	opcode	Upper immediate format

1）R 型格式

加法指令：add rd, rs1, rs2

funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

add x9,x20,x21

0	21	20	0	9	51

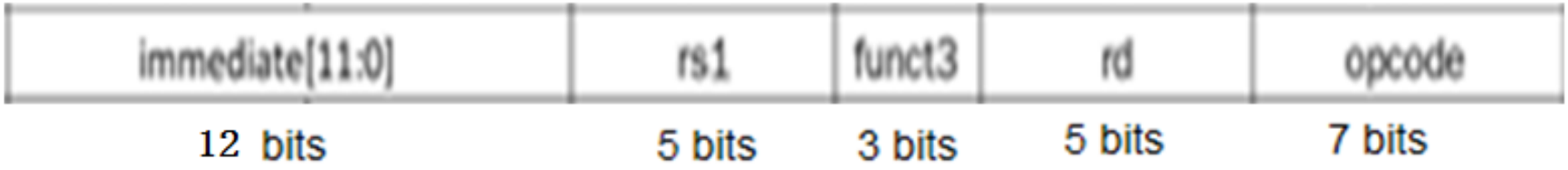
0000000	10101	10100	000	01001	0110011

$0000\ 0001\ 0101\ 1010\ 0000\ 0100\ 1011\ 0011_{two} =$

$015A04B3_{16}$

2）I 型格式

立即数加法指令： addi rd, rs1, imm



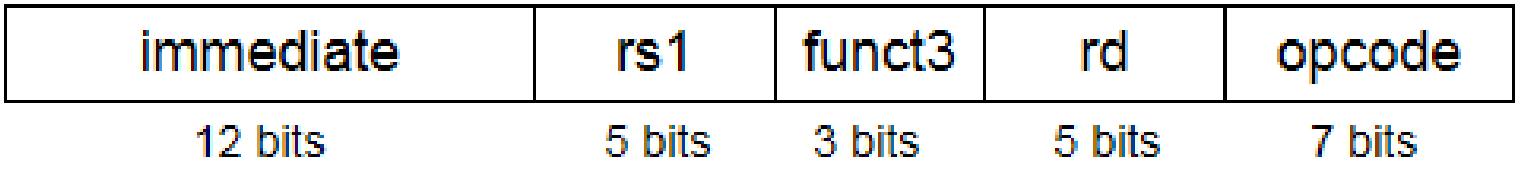
addi x9, x20, −2000　　0x830

−2000	20	0	9	19
100000110000	10100	000	01001	0010011

8 3 0

$1000\ 0011\ 0000\ 1010\ 0000\ 0100\ 1001\ 0011_{two} = 830A0493_{16}$

加载指令： lw rd, imm(rs1)



lw x9, -100(x20)

-100	20	2	9	3
111110011100	10100	010	01001	0000011

1111 1001 1100 1010 0010 0100 1000 0011

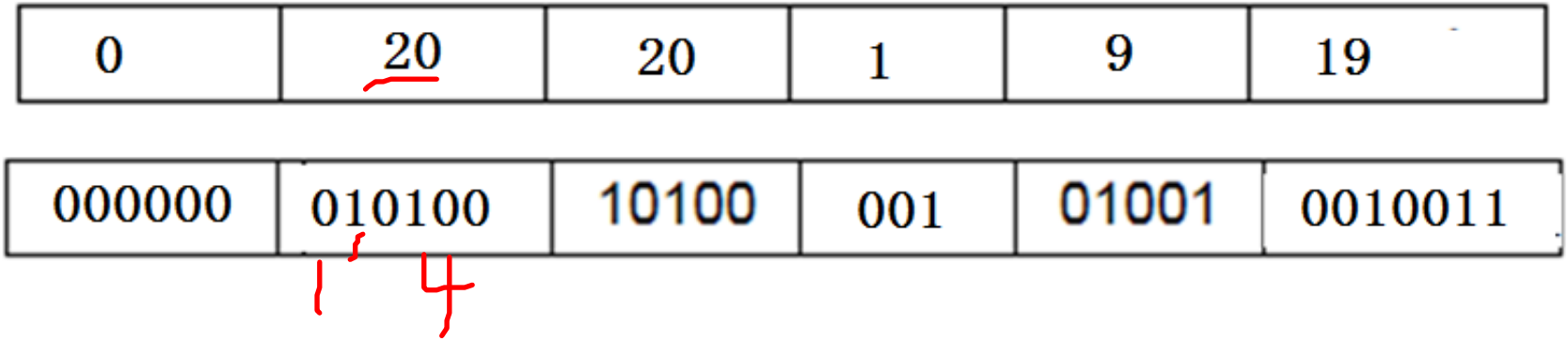
=F9CA2483

立即数移位指令： slli rd, rs1, imm

slli x9,x20, 20

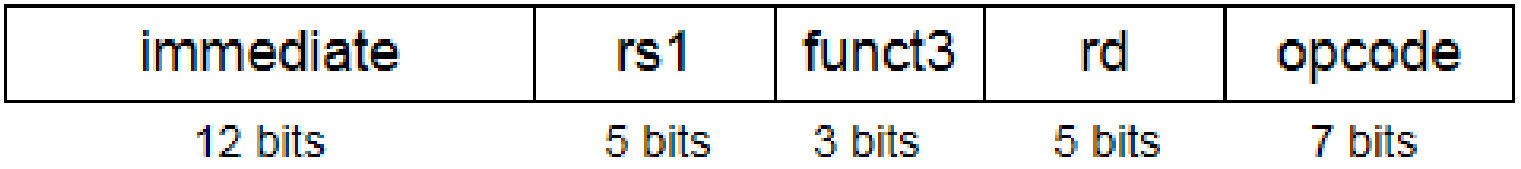
0x14

20



$0000\ 0001\ 0100\ 1010\ 0001\ 0100\ 1001\ 0011_{two} = 014A1493_{16}$

寄存器转移连接指令：jalr rd, imm(rs1)



immediate	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

jalr x0, 0(x1)

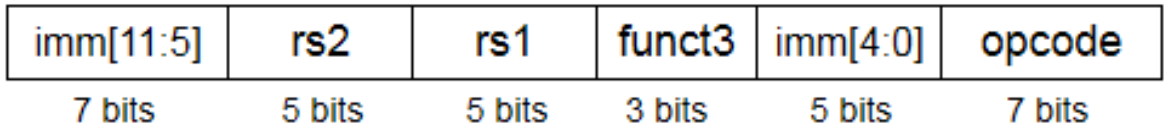
0	1	0	0	103
12 bits	5 bits	3 bits	5 bits	7 bits

000000000000	00001	000	00000	1100111
12 bits	5 bits	3 bits	5 bits	7 bits

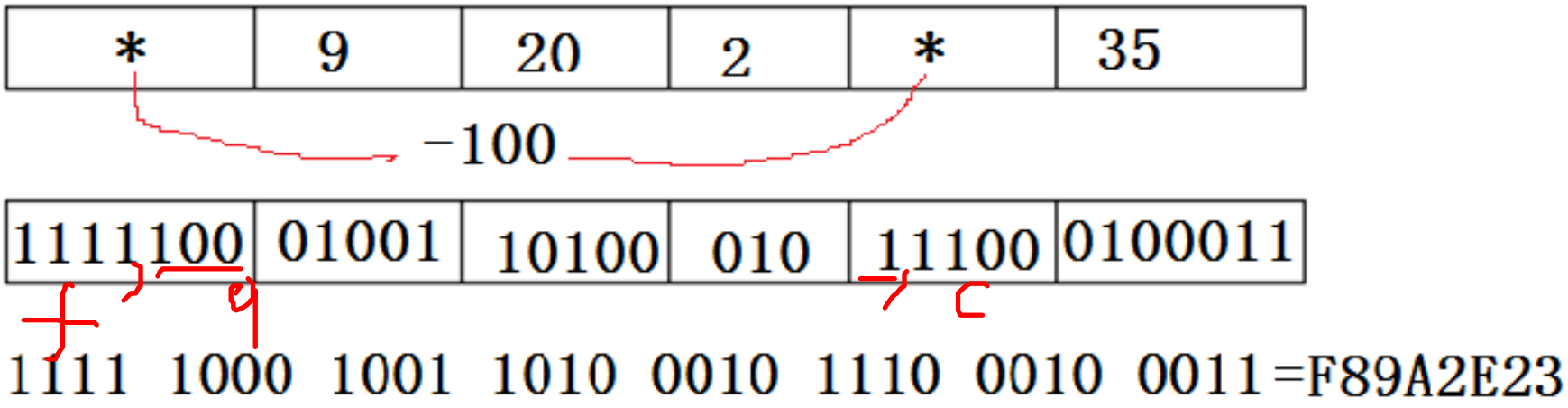
0000 0000 0000 0000 1000 0000 0110 0111=00008067

3）S 型格式

存储指令： sw rs2, imm(rs1)

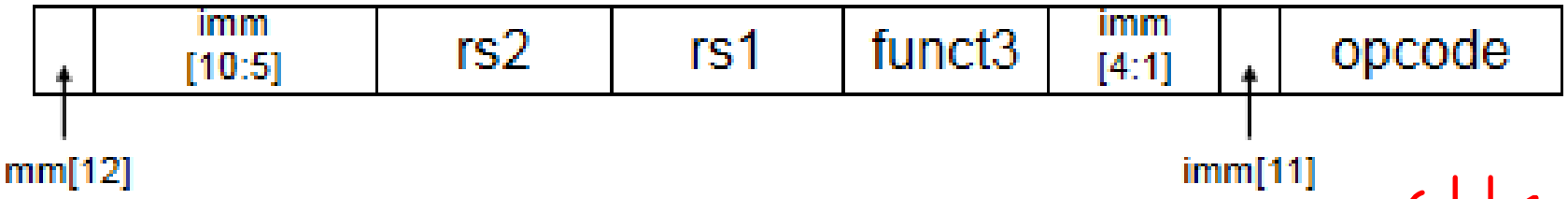


sw x9, -100(x20)



SB 型格式

分支指令：beq rs1, rs2, 标号



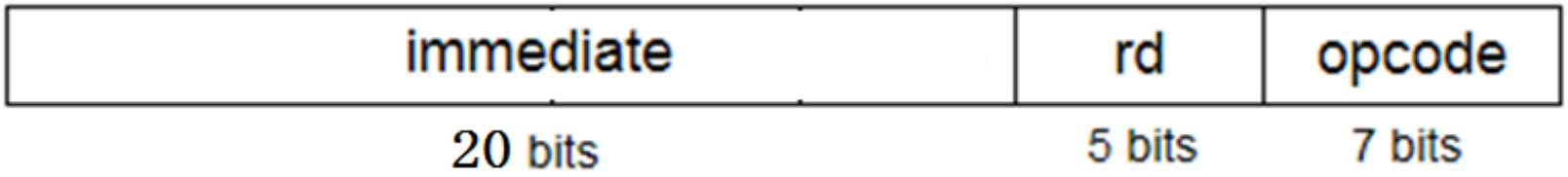
beq x22, x23, label1（设偏移量为-100）

1	57	23	22	0	12	1	103
1	111001	10111	10110	000	1100	1	1100111

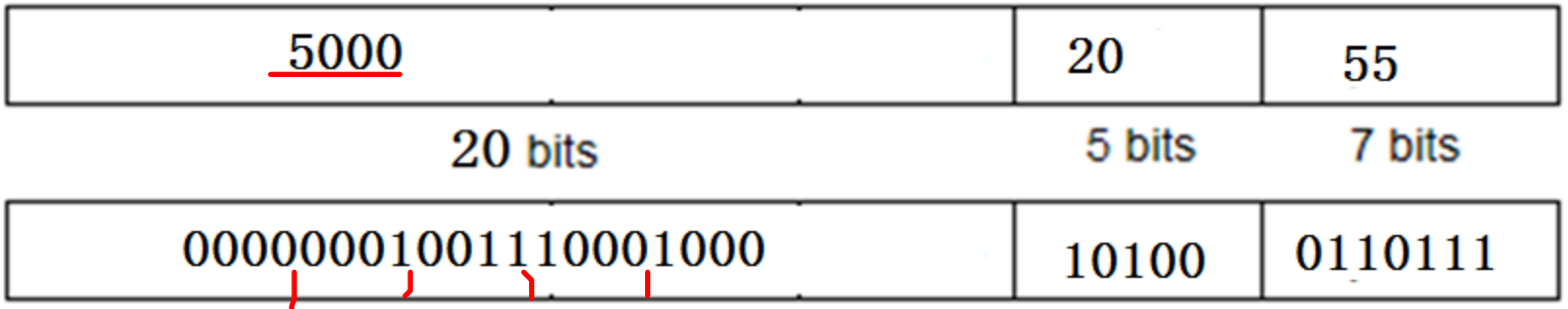
1111 0011 0111 1011 0000 1100 1110 0111=F37B0CE7

4）U 型格式

立即数加载指令： lui rd, imm



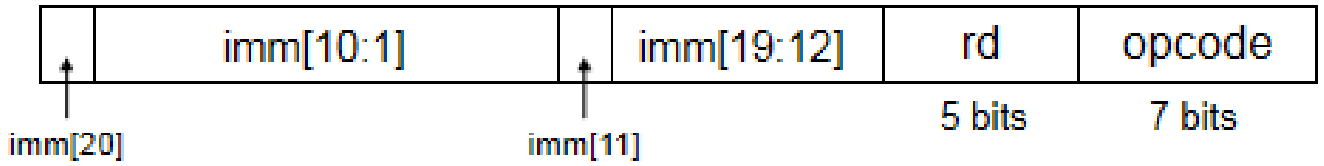
lui x20, 5000



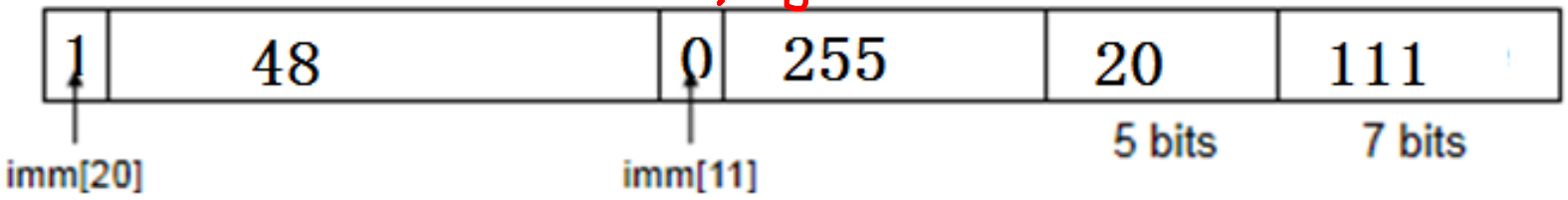
0000 0001 0011 1000 1000 1010 0011 0111=01388A37

UJ 型格式

转移连接指令： jal rd, label



Jal x20, label（设偏移量为-2000）



1	0000110000	0	11111111	10100	1101111

1000 0110 0000 1111 1111 1010 0110 1111=860FFA6F

五、RISC-V 基本指令集

RISC-V 基本体系结构主要包括以下几类指令。

运算类指令：操作数可以是常数、寄存器变量，不能是存储器变量（无需访问内存）；采用 I 型格式和 R 型格式；采用立即寻址和寄存器寻址。

支持寄存器和寄存器、常数和寄存器之间的运算，结果仍放到寄存器。

传送类指令：专门负责在寄存器和内存之间传送数据；数据长度可以是 8、16、32、64 位；采用 I 型格式；采用寄存器寻址和基址寻址。

流程控制类指令：无条件转移指令采用 I 型和 UJ 型格式，基址寻址和 PC-相对寻址；分支指令采用 SB 型格式，PC-相对寻址

比较类指令：负责对两个操作数进行比较，并且根据比较结果将**目的寄存器**置 1 或清零，从而

建立条件码，供分支指令使用；可以是寄存器和寄存器作比较，也可以是立即数和寄存器做比较；采用 R 型和 I 型格式，立即寻址和寄存器寻址

以下是 RISC-V 基本体系结构的指令，共 51 条，用于整数运算。

后面加u的表示unsigned

RISC-V 操作数

名字	示例	注解
32个寄存器	x0～x31	快速定位数据。在RISC-V中，只对在寄存器中的数据执行算运算
2^{61}个存储字	Memory[0], Memory[8], ..., Memory[18 446 744 073 709 551 608]	只能被数据传输指令访问。RISC-V使用字节寻址，因此顺序字访问相差8。存储器保存数据结构、数组和换出的寄存器的内

RISC-V 汇编语言

类别	指令	示例	含义	注解
算术运算	加	`add x5, x6, x7`	`x5 = x6 + x7`	三寄存器操作数；加
	减	`sub x5, x6, x7`	`x5 = x6 - x7`	三寄存器操作数；减
	立即数加	`addi x5, x6, 20`	`x5 = x6 + 20`	用于加常数
数据传输	取双字	`ld x5, 40(x6)`	`x5 = Memory[x6 + 40]`	从存储器取双字到寄存器
	存双字	`sd x5, 40(x6)`	`Memory[x6 + 40] = x5`	从寄存器存双字到存储器
	取字	`lw x5, 40(x6)`	`x5 = Memory[x6 + 40]`	从存储器取字到寄存器
	取字（无符号数）	`lwu x5, 40(x6)`	`x5 = Memory[x6 + 40]`	从存储器取无符号字到寄存器
	存字	`sw x5, 40(x6)`	`Memory[x6 + 40] = x5`	从寄存器存字到存储器
	取半字	`lh x5, 40(x6)`	`x5 = Memory[x6 + 40]`	从存储器取半字到寄存器
	取半字（无符号数）	`lhu x5, 40(x6)`	`x5 = Memory[x6 + 40]`	从存储器取无符号半字到寄存器

数据传输	存半字	sh x5, 40(x6)	Memory[x6 + 40] = x5	从寄存器存半字到存储器
	取字节	lb x5, 40(x6)	x5 = Memory[x6 + 40]	从存储器取字节到寄存器
	取字节（无符号数）	lbu x5, 40(x6)	x5 = Memory[x6 + 40]	从存储器取无符号字节到寄存器
	存字节	sb x5, 40(x6)	Memory[x6 + 40] = x5	从寄存器存字节到存储器
	取保留字	lr.d x5, (x6)	x5 = Memory[x6]	取；原子交换的前半部分
	存条件字	sc.d x7, x5, (x6)	Memory[x6] = x5; x7 = 0/1	存；原子交换的后半部分
	取立即数高位	lui x5, 0x12345	x5 = 0x12345000	取左移12位后的20位立即数
逻辑运算	与	and x5, x6, x7	x5 = x6 & x7	三寄存器操作数；按位与
	或	or x5, x6, x8	x5 = x6 | x8	三寄存器操作数；按位或
	异或	xor x5, x6, x9	x5 = x6 ^ x9	三寄存器操作数；按位异或
	与立即数	andi x5, x6, 20	x5 = x6 & 20	寄存器与常数按位与
	或立即数	ori x5, x6, 20	x5 = x6 | 20	寄存器与常数按位或
	异或立即数	xori x5, x6, 20	x5 = x6 ^ 20	寄存器与常数按位异或
移位操作	逻辑左移	sll x5, x6, x7	x5 = x6 << x7	按寄存器给定位数左移
	逻辑右移	srl x5, x6, x7	x5 = x6 >> x7	按寄存器给定位数右移
	算术右移	sra x5, x6, x7	x5 = x6 >> x7	按寄存器给定位数算术右移
	逻辑左移立即数	slli x5, x6, 3	x5 = x6 << 3	根据立即数给定位数左移
	逻辑右移立即数	srli x5, x6, 3	x5 = x6 >> 3	根据立即数给定位数右移
	算术右移立即数	srai x5, x6, 3	x5 = x6 >> 3	根据立即数给定位数算术右移
条件分支	相等即跳转	beq x5, x6, 100	if (x5 == x6) go to PC+100	若寄存器数值相等则跳转到PC相对地址
	不等即跳转	bne x5, x6, 100	if (x5 != x6) go to PC+100	若寄存器数值不等则跳转到PC相对地址
	小于即跳转	blt x5, x6, 100	if (x5 < x6) go to PC+100	若寄存器数值比较结果小于则跳转到PC相对地址
	大于等于即跳转	bge x5, x6, 100	if (x5 >= x6) go to PC+100	若寄存器数值比较结果大于或等于则跳转到PC相对地址
	小于即跳转（无符号）	bltu x5, x6, 100	if (x5 < x6) go to PC+100	若寄存器数值比较结果小于则跳转到PC相对地址（无符号）
	大于等于即跳转（无符号）	bgeu x5, x6, 100	if (x5 >= x6) go to PC+100	若寄存器数值比较结果大于或等于则跳转到PC相对地址（无符号）
无条件跳转	跳转-链接	jal x1, 100	x1 = PC+4; go to PC+100	用于PC相关的过程调用
	跳转-链接（寄存器地址）	jalr x1, 100(x5)	x1 = PC+4; go to x5+100	用于过程返回；非直接调用

RISC-V基本体系结构的特有指令

指令	名称	格式	描述
Add upper immediate to PC	auipc	U	立即数高20位与PC相加；将结果写到寄存器
Set if less than	slt	R	比较寄存器；将布尔结果写到寄存器
Set if less than, unsigned	sltu	R	比较寄存器；将布尔结果写到寄存器
Set if less than, immediate	slti	I	比较寄存器；将布尔结果写到寄存器
Set if less than immediate, unsigned	sltiu	I	比较寄存器；将布尔结果写到寄存器
Add word	addw	R	32位数字相加
Subtract word	subw	R	32位数字相减
Add word immediate	addiw	I	常数与32位数字相加
Shift left logical word	sllw	R	根据寄存器左移32位数字
Shift right logical word	srlw	R	根据寄存器右移32位数字
Shift right arithmetic word	sraw	R	根据寄存器算术右移32位数字
Shift left logical word immedate	slliw	I	根据立即数左移32位数字
Shift right logical word immediate	srliw	I	根据立即数右移32位数字
Shift right arithmetic word immediate	sraiw	I	根据立即数算术右移32位数字

RISC-V基本体系结构与扩展

助记符	描述	指令数
I	基本体系结构	51
M	整数乘法/除法	13
A	原子操作	22
F	单精度浮点	30
D	双精度浮点	32
C	压缩指令	36

图 2-38　RISC-V 指令系统体系结构分为基本 ISA（称作 I）以及五个标准扩展：M、A、F、D 和 C

例题：设有如下程序段*OK*

```
        add x1, x0, x0          （第 1 条指令）
        addi x1, x1, -150       x1=-150
        sub x2, x0, x0
label1: addi x2, x2, 1000       x2=1000
        lw x3, -100(x2)         x3=67305985
        sub x2, x2, x1          x2=1150
```

```
        slt x3, x1, x2                  x3=1
        beq x3, x0, label1        false
        slli x3, x3, 10
        sw x3, -100(x2)
        jal x10, label2
        ……（省略 5 条指令）
Label2: add x2, x2, x3      （最后一条指令）
```

……

执行时，从地址0*x*0000 0000 7000 0000开始连续存放全部指令。

此时，有关内存状态如下：

16 进制地址（64 比特）	10 进制地址	单元内容（机器数）
⋮	⋮	⋮
0000 0000 0000 0383	899	0*x*00

0000 0000 0000 0384	900	0*x*01
0000 0000 0000 0385	901	0*x*02
0000 0000 0000 0386	902	0*x*03
0000 0000 0000 0387	903	0*x*04
0000 0000 0000 0388	904	0*x*05
⋮	⋮	

（1）指令 addi x1, x1, -150 执行后，寄存器 x1 的机器数是什么？其真值是什么？此指令的机器

码（机器指令）的 imm 字段是什么？其真值是什么？

参考：x1 的机器数是$0xffff\ ffff\ ffff\ ff6a$，其真值是-150；此指令的机器码的 imm 字段是$0xf6a$，其真值是-150；

（2）标号 label1 所代表的地址是多少？

参考：label1 所代表的地址就是第 4 条指令的

地址。第 1 条指令的地址是$0x0000\ 0000\ 7000\ 0000$

label1 所代表的地址是

$$0x0000\ 0000\ 7000\ 0000 + 3 \times 4$$

$$= 0x0000\ 0000\ 7000\ 000c$$

（3）指令 lw x3, -100(x2)读内存使用的地址是多少？此指令执行后，寄存器 x3 的机器数是什么？其真值是多少？

参考：内存地址=x2-100=1000-100=900

$$=0x0000\ 0000\ 0000\ 03e8+0xffff\ ffff\ ffff\ ff9c$$

$$=0x0000\ 0000\ 0000\ 0384$$

x3 的机器数是$0x04030201$，其真值是67305985

（4）指令 slt x3, x1, x2 执行后，寄存器 x3 的内容是多少？

参考：x1=-150, x2=1150, x1<x2, x3=1

（5）指令 beq x3, x0, label1 的机器码的 imm 字段是多少？其真值是多少？此指令的机器码是多少？

参考：此指令的地址是$0x0000\ 0000\ 7000\ 001c$

（$0x0000\ 0000\ 7000\ 0000 + 7 \times 4$）

label1 表示的地址是$0x0000\ 0000\ 7000\ 000c$

$0x0000\ 0000\ 7000\ 000c - 0x0000\ 0000\ 7000\ 001c = 0xff\ 0$

此指令机器码的 imm 字段是0*xff*8，其真值是−8；此指令的机器码是0*xfe*01 88*e*7

（6）指令 slli x3, x3, 10 执行后，寄存器 x3 的内容是多少？

参考：$1\times 2^{10}=0x400$

（7）指令 sw x3, -100(x2)写内存使用的地址是多少？内存中从此地址开始的连续 4 个字节单元

的内容是多少？用存储示意图说明。

参考：内存地址=x2-100=1150-100=1050

$$= 0x0000\ 0000\ 0000\ 047e + 0xffff\ ffff\ ffff\ ff9c$$

$$= 0x0000\ 0000\ 0000\ 041a$$

16 进制地址（64 比特）	10 进制地址	单元内容（机器数）
⋮	⋮	⋮
0000 0000 0000 041a	1050	0x00

0000 0000 0000 041*b*	1051	0*x*04
0000 0000 0000 041*c*	1052	0*x*00
0000 0000 0000 041*d*	1053	0*x*00
⋮	⋮	

（8）指令 jal x10, label2 中，标号 label2 代表的地址是多少？此指令的机器码中的 imm 字段是多少？此指令执行后，寄存器 x10 的值是多少？

参考：label2 代表的地址是

$0x0000\ 0000\ 7000\ 0000 + 16 \times 4$

$= 0x0000\ 0000\ 7000\ 0040$

此指令的地址是

$0x0000\ 0000\ 7000\ 0000 + 10 \times 4$

$= 0x0000\ 0000\ 7000\ 0028$

$0x0000\ 0000\ 7000\ 0040 - 0x0000\ 0000\ 7000\ 0028 = 0x18$

0x182

此指令的机器码中的 imm 字段是 $0x0000c = 12$

此指令执行后，寄存器 x10 的值是

$$0x0000\ 0000\ 7000\ 0028 + 4 = 0x0000\ 0000\ 7000\ 002c$$