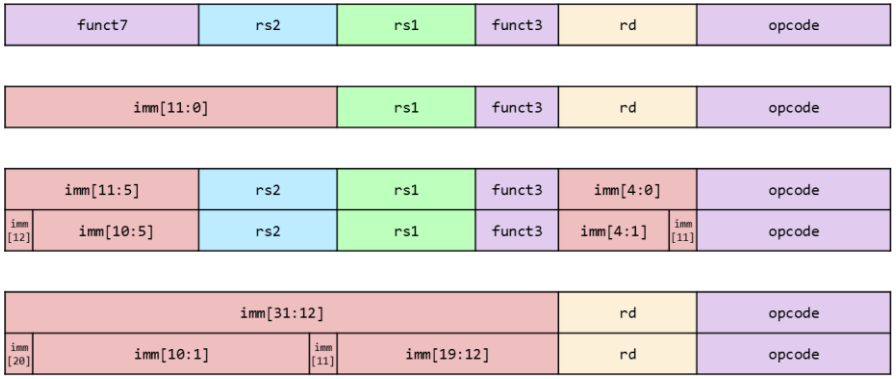
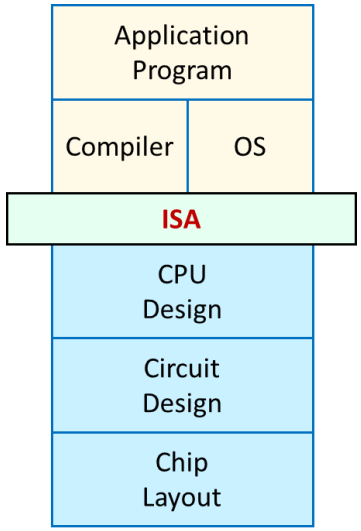
# Processor Architecture

# RISC-V ISA Encoding



# Module Outline

- Recap: The Instruction Set Architecture
- RISC-V Instruction Formats
- RISC-V Instruction Encodings
  - R-Type Instructions
  - I-Type Instructions
  - S-Type Instructions
  - SB-Type Instructions
  - U-Type Instructions
  - UJ-Type Instructions
- Module Summary

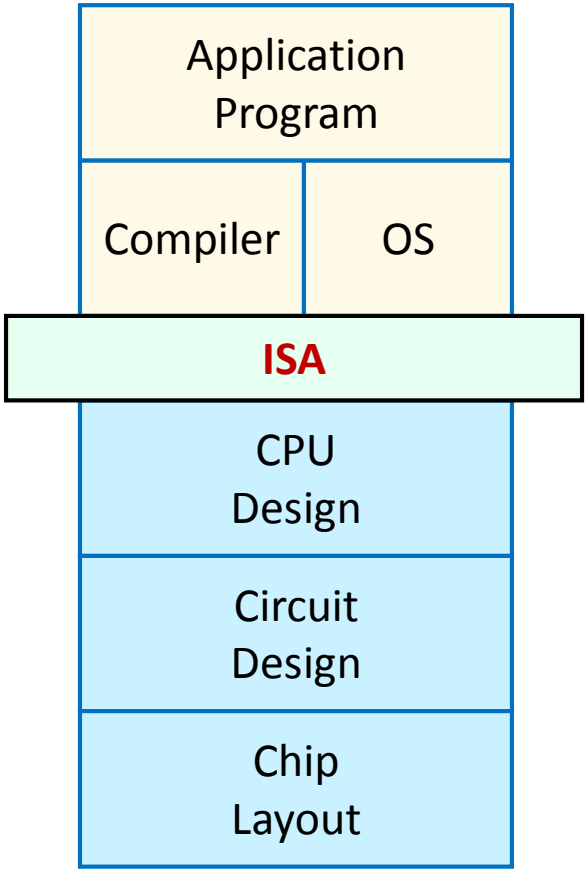



## Recap

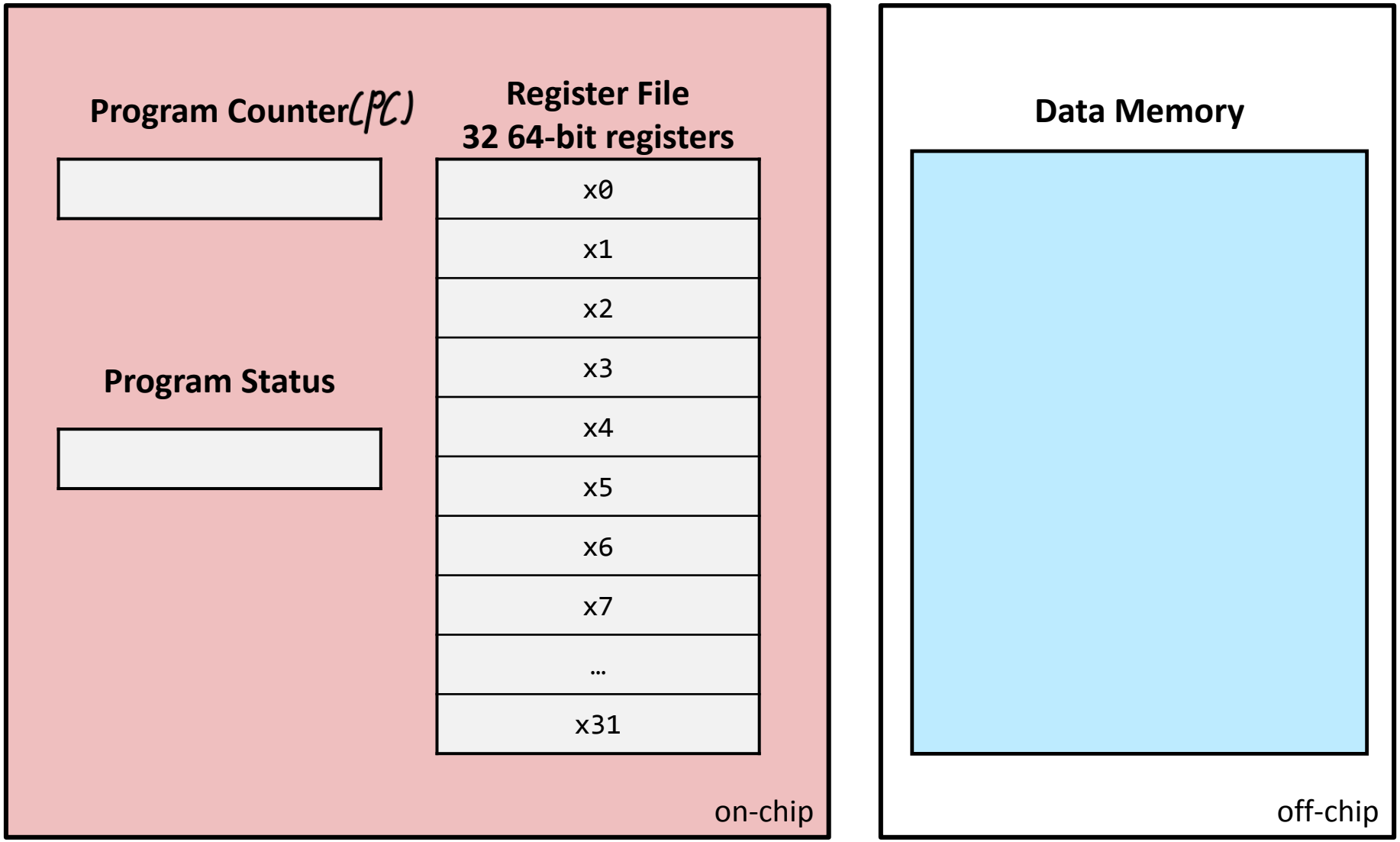
# The Instruction Set Architecture

# Recap – The Role of the ISA

- Assembly language view
  - Processor state
    - Registers, memory, …
  - Instructions
    - `addi`, `ld`, `jral`, …
    - How instructions are encoded as bytes
- Layer of abstraction
  - Above: how to program machine
    - Processor executes instructions in a sequence
  - Below: what needs to be built
    - Use variety of tricks to make it run fast
    - E.g., execute multiple instructions simultaneously

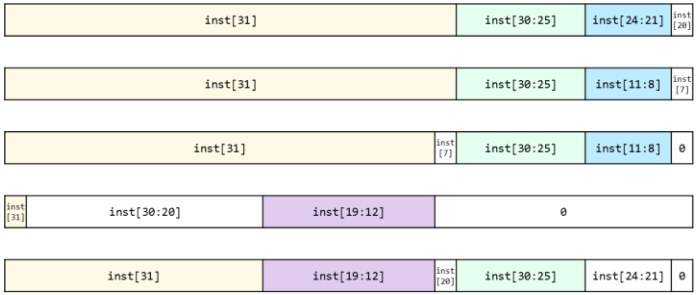


# Recap – The (Visible) RISC-V Processor State



# Recap – The (Visible) RISC-V Processor State

- **Program Registers**
  - 32 64-bit registers
  - x0 hard-wired to zero
  - ABI: x1: link register, x2: stack pointer
- **Program Counter**
  - Indicates address of next instruction
- **Program Status**
  - Indicates either normal operation or some error condition
- **Memory**
  - Byte-addressable storage array
  - Words stored in little-endian byte order



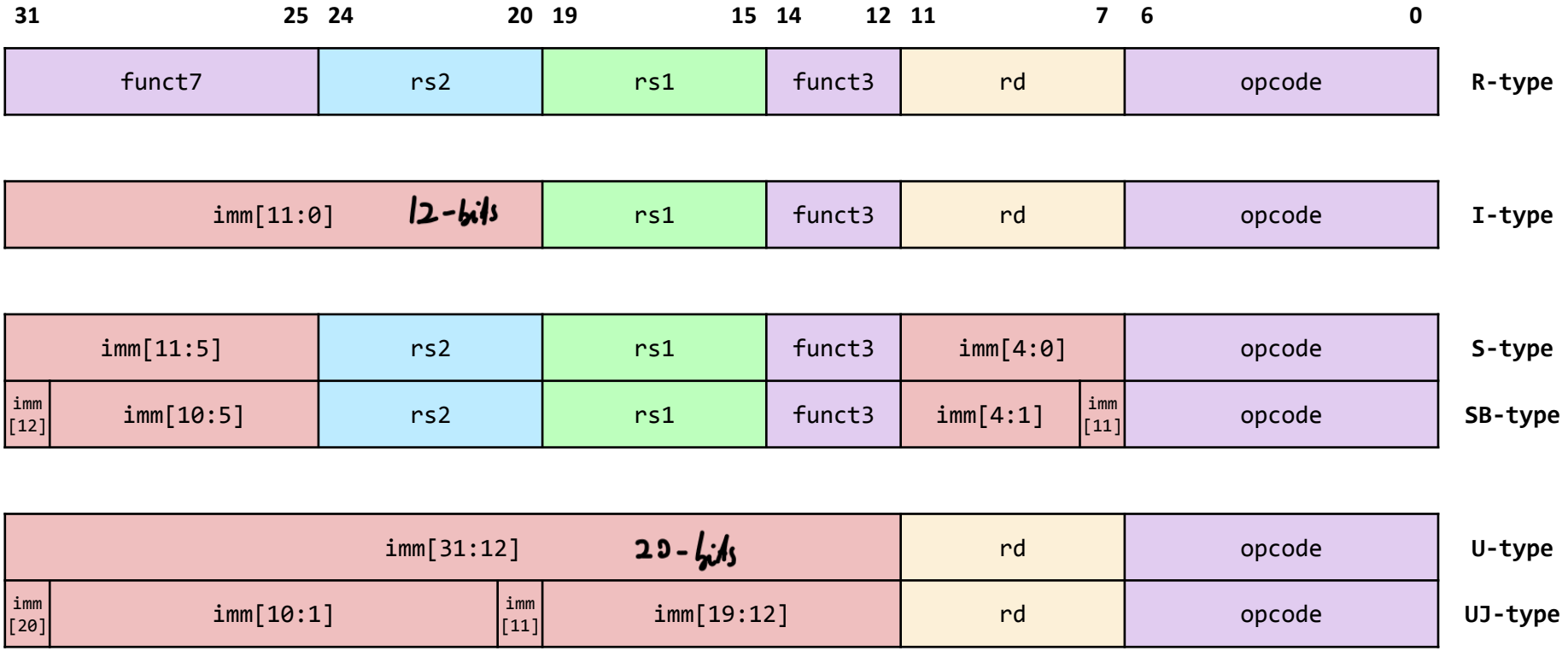

# RISC-V Instruction Formats

# Representing Instructions

- Instructions are encoded in binary
  - Called machine code

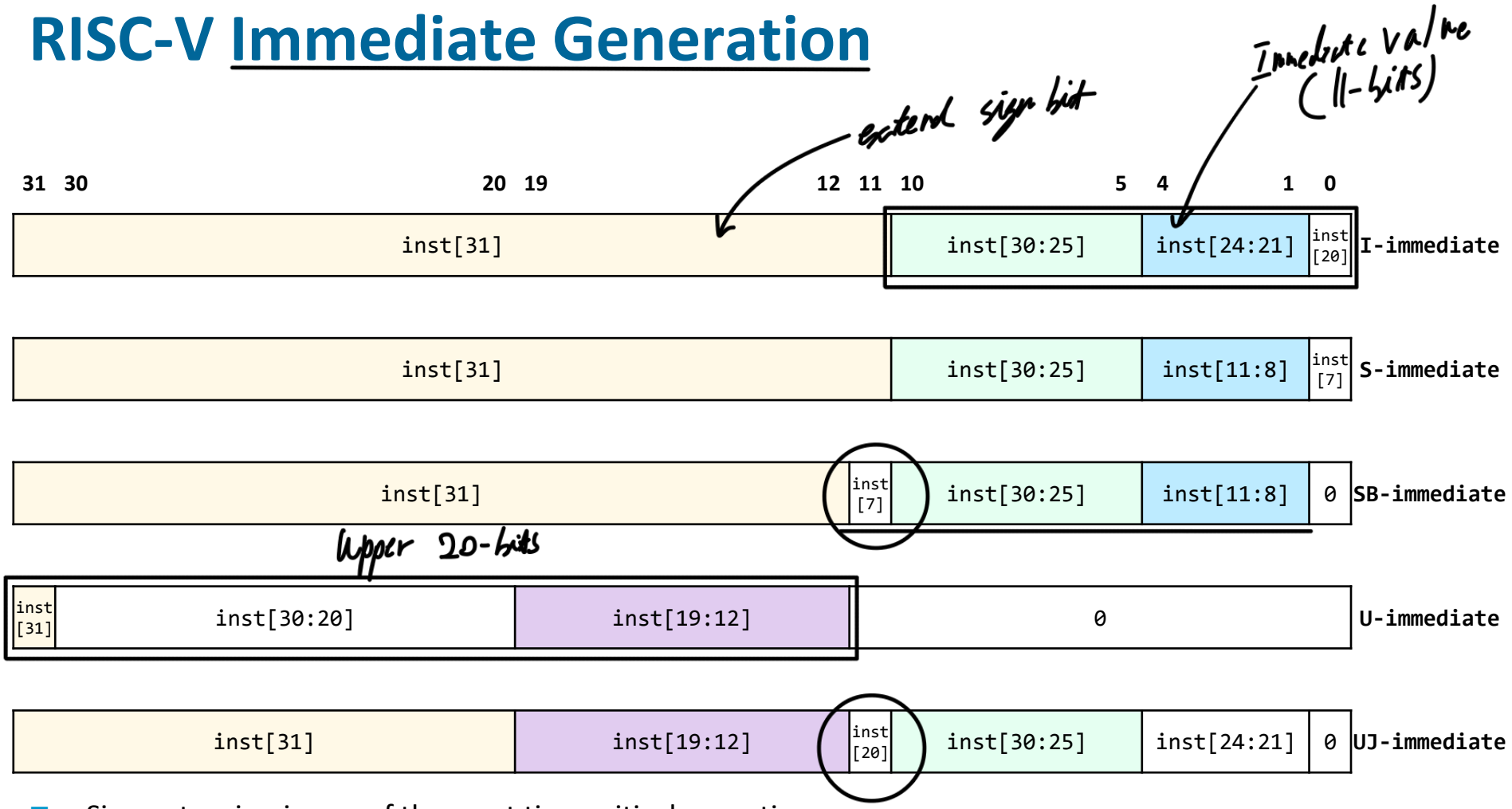
- RISC-V instructions
  - Encoded as 32-bit instruction words
  - Small number of formats encoding operation code (opcode), register numbers, …
  - Design principle: regularity

# RISC-V Instruction Formats



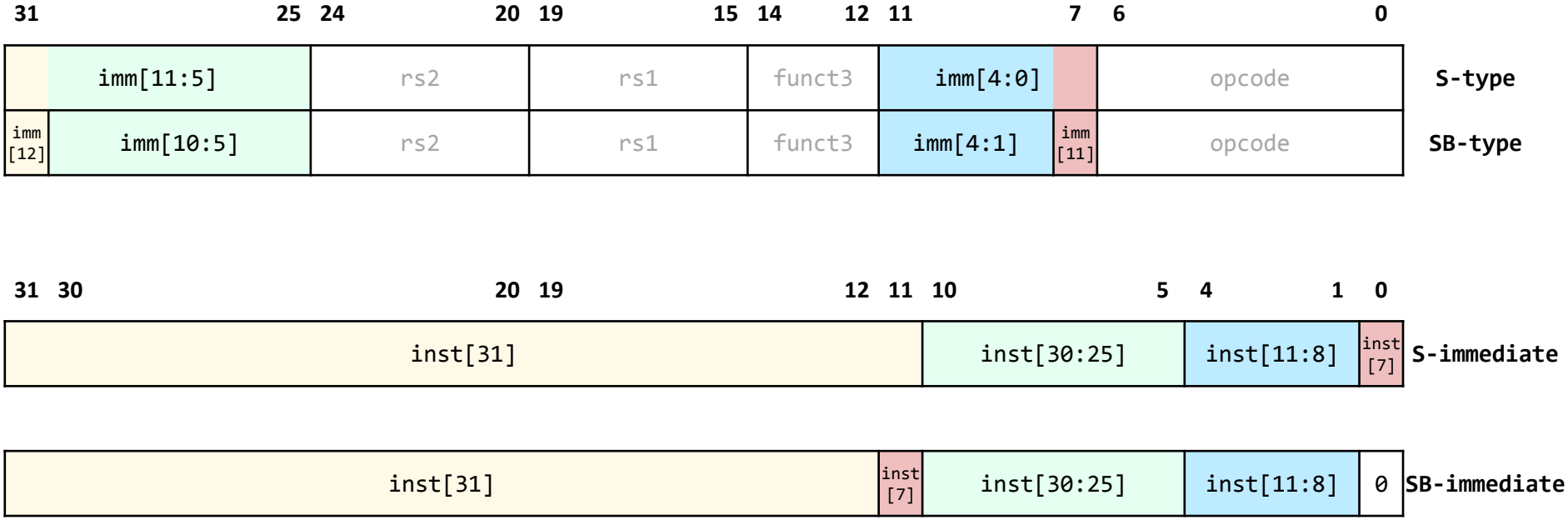

- Four basic types with two subtypes
- Type determined by opcode
- B/J-type: immediate value is encoded value shifted by 2
  - Seemingly weird immediate encoding chosen to maximize regularity

# RISC-V Immediate Generation



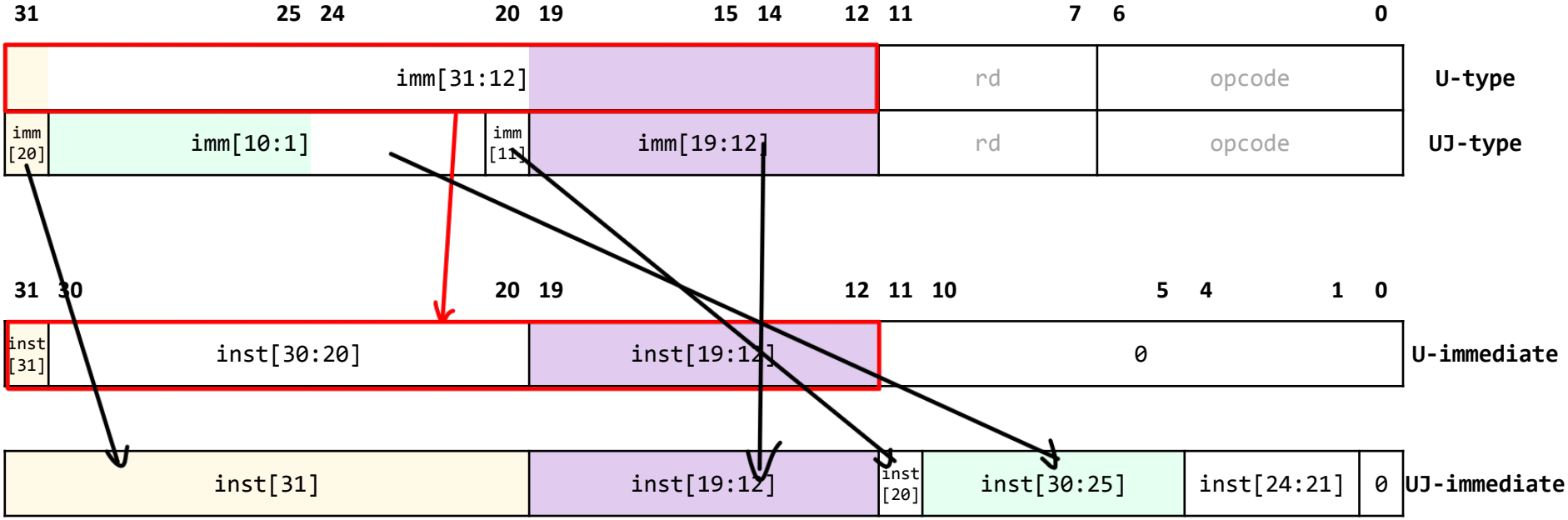

- Sign-extension is one of the most time critical operations
  - Using the same bit position as the sign bit allows sign-extension to proceed in parallel with instruction decoding
- B/J-type: multiplication by 2 implicit through immediate mapping
  - shift work from hardware (at-runtime) to software (compiler; before-runtime)

# RISC-V Immediate Generation Example

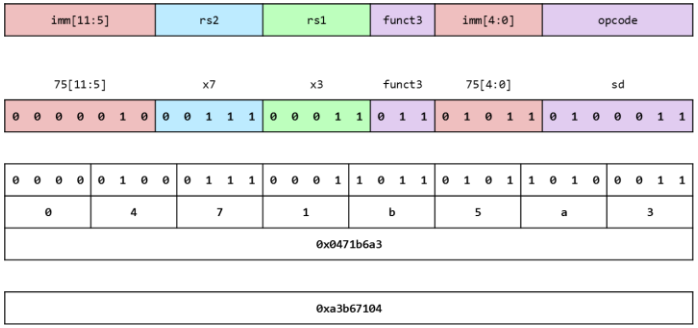



- Bit 31 of machine instruction is sign bit
- Bits 30-25 of encoding mapped to immediate at 10-5
- Bits 11-8 of encoding mapped to immediate at 4-1
- Only bit 7 is mapped to different positions (S: 0, SB: 11)
- SB-immediate: implicit multiplication by 2

# RISC-V Immediate Generation Example

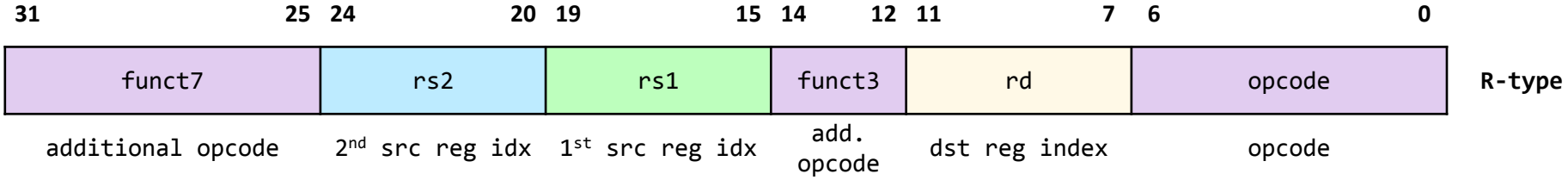



- Bit 31 of machine instruction is sign bit
- Bits 30-21 mapped directly (U) or moved to 10-1 (UJ)
- Bits 19-12 mapped directly
- Bit 20 is mapped directory (U) or to bit 11 (UJ)
- UJ-immediate: implicit multiplication by 2




# RISC-V Instruction Encodings

# R-Type Instructions

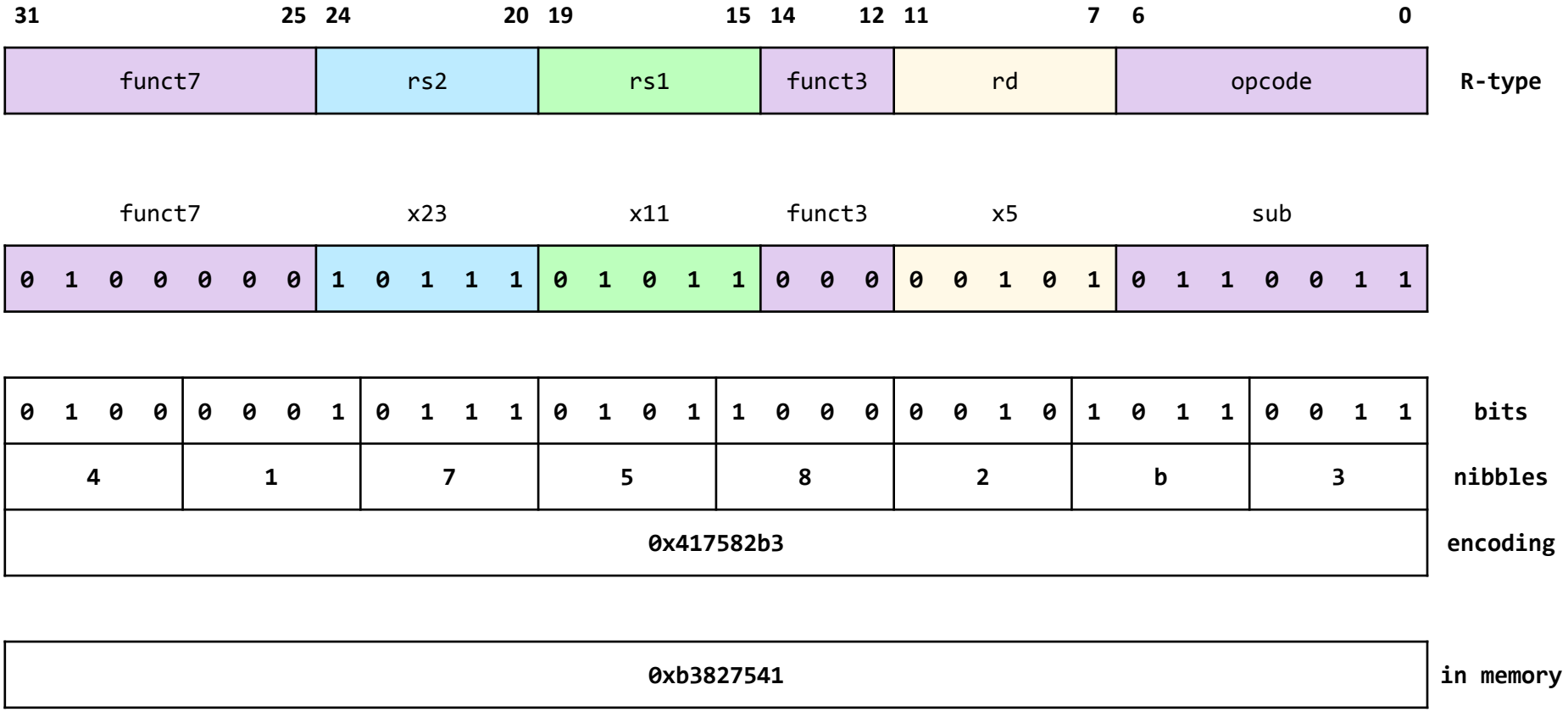



select function  R-type

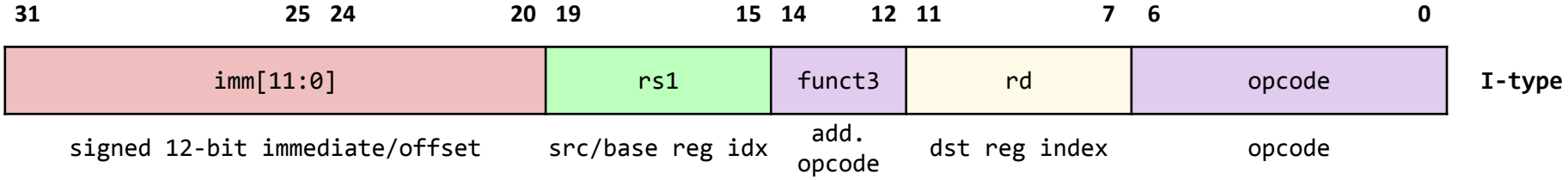
| Instruction | Type | Example | funct7 | funct3 | opcode |
|---|---|---|---|---|---|
| **add** | **R** | add rd, rs1, rs2 | 0000000 | 000 | 0110011 |
| **sub** | **R** | sub rd, rs1, rs2 | 0100000 ✓ | 000 | 0110011 |
| **sll** | **R** | sll rd, rs1, rs2 | 0000000 | 001 | 0110011 |
| **slt** | **R** | 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 |

# R-Type Instruction Example

**sub x5, x11, x23**



# I-Type Instructions



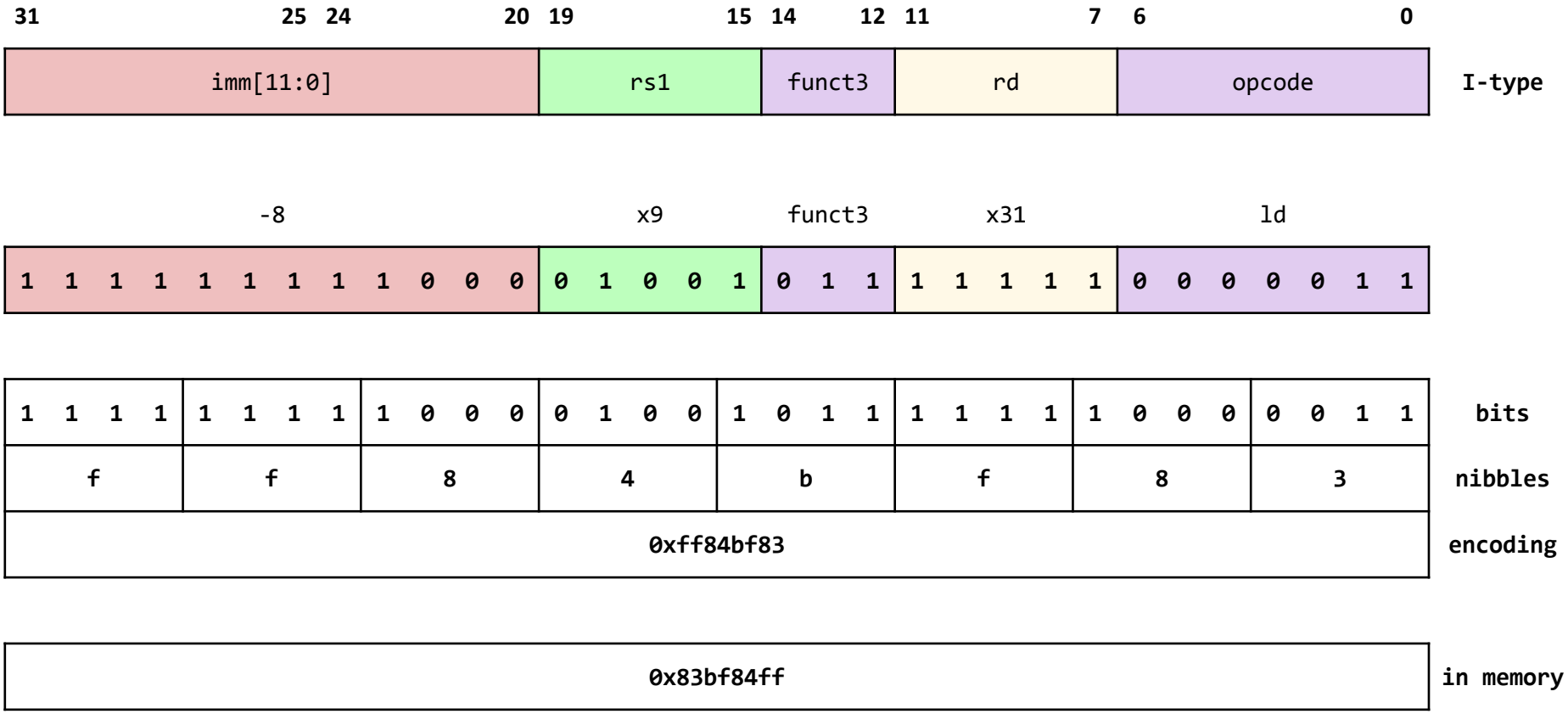

| Instruction | Type | Example | inst[31:20] | funct3 | opcode |
|---|---|---|---|---|---|
| **addi** | **I** | addi rd, rs1, imm12 | imm12 | 000 | 0010011 |
| **slti** | **I** | slti rd, rs1, imm12 | imm12 | 010 | 0010011 |
| **sltiu** | **I** | sltiu rd, rs1, imm12 | imm12 | 011 | 0010011 |
| **xori** | **I** | xori rd, rs1, imm12 | imm12 | 100 | 0010011 |
| **ori** | **I** | ori rd, rs1, imm12 | imm12 | 110 | 0010011 |
| **andi** | **I** | andi rd, rs1, imm12 | 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) | imm12 | 000 | 0000011 |
| **lh** | **I** | lh rd, imm12(rs1) | imm12 | 001 | 0000011 |
| **lw** | **I** | lw rd, imm12(rs1) | imm12 | 010 | 0000011 |
| **ld** | **I** | ld rd, imm12(rs1) | imm12 | 011 | 0000011 |
| **lbu** | **I** | lbu rd, imm12(rs1) | imm12 | 100 | 0000011 |
| **lhu** | **I** | lhu rd, imm12(rs1) | imm12 | 101 | 0000011 |
| **lwu** | **I** | lwu rd, imm12(rs1) | imm12 | 110 | 0000011 |
| **jalr** | **I** | jalr rd, imm12(rs1) | imm12 | 000 | 1100111 |

load

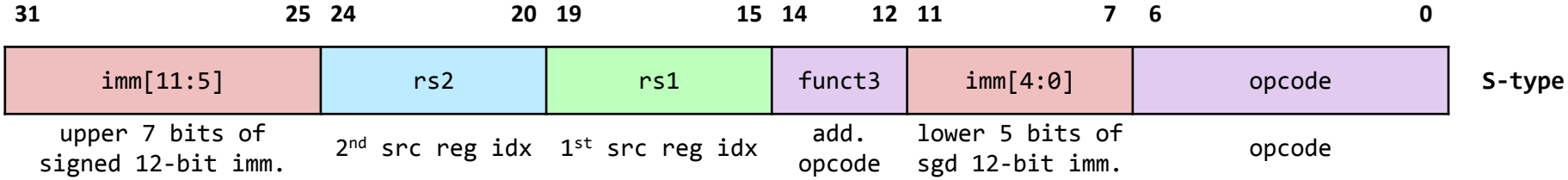
new branch

# I-Type Instruction Example

**ld x31, -8(x9)**

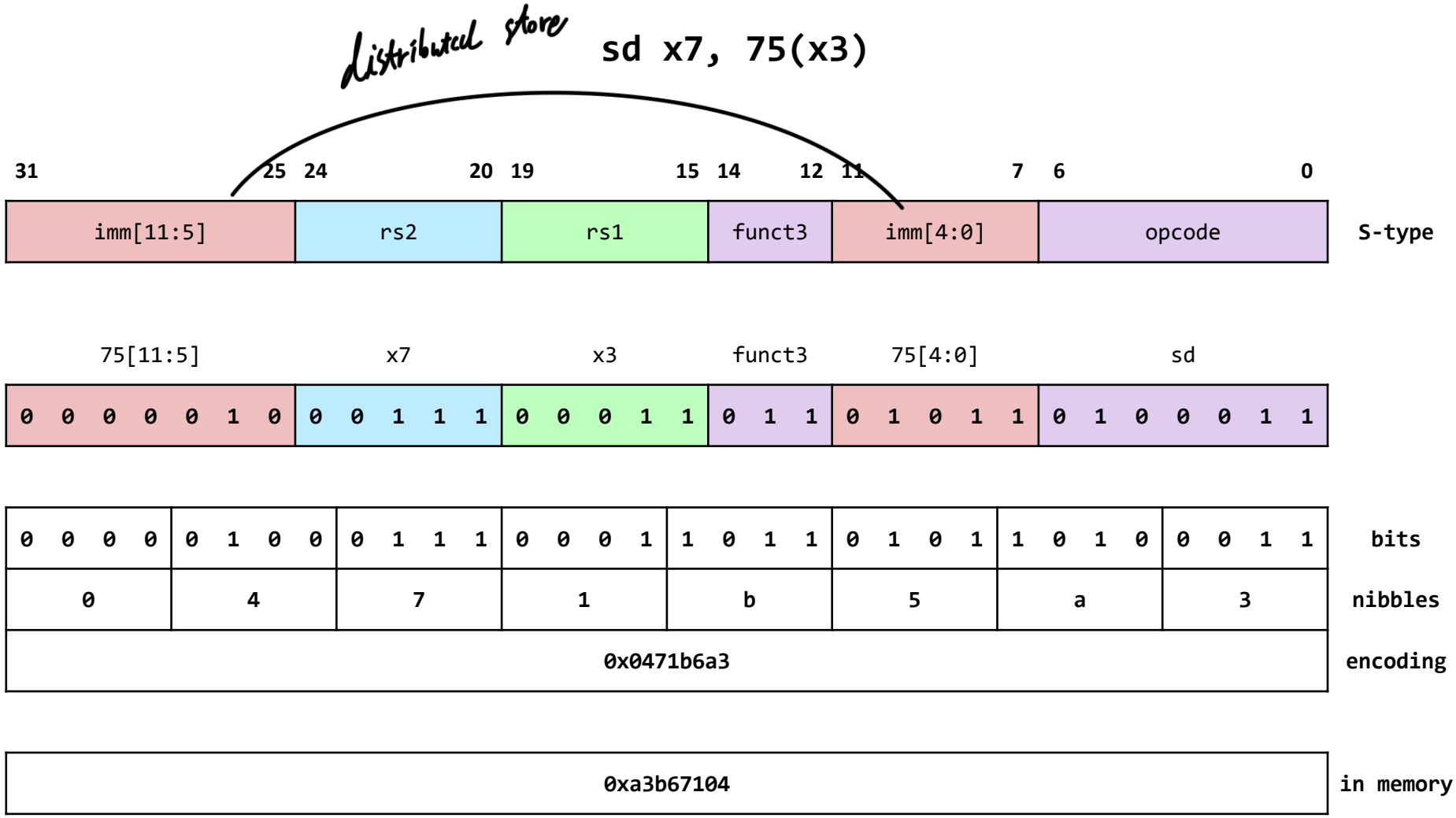


# S-Type Instructions : Save

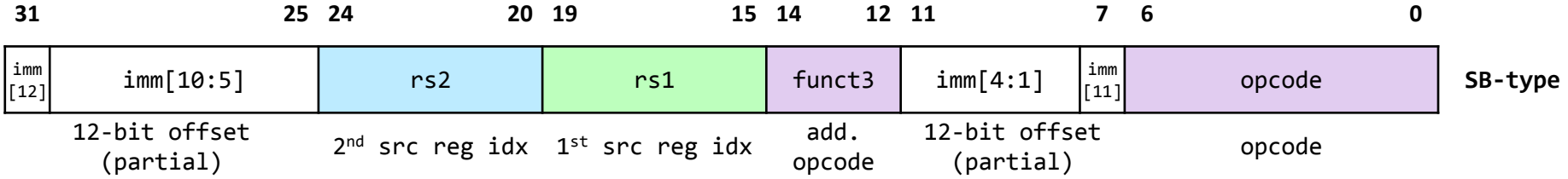



| Instruction | Type | Example | inst[31:25\|11:7] | funct3 | opcode |
|---|---|---|---|---|---|
| **sb** | **S** | sb rs2, imm12(rs1) | imm12 | 000 | 0100011 |
| **sh** | **S** | sh rs2, imm12(rs1) | imm12 | 001 | 0100011 |
| **sw** | **S** | sw rs2, imm12(rs1) | imm12 | 010 | 0100011 |
| **sd** | **S** | sd rs2, imm12(rs1) | imm12 | 011 | 0100011 |

# S-Type Instruction Example



# SB-Type Instructions : branch



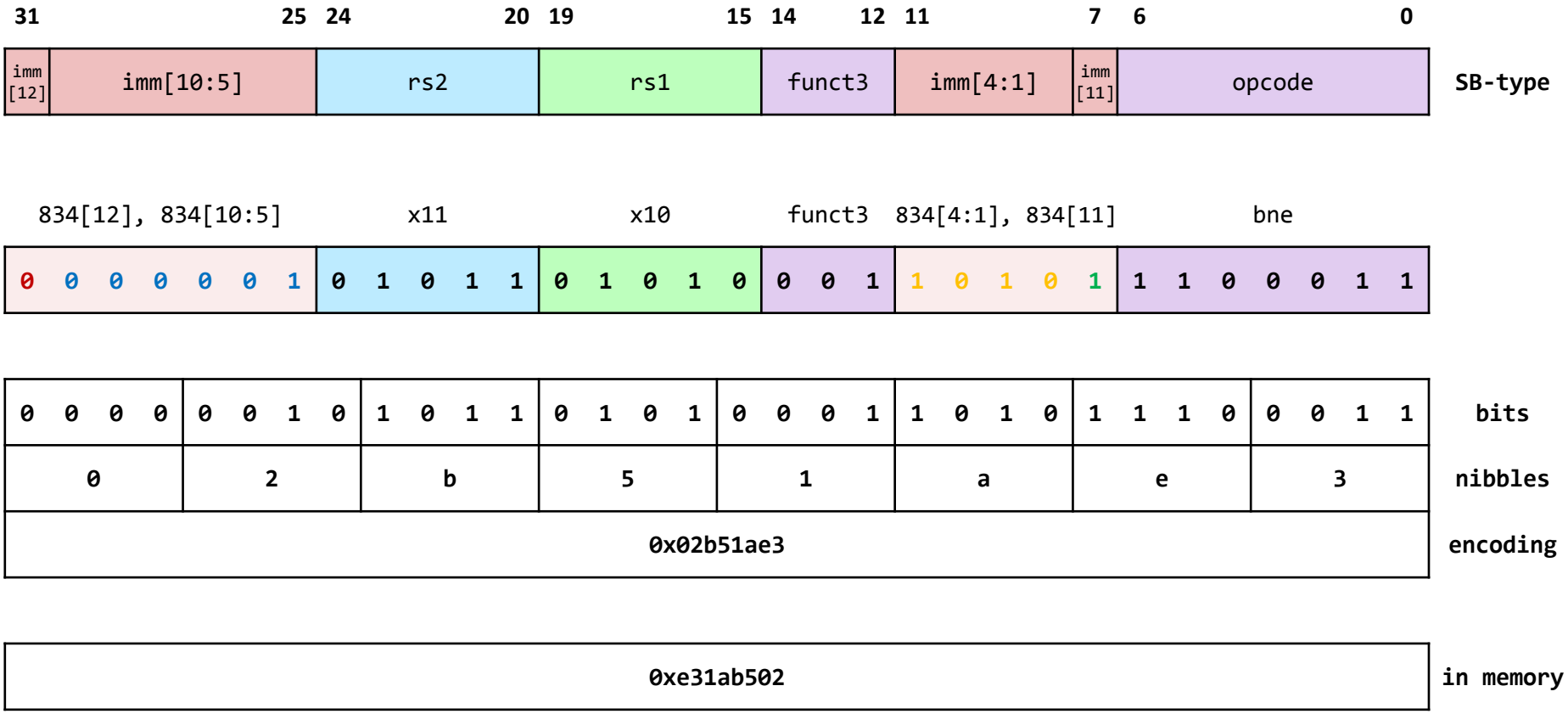

- (imm[12:0] << 1) added to PC

| Instruction | Type | Example | inst[31\|7\|30:25\|4:1] | funct3 | opcode |
|---|---|---|---|---|---|
| **beq** | SB | beq rs1, rs2, imm13 | imm13 >> 1 | 000 | 1100011 |
| **bne** | SB | bne rs1, rs2, imm13 | imm13 >> 1 | 001 | 1100011 |
| **blt** | SB | blt rs1, rs2, imm13 | imm13 >> 1 | 100 | 1100011 |
| **bge** | SB | bge rs1, rs2, imm13 | imm13 >> 1 | 101 | 1100011 |
| **bltu** | SB | bltu rs1, rs2, imm13 | imm13 >> 1 | 110 | 1100011 |
| **bgeu** | SB | bgeu rs1, rs2, imm13 | imm13 >> 1 | 111 | 1100011 |

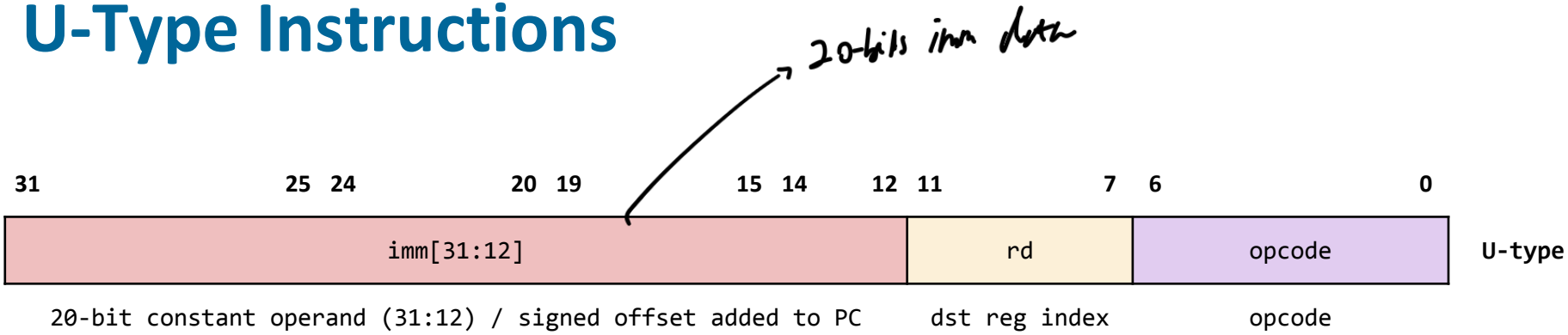
# SB-Type Instruction Example

**bne x10, x11, 2100**

**2100 = 0x834 = 0 1000 0011 0100**



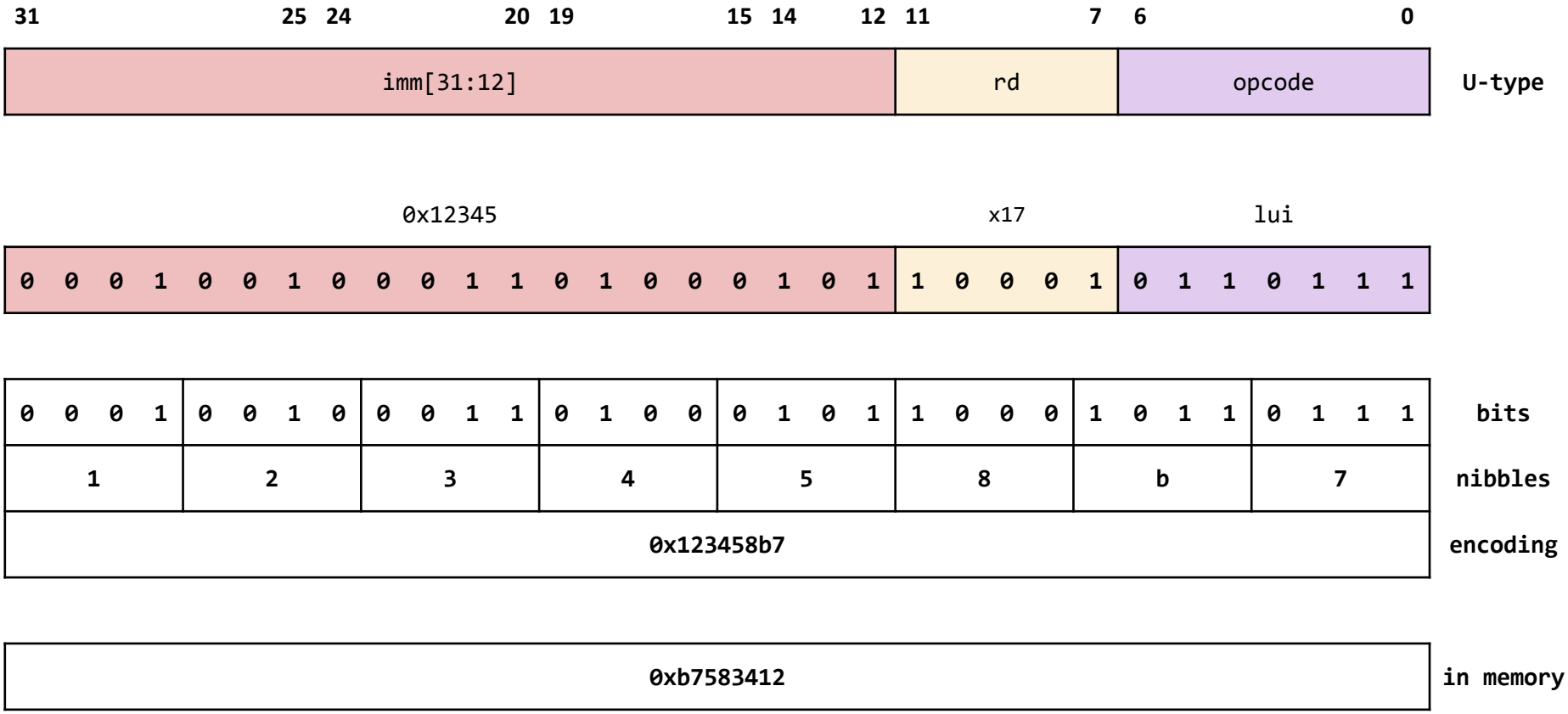
# U-Type Instructions



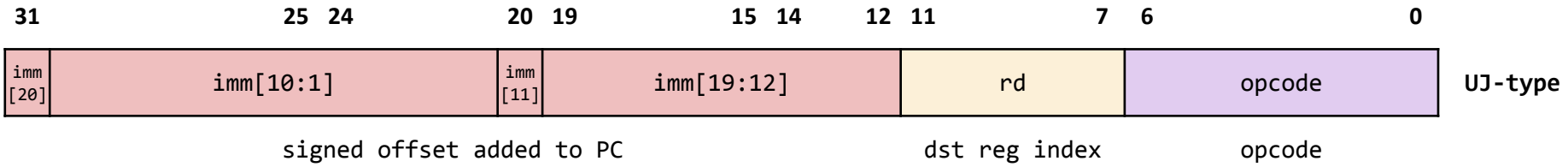

| Instruction | Type | Example | inst[31:12] | funct3 | opcode |
|---|---|---|---|---|---|
| **lui** | **U** | lui rd, imm20 | imm20 | - | 0110111 |
| **auipc** | **U** | auipc rd, imm20 | imm20 | - | 0010111 |

# U-Type Instruction Example

**lui x17, 0x12345**



# UJ-Type Instructions

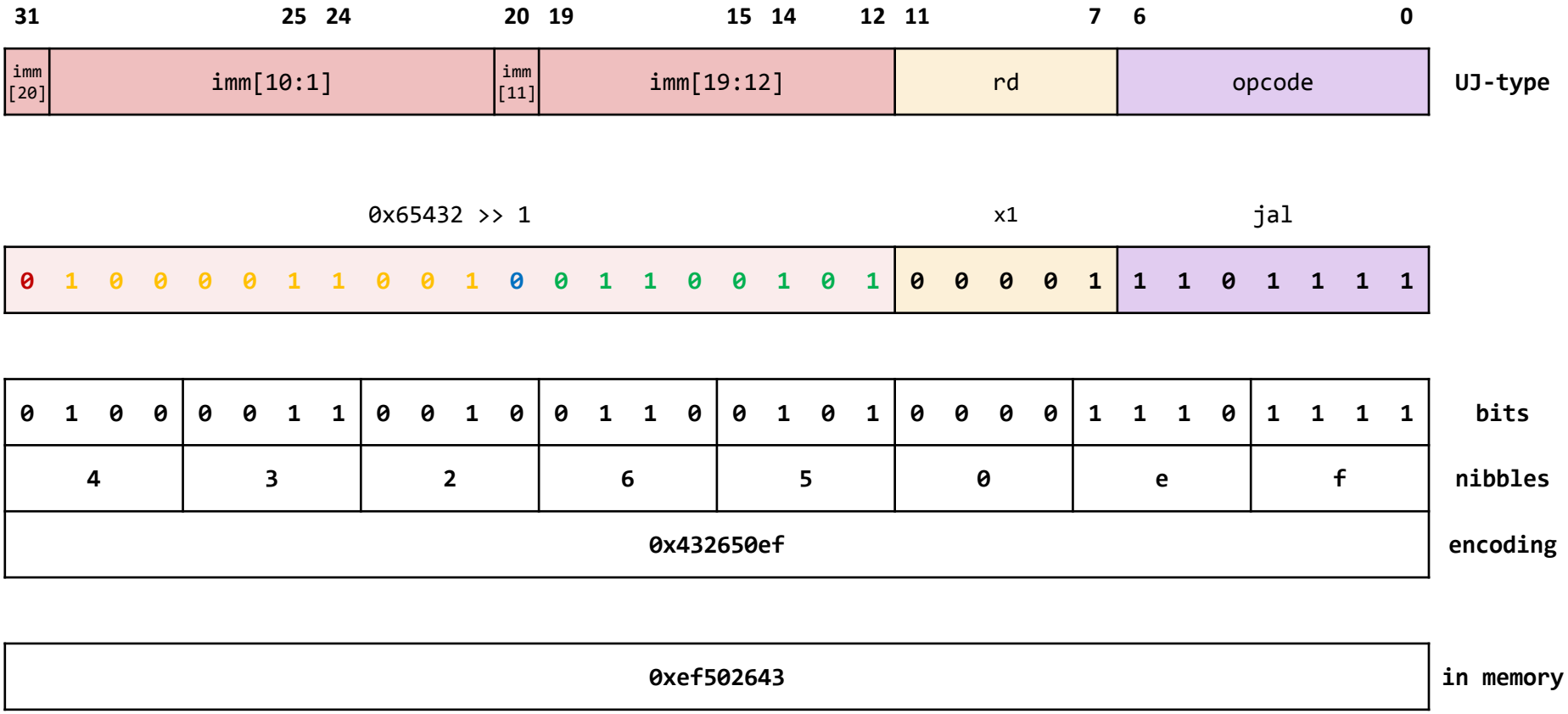


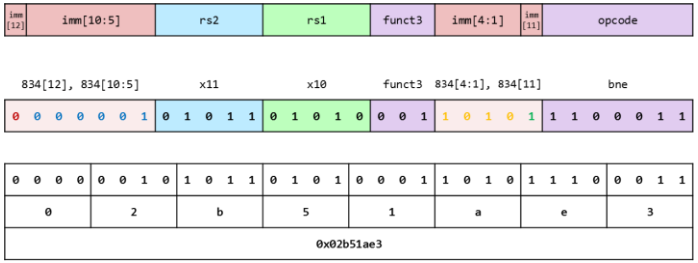

| Instruction | Type | Example | inst[31\|19:12\|20\|30:21] | funct3 | opcode |
|---|---|---|---|---|---|
| **jal** | **UJ** | jal rd, imm21 | imm21 >> 1 | - | 1101111 |

# UJ-Type Instruction Example

**jal x1, 0x65432**

**0x65432 = 0 0110 0101 0100 0011 0010**






# Module Summary

# Module Summary – RISC-V Instruction Encoding

- **Designed for maximal regularity**
- **Four basic encoding types:**
  - R, I, S, U
- **Two subtypes**
  - B (SB), J (UJ)
- **Immediate encoding shifts work from hardware to software**