Computer Architecture

3. Instructions: Language of the Machine

Young Geun Kim
(younggeun_kim@korea.ac.kr)
Intelligent Computer Architecture & Systems Lab.

RISC-V: Machine-level Representation

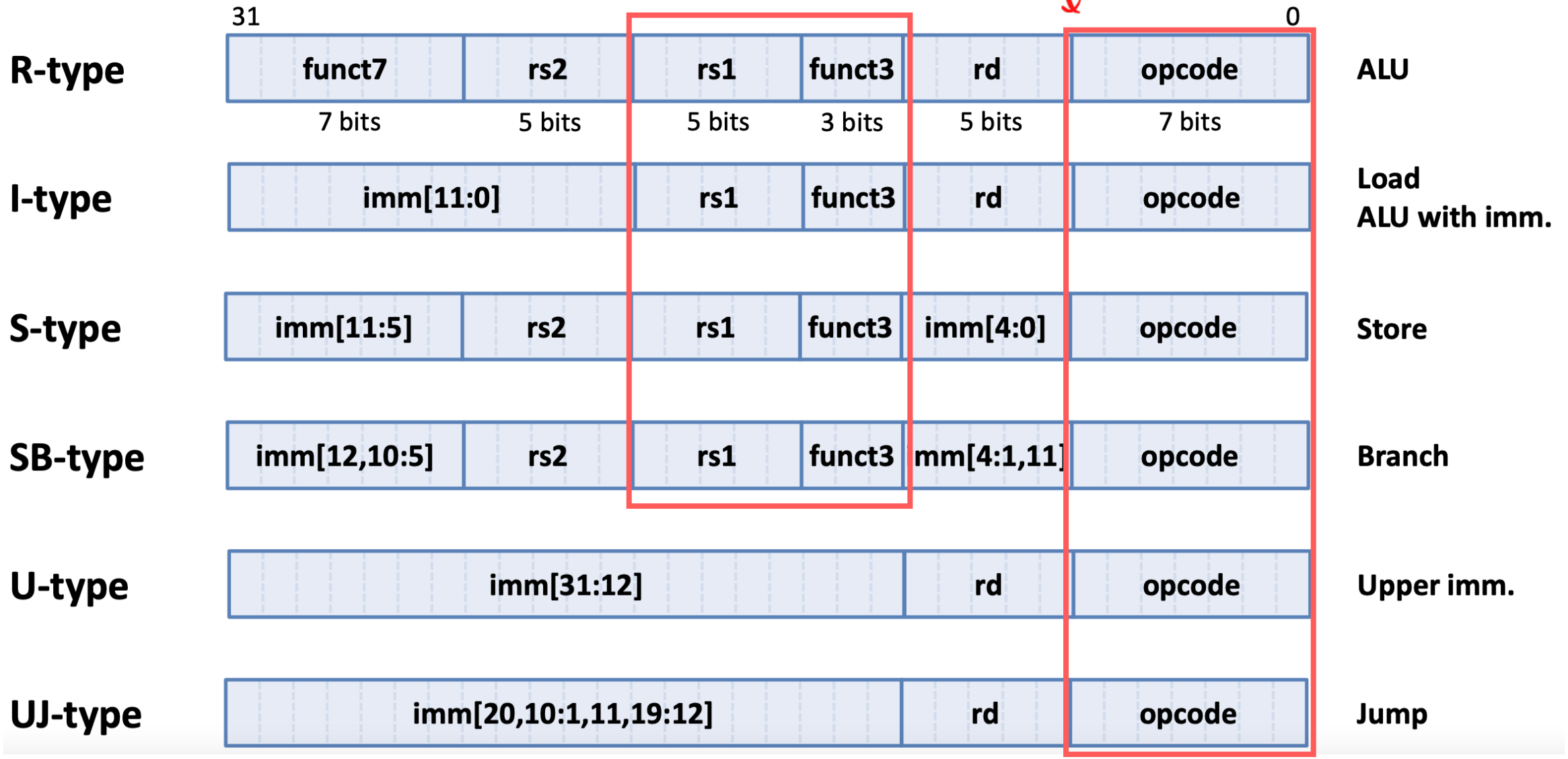
Representing Instructions

- **Instructions are encoded in binary**
  - Called machine code

- **RISC-V instructions**
  - Encoded as 32-bit instruction words
  - Small number of formats encoding operation code (opcode), register numbers, …
  - Regularity!

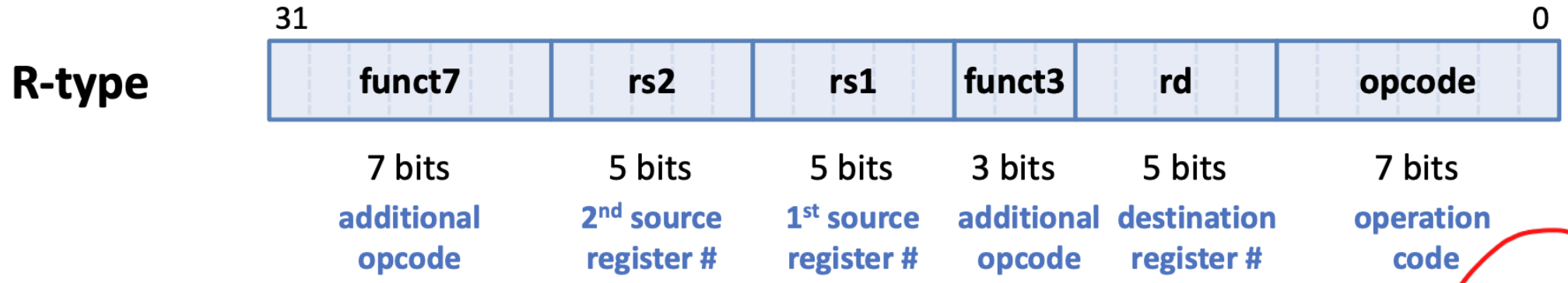
RISC-V Instruction Formats

instruction의 종류를 구별하는 데 사용
→ 항상 같은 위치



RISC-V R-Type Instructions

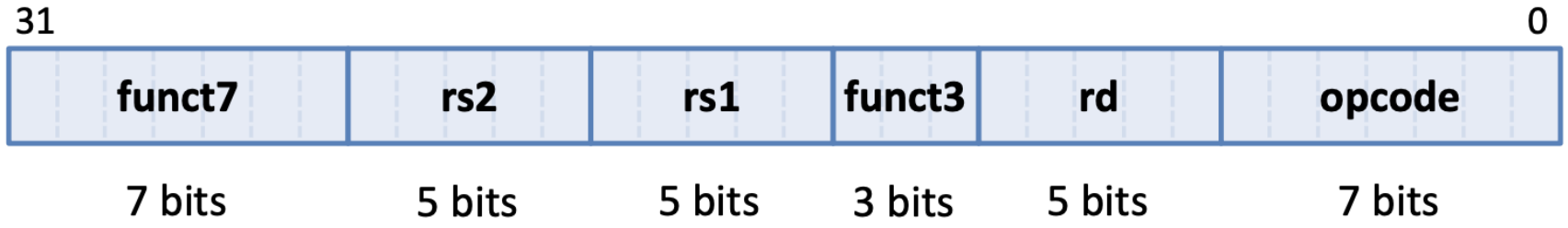
Regular



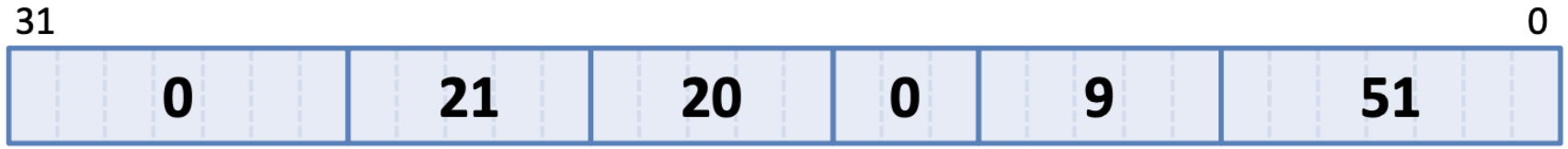
Instruction	Type	Example	funct7	funct3	opcode
add	R	add rd, rs1, rs2	0000000	000	0110011
sub	R	sub rd, rs1, rs2	0100000	000	0110011
sll	R	sll rd, rs1, rs2	0000000	001	0110011
slt	R	slt rd, rs1, rs2	0000000	010	0110011
sltu	R	sltu rd, rs1, rs2	0000000	011	0110011
xor	R	xor rd, rs1, rs2	0000000	100	0110011
srl	R	srl rd, rs1, rs2	0000000	101	0110011
sra	R	sra rd, rs1, rs2	0100000	101	0110011
or	R	or rd, rs1, rs2	0000000	110	0110011
and	R	and rd, rs1, rs2	0000000	111	0110011

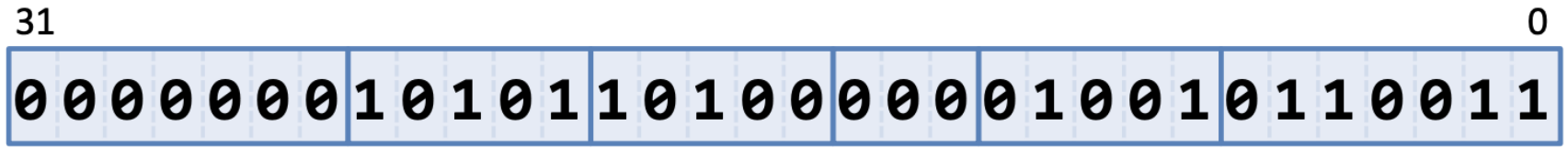
Same type

R-Type Example



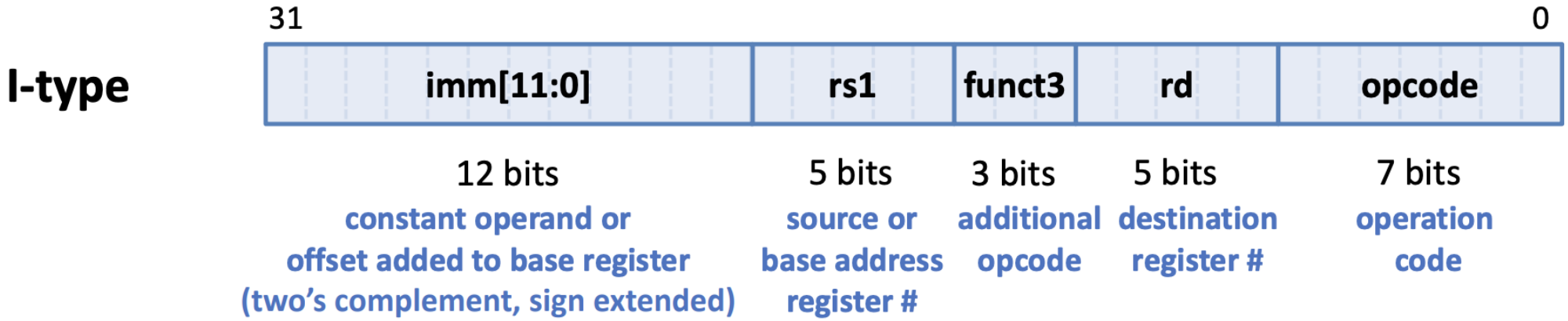
`add x9, x20, x21` == $015A04B3_{16}$





RISC-V I-Type Instructions

Immediate



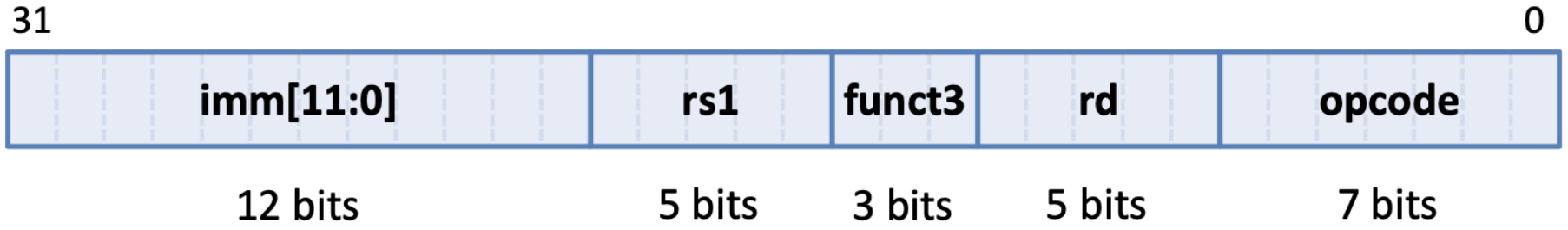
- **Immediate arithmetic or load instructions**

- **Design Principle 3: Good design demands good compromises**
  - Different formats complicate decoding
  - Keep formats as similar as possible 최대한 같은 형식 유지
  - Regularity!

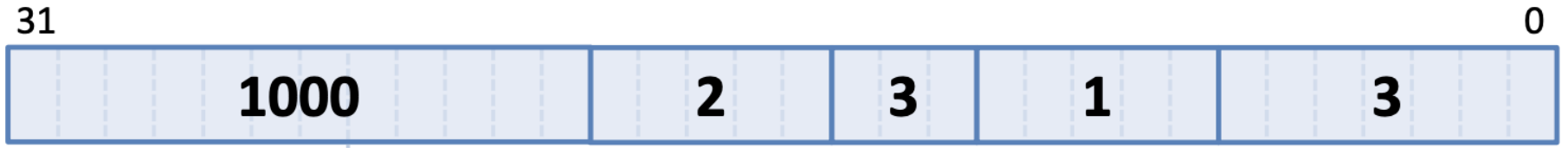
RISC-V I-Type Instructions (Cont'd)

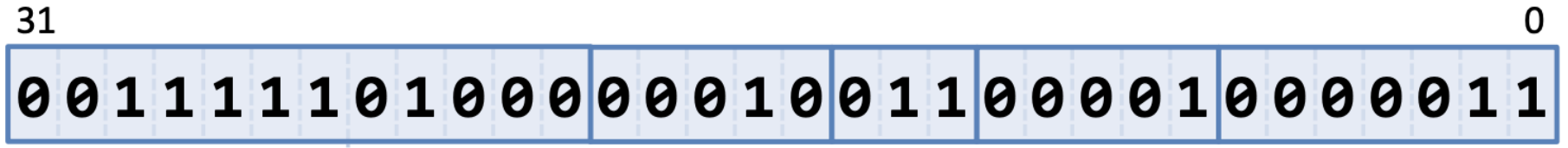
Instruction	Type	Example	funct7		funct3	opcode
addi	I	addi rd, rs1, imm12	-		000	0010011
slti	I	slti rd, rs1, imm12	-		010	0010011
sltiu	I	sltiu rd, rs1, imm12	-		011	0010011
xori	I	xori rd, rs1, imm12	-		100	0010011
ori	I	ori rd, rs1, imm12	-		110	0010011
andi	I	andi rd, rs1, imm12	-		111	0010011
slli	I	slli rd, rs1, shamt	000000	shamt	001	0010011
srli	I	srli rd, rs1, shamt	000000	shamt	101	0010011
srai	I	srai rd, rs1, shamt	010000	shamt	101	0010011
lb	I	lb rd, imm12(rs1)	-		000	0000011
lh	I	lh rd, imm12(rs1)	-		001	0000011
lw	I	lw rd, imm12(rs1)	-		010	0000011
ld	I	ld rd, imm12(rs1)	-		011	0000011
lbu	I	lbu rd, imm12(rs1)	-		100	0000011
lhu	I	lhu rd, imm12(rs1)	-		101	0000011
lwu	I	lwu rd, imm12(rs1)	-		110	0000011
jalr	I	jalr rd, imm12(rs1)	-		000	1100111

I-Type Example



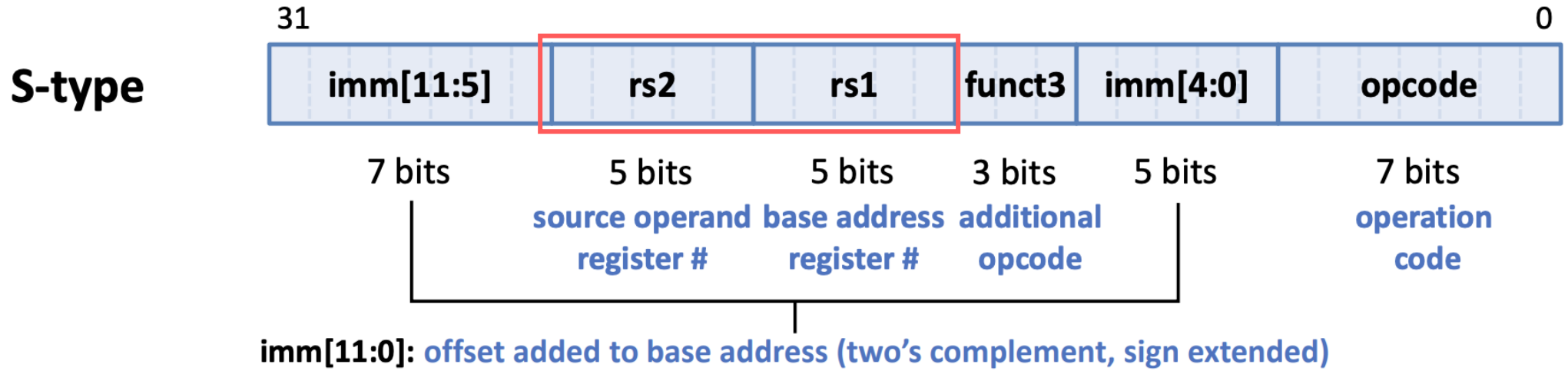
`ld x1, 1000(x2)` == $3E813083_{16}$





RISC-V S-Type Instructions

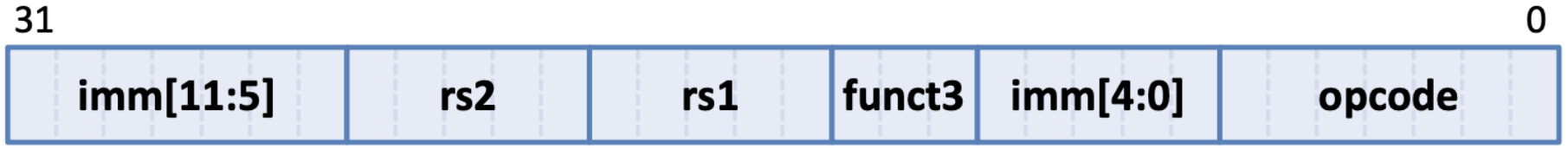
Store



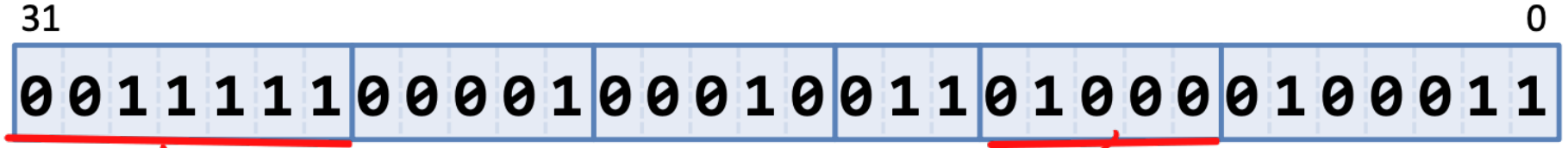
- Different immediate format for store instructions
  - Split so that rs1 and rs2 fields always in the same place

Instruction	Type	Example	funct7	funct3	opcode
sb	S	`sb rs2, imm12(rs1)`	-	000	0100011
sh	S	`sh rs2, imm12(rs1)`	-	001	0100011
sw	S	`sw rs2, imm12(rs1)`	-	010	0100011
sd	S	`sd rs2, imm12(rs1)`	-	011	0100011

S-Type Example

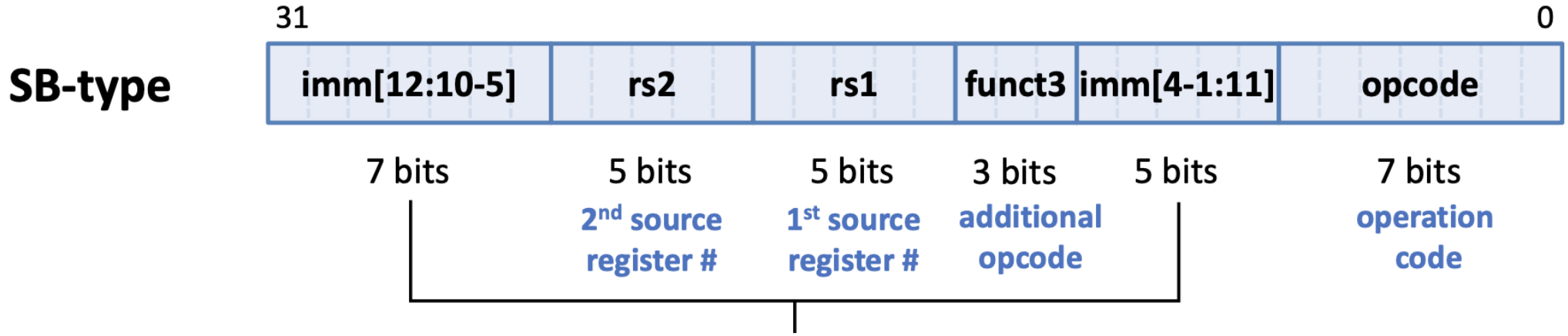


`sd x1, 1000(x2)` == $3E113423_{16}$



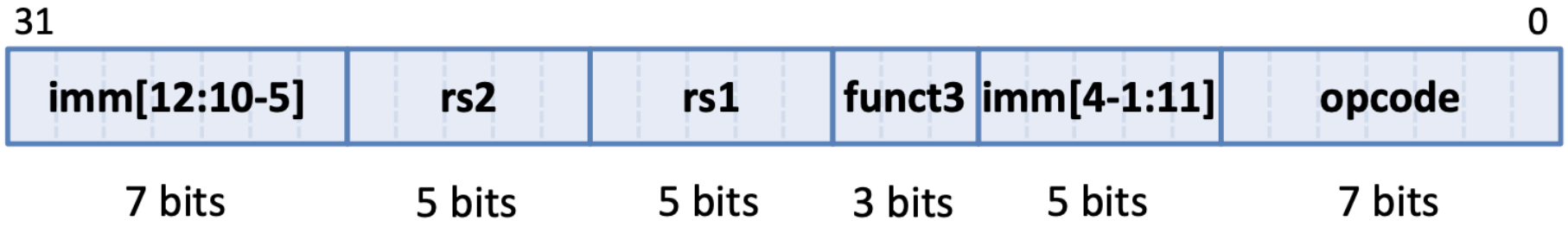
Branch

RISC-V SB-Type Instructions



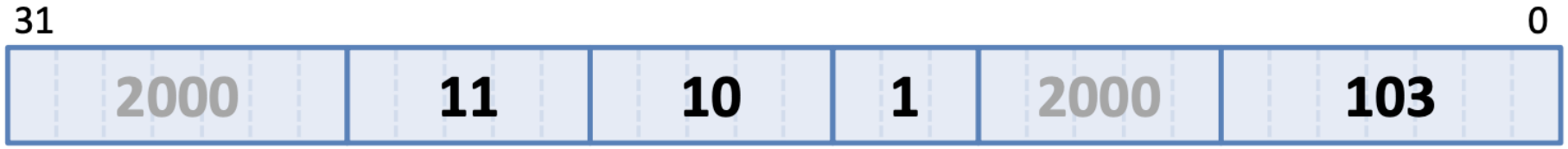
Instruction	Type	Example	funct7	funct3	opcode
beq	SB	beq rs1, rs2, imm12	-	000	1100011
bne	SB	bne rs1, rs2, imm12	-	001	1100011
blt	SB	blt rs1, rs2, imm12	-	100	1100011
bge	SB	bge rs1, rs2, imm12	-	101	1100011
bltu	SB	bltu rs1, rs2, imm12	-	110	1100011
bgeu	SB	bgeu rs1, rs2, imm12	-	111	1100011

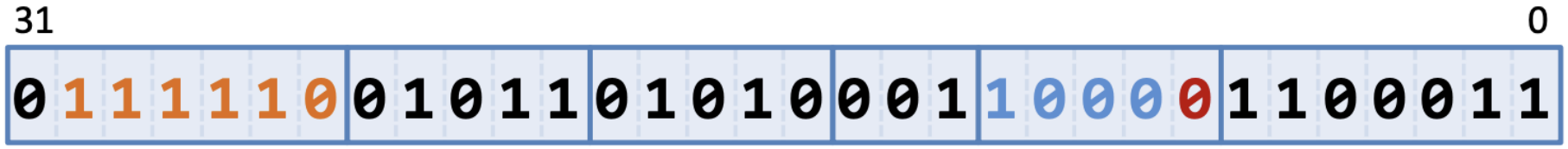
SB-Type Example



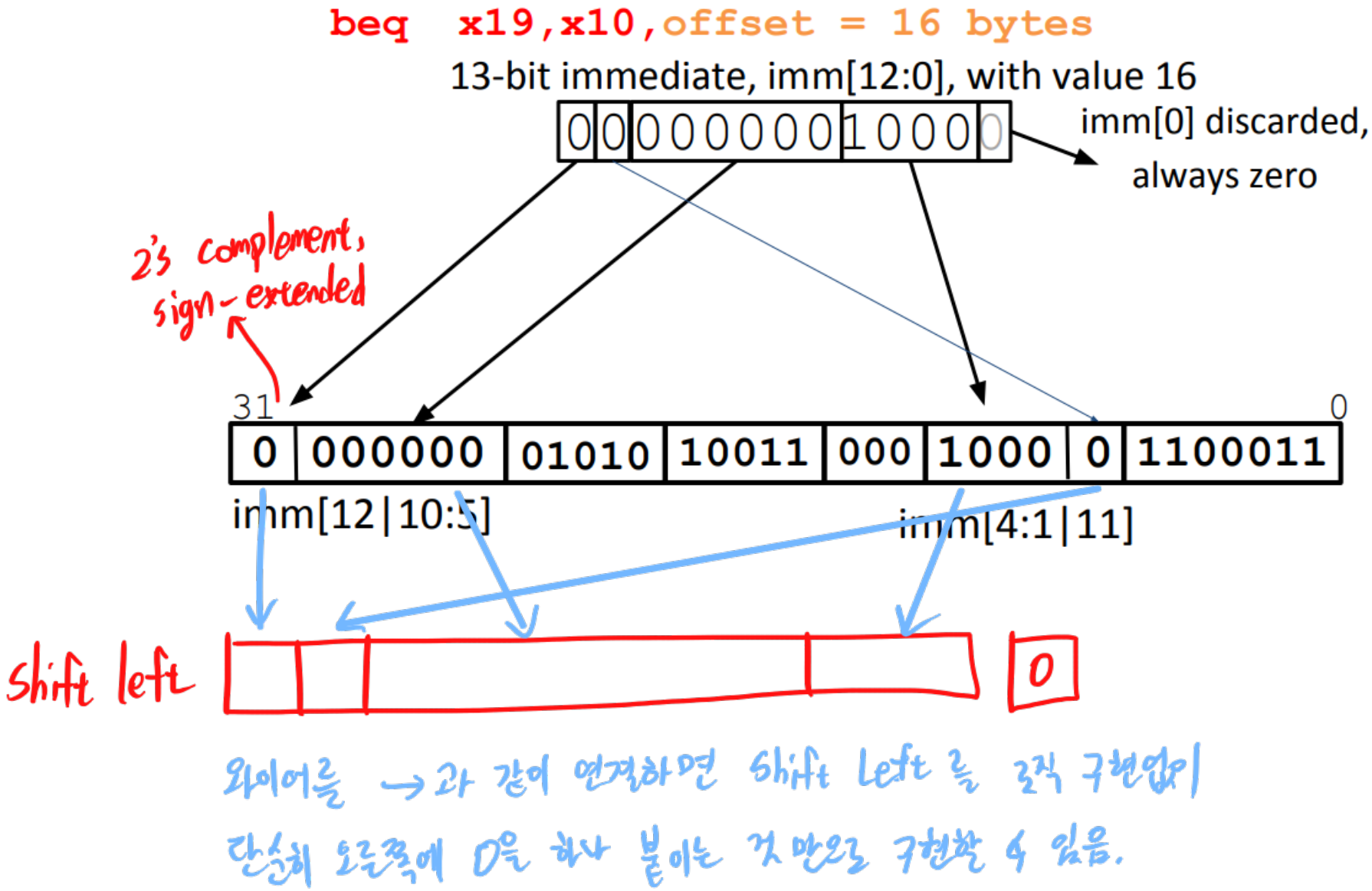
bne x10, x11, 2000 == $7CB51863_{16}$

($0111\ 1101\ 0000_2$)

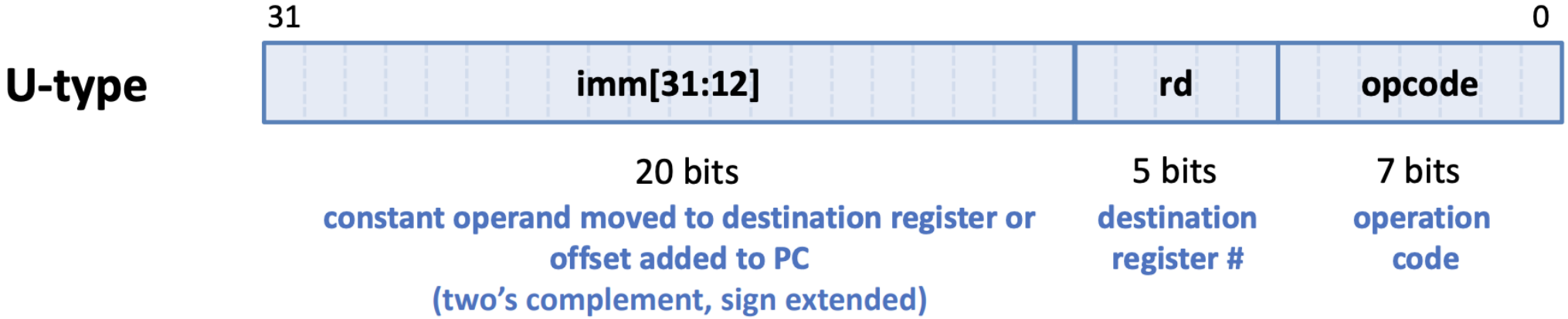




SB-Type Example



RISC-V U-Type Instructions



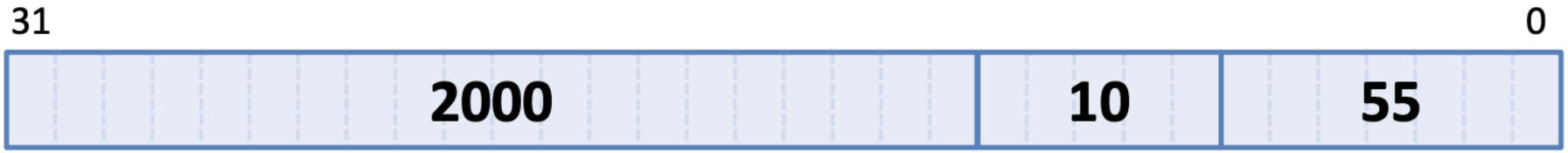
- 20-bit immediate is shifted left by 12 bits

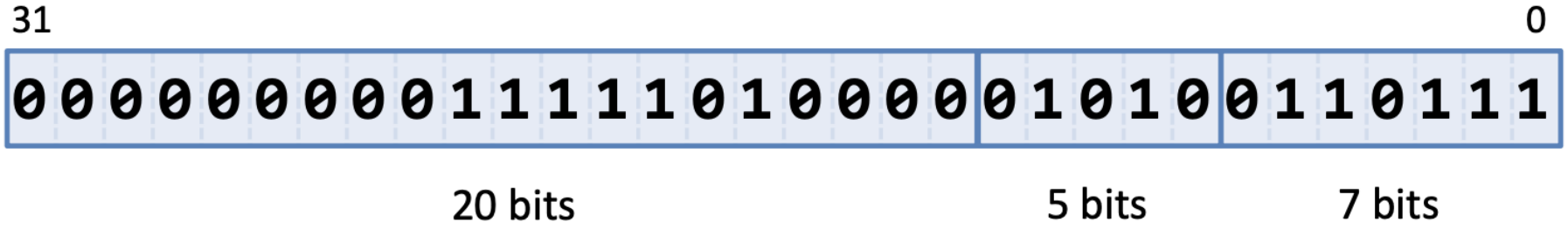
Instruction	Type	Example	funct7	funct3	opcode
lui	U	`lui rd, imm20`	-	-	0110111
auipc	U	`auipc rd, imm20`	-	-	0010111

U-Type Example

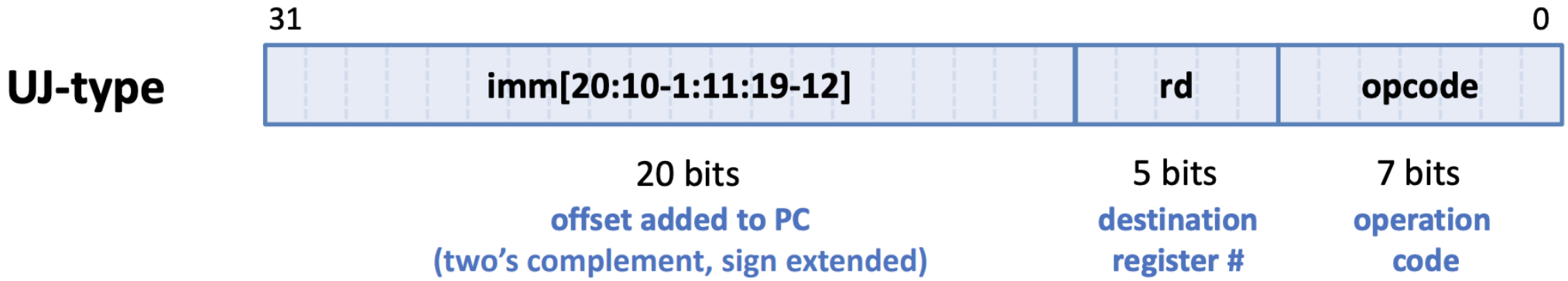
`lui x10, 2000` == $007D0537_{16}$

(0000 0000 0111 1101 0000$_2$)





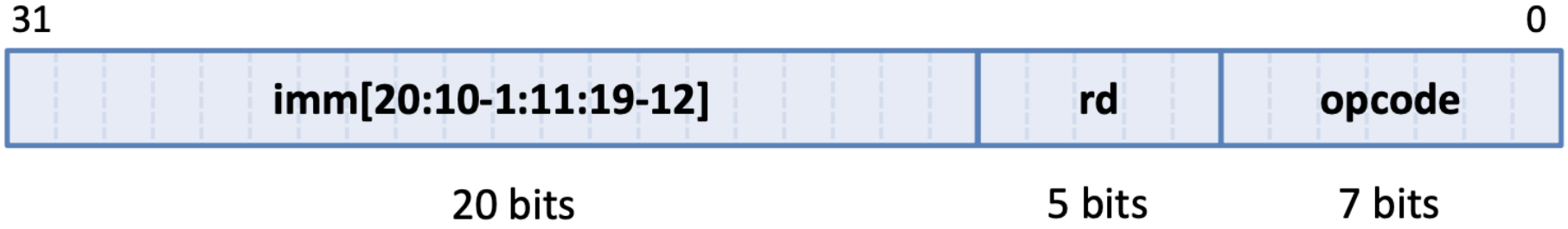
RISC-V UJ-Type Instructions



- 20-bit immediate is shifted left by 1 bit and added to PC

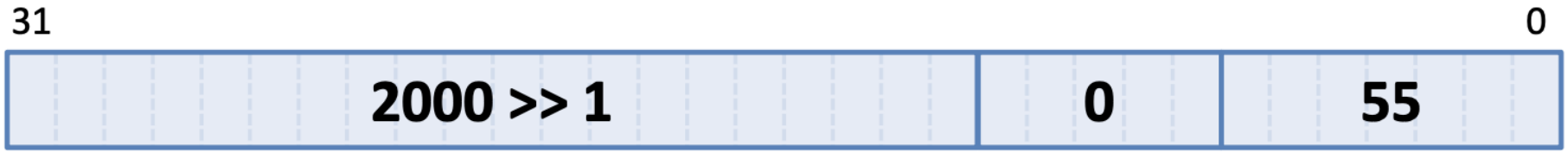
Instruction	Type	Example	funct7	funct3	opcode
jal	UJ	jal rd, imm20	-	-	1101111

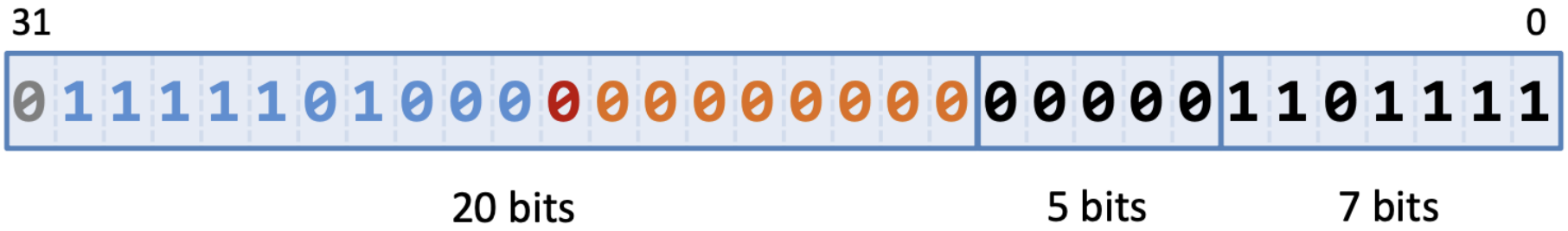
U-Type Example



jal x0, 2000 == $7D00006F_{16}$

($0000\ 0000\ 0111\ 1101\ 0000_2$)





RISC vs. CISC

RISC (Reduced Instruction Set Computer)

- **Philosophy: Fewer, simple instructions**
  - Might take more to get given task done
  - Can execute them with small and fast hardware
  - ARM, RISC-V, MIPS, IBM PowerPC, …

  한 instruction은 하나의 동작만
- **Register-oriented instruction set**
  - Many more (typically 32+) registers
  - Use for arguments, return address, temporaries
- **Only load & store instructions can access memory**
- **Each instruction has fixed size**
- **No condition codes**
  - Test instructions return 0/1 in register

RISC-V

```
   la      a0, A
   la      a1, B
   li      a2, n
   add     a3, a0, a2
L0:
   lbu     a4, 0(a0)
   sbu     a4, 0(a1)
   addi    a0, a0, 1
   addi    a1, a1, 1
   bne     a0, a3, L0
```

CISC (Complex Instruction Set Computer)

- **Add instructions to perform "typical" programming tasks**
  - IA-32, Intel 64, IBM System/360, …

한 instruction이 하나의 high-level code 동작 (여러 개의 동작)

- **Stack-oriented instruction set**
  - Use stack to pass arguments, save PC, etc.
  - Explicit push and pop instructions

x86_64

```
movq    A, %rsi
movq    B, %rdi
movq    n, %rcx
REP MOVS
```

- **Arithmetic instructions can access memory**
  - Requires memory read and write during computation
  - Complex addressing modes

- **Instructions can have varying lengths**
- **Condition codes**
  - Set as side effect of arithmetic or logical instructions

RISC vs. CISC

- **Original debate**
  - RISC proponents – better for optimizing compilers, can make run fast with simple chip design
  - CISC proponents – easy for compiler, fewer code bytes

- **Current status**
  - For desktop/server processors, choice of ISA not a technical issue
    - With enough hardware, can make anything run fast
    - Code compatibility more important
  - x86-64 adopted many RISC features
    - More registers, use them for argument passing
    - Hardware translates instructions to simpler micro-operations
  - For embedded processors, RISC makes sense: smaller, cheaper, less power

Designing an ISA

- **Important metrics**
  - Design cost to hardware and software
  - Performance and other execution measurements
    - Instructions and data
  - Static measurements (code size)

- **Influence of ISA effectiveness**
  - Program usage
  - Organization techniques
    - Pipelining, memory hierarchies
    - Compiler technology
    - OS requirements
    - Implementation technology (e.g., memory vs. logic, frequency vs. parallelism)

Four Design Principles

- Simplicity favors regularity
- Smaller is faster
- Good design demands good compromises

- Make the common case fast