Introduction to Computer Architecture

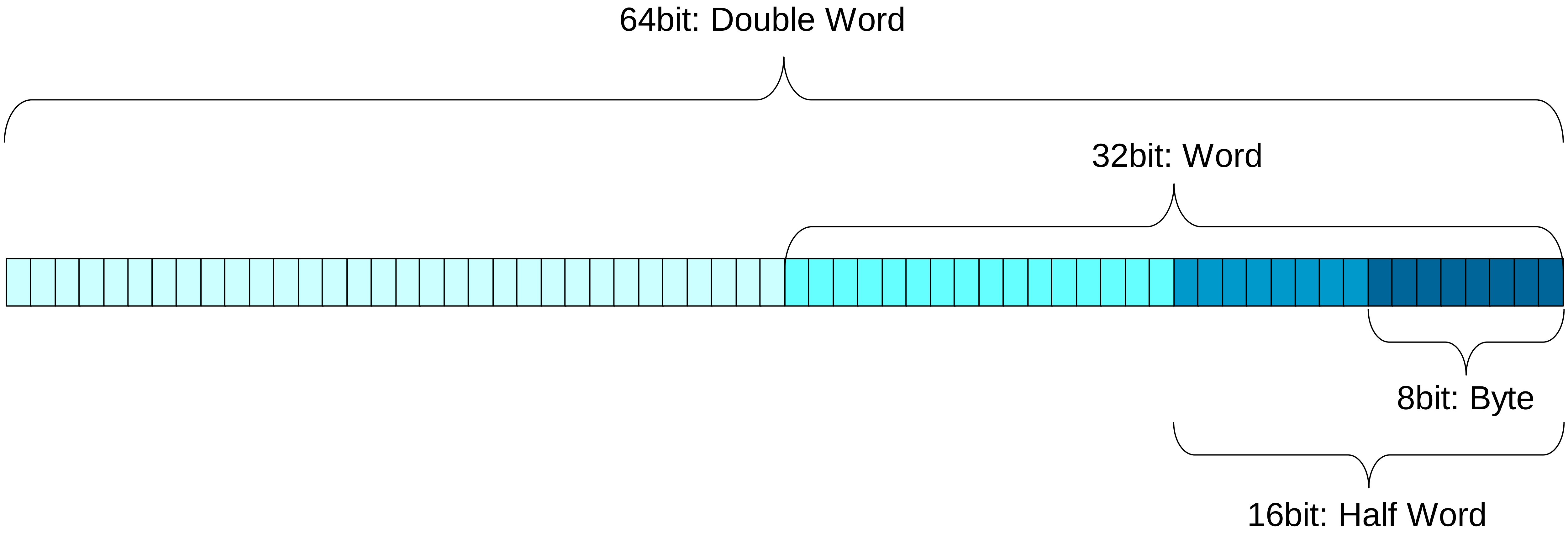
Chapter 2

Instructions: Language of the Computer - 2

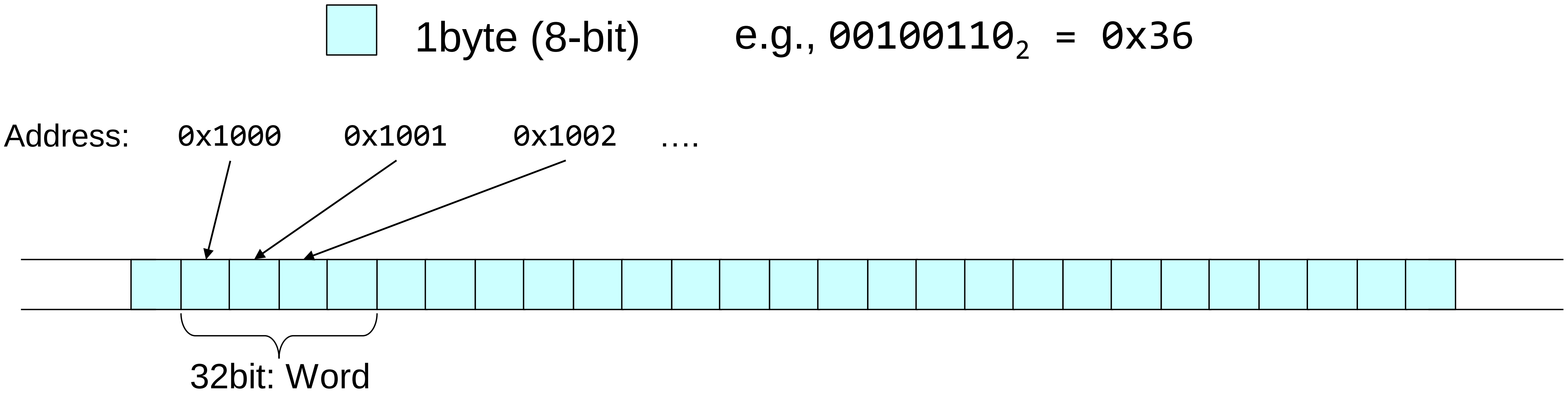
Hyungmin Cho

Department of Computer Science and Engineering
Sungkyunkwan University

Data Size



Memory addressing



1byte (8-bit)   e.g., 00100110_2 = 0x36

Address: 0x1000 0x1001 0x1002 ….

32bit: Word

- Memory “address” is also a 32-bit (non-negative) integer.
  - 0x0000_0000 – 0xFFFF_FFFF

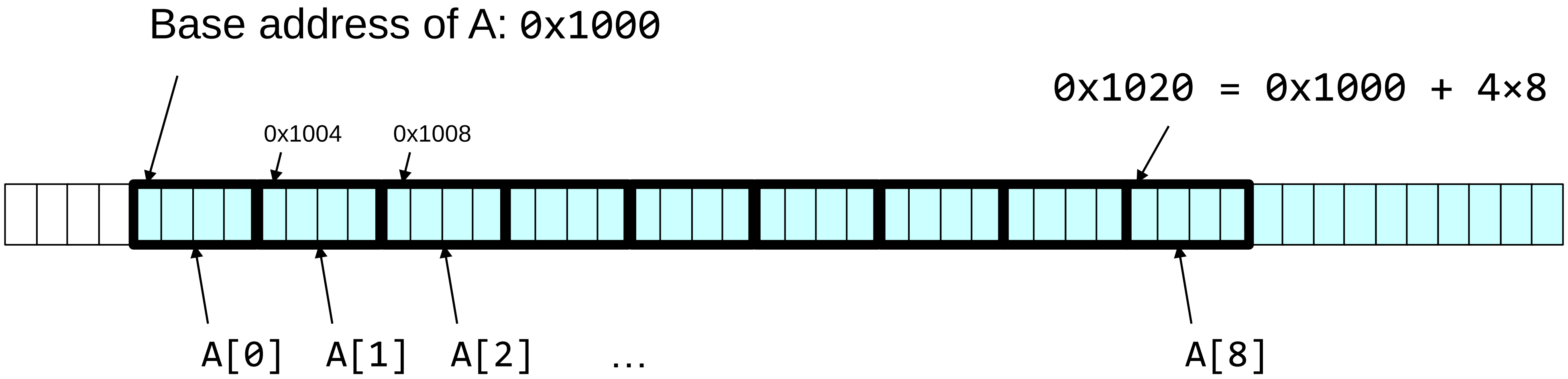
- Memory → register: **Load**   Need address & destination register
- Register → memory: **Store**   Need address & source register

Memory Operand Example 1

- C code:

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

  - g in x1, h in x2, base address (starting address) of A in x3
  - A is an ‘int’ type array (i.e., each element is a 4-byte value)



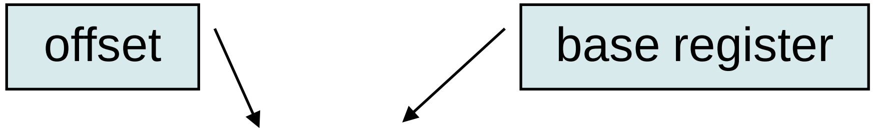
Memory Operand Example 1

- C code:

  `g = h + A[8];`

  - `g` in `x1`, `h` in `x2`, base address (starting address) of `A` in `x3`
  - `A` is an '`int`' type array (i.e., each element is a 4-byte value)

- Compiled RISC-V code:
  - Index "8" requires an offset of 32



```
lw  x10, 32(x3)     # lw = load word
add x1, x2, x10
```

Memory Operand Example 2

- C code:

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

  - h in x2, base address of A in x3

- Compiled RISC-V code:

```
lw  x10, 32(x3)    # load word
add x11, x2, x10
sw  x11, 48(x3)    # store word
```

Memory Operand Example 3

- C code:

```
for( i = 0; i < 10; i++ ) {
   sum = sum + A[i];
}
```

  - `sum` in `x1`, `i` in `x2`, base address of `A` in `x3`

- Compiled RISC-V code (only the loop body):

```
…
addi x10, x10, 4
lw   x11, 0(x10)
add  x1, x1, x11
…
```

MIPS Instructions – Memory

- **`lw, lh, lb`**
  - Load word / halfword / byte

- **`sw, sh, sb`**
  - Store word / halfword / byte

RISC-V I-format Instructions

Bit 31 Bit 0

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

```
lw x9,-123(x20)
```

immediate	x20	funct3	x9	Load
-123	20	2	9	3
1111 1000 0101	10100	010	01001	0000011

111110000101 10100 010 01001 0000011_2

1111 1000 0101 1010 0010 0100 1000 0011_2

= 0xF85A2483

RISC-V S-format Instructions

Bit 31 Bit 0

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

- Store instructions use a different binary format:
  - rs1: base address register number
  - rs2: source operand register number
  - immediate: offset added to base address
    - Designed so that rs1 and rs2 fields are always in the same place

RISC-V Machine Code Encoding Comparison

	Bit 31					Bit 0
R format	funct7	rs2	rs1	funct3	rd	opcode
I format	Immediate[11:0]		rs1	funct3	rd	opcode
S format	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode

RISC-V S-format Instructions

Bit 31 Bit 0

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

sw x9,-123(x20)

-123[11:5]	x9	x20	funct3	-123[4:0]	Store
	9	20	2		35
1111100	01001	10100	010	00101	0100011

1111100 01001 10100 010 00101 0100011$_2$

1111 1000 1001 1010 0010 0010 1010 0011$_2$

= 0xF89A22A3

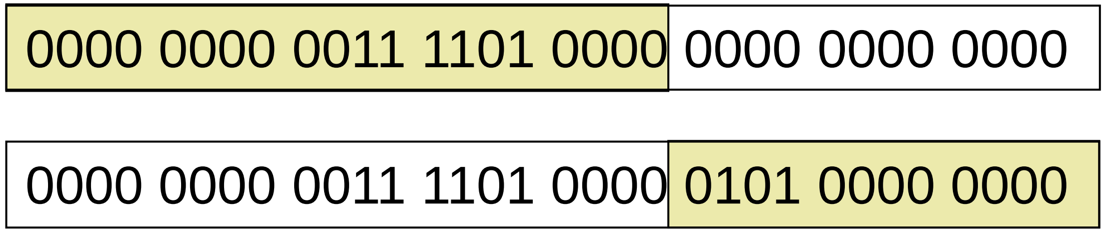
32-bit Constants

- Most constants are small
  - 12-bit immediate is sufficient for common cases
- For occasional 32-bit constant,

```
lui rd, constant
```

  - Copies a 20-bit constant to [31:12] of `rd`
  - Sets the lower 12 bits of `rd` to 0
  - e.g., **`lui x10, 0x123`** → `x10` becomes `0x00123000`

```
lui x19, 976 //0x003D0
addi x19, x19, 1280 //0x500
```



0000 0000 0011 1101 0000 0000 0000 0000

0000 0000 0011 1101 0000 0101 0000 0000

Sign Extension

- RISC-V instructions sign-extend the immediate value

```
li $t0, 11
addi $t1, $t0, -10
```

- **`lh`** and **`lb`** also sign-extends the loaded halfword or byte

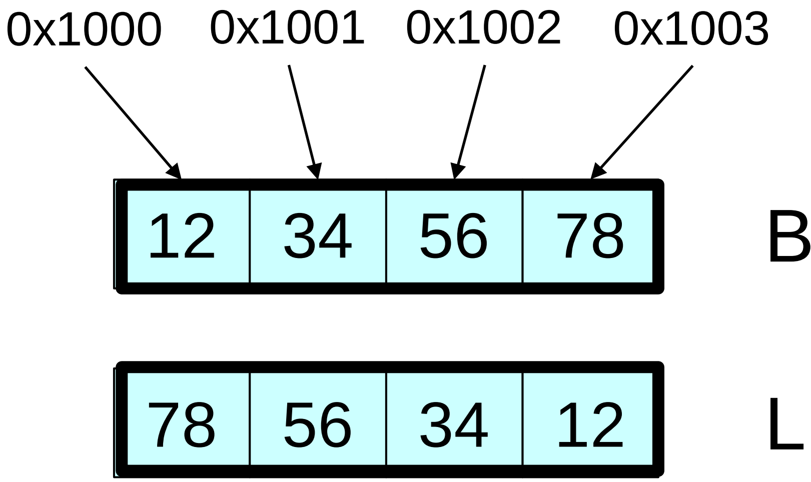
- **`lhu`** and **`lbu`** zero-extends the loaded halfword or byte

- There are no **`shu`** or **`sbu`** instructions!

Endianness

What happens if store this 4-byte value to address 0x1000 ?

$$\mathbf{12\ 34\ 56\ 78}_{16} = 1\times16^{7}+ 2\times16^{6} +3\times16^{5}\ldots \quad +7\times16^{1} +8\times16^{0}$$



0x1000 0x1001 0x1002 0x1003

12	34	56	78

Big Endian : MIPS, SPARC

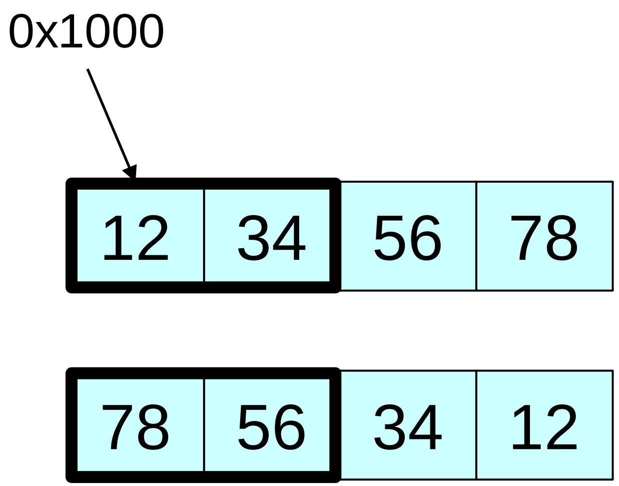
78	56	34	12

Little Endian : x86, **RISC-V**

Endianness

- Endianness does not matter when storing/loading 4 bytes at a time at 4-byte aligned addresses.

- It matters when you read in smaller sizes.



Big Endian: read 2 bytes at 0x1000 → 0x1234

Little Endian: read 2 bytes at 0x1000 → 0x5678

Conditional Operations

- Branch to a labeled instruction if a condition is true
  - Otherwise, continue to the next instruction..

- **`beq rs1, rs2, L1`**
  - if (*rs1 == *rs2), jump to the instruction labeled L1

- **`bne rs1, rs2, L1`**
  - if (*rs1 != *rs2), jump to the instruction labeled L1

```
    beq x10, x12, L1
    addi x10, x10, 1
    bne x10, x12, L2
L1: and x1, x2, x3
    sub x4, x5, x6
L2:
    xori x7, x8, x9
```

More Conditional Operations

- **`blt rs1, rs2, L1`**
  - if (*rs1 < *rs2), jump to the instruction labeled L1

- **`bge rs1, rs2, L1`**
  - if (*rs1 >= *rs2), jump to the instruction labeled L1

Signed vs. Unsigned

- Signed comparison: **`blt`**, **`bge`**
- Unsigned comparison: **`bltu`**, **`bgeu`**

- Example
  - `x22 = 1111 1111 1111 1111 1111 1111 1111 1111`
  - `x23 = 0000 0000 0000 0000 0000 0000 0000 0001`
  - `x22 < x23 // signed comparison`
    - `–1 < +1`
  - `x22 > x23 // unsigned comparison`
    - `+4,294,967,295 > +1`

Compiling If Statements

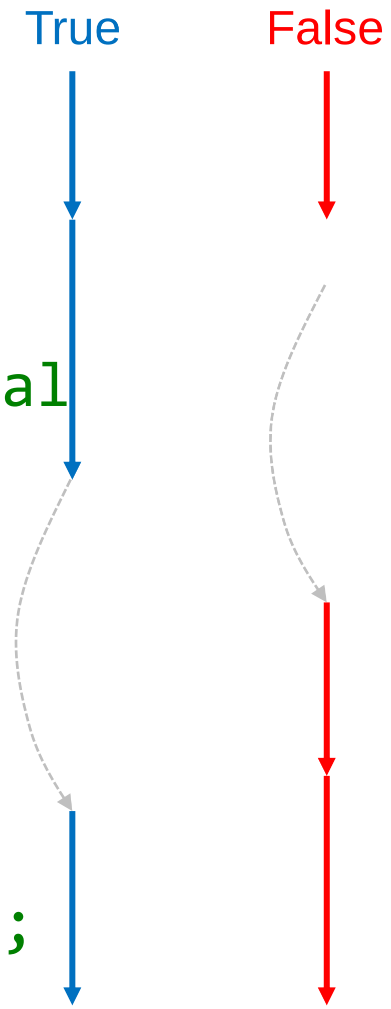
- C code:

```
if (i==j) {
    f = g + h;
}
else {
    f = g - h;
}
f = f << 1;
```

```
f,   g,   h,   i,   j   in
x19, x20, x21, x22, x23
```

- RISC-V code:

```
bne x22, x23, Else


add x19, x20, x21  //f=g+h;
beq x0, x0, Exit   //unconditional


Else:
sub x19, x20, x21  //f=g-h;


Exit: slli x19, x19, 1  //f=f<<1;
```



Compiling If Statements

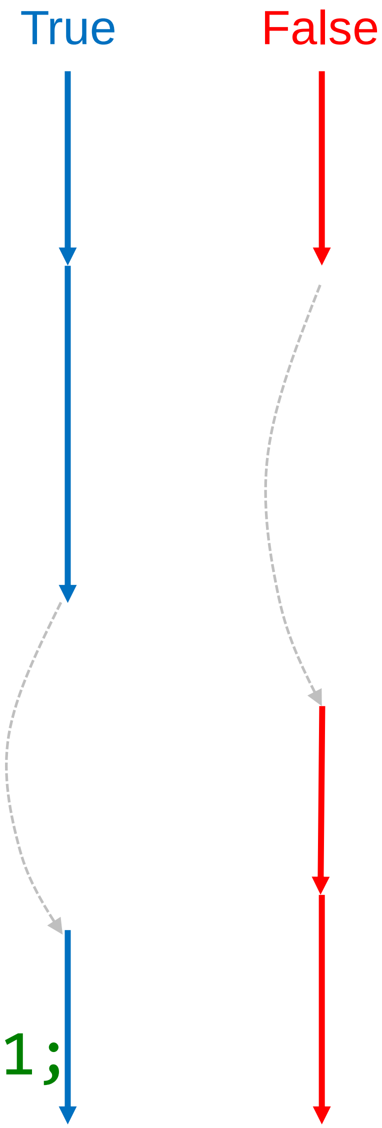
- C code:

```
if (i<10) {
    f = g + h;
}
else {
    f = g - h;
}
f = f << 1;
```

```
f,   g,   h,   i,   j   in
x19, x20, x21, x22, x23
```

- RISC-V code:

```
addi x10, x0, 10

bge x22, x10, Else


add x19, x20, x21  //f=g+h;
j   Exit


Else:
sub x19, x20, x21  //f=g-h;



Exit: slli x19, x19, 1  //f=f<<1;
```



Compiling Loop Statements

- C code:

```
k = 0;
while (i>=0) {
    i += save[k];
    k++;
}
```

i in x22, k in x24

save is an **int** array

base address of save in x25

- RISC-V code:

```
add x24, x0, x0

loop: blt  x22, x0, loop_end

      slli  x11, x24, 2

      add  x10, x11, x25

      lw   x10, 0(x10)

      add  x22, x22, x10

      addi x24, x24, 1

      beq  x0, x0, loop

loop_end: …
```

Compiling Loop Statements

- C code:

```
k = 0;
while (i>=0) {
    i += save[k];
    k++;
}
```

i in x22, k*4 in x24

save is an **int** array

base address of save in x25

- RISC-V code:

```
add x24, x0, x0

loop: blt  x22, x0, loop_end

      add  x10, x24, x25

      lw   x10, 0(x10)

      add  x22, x22, x10

      addi x24, x24, 4

      beq  x0, x0, loop

loop_end: …
```

MIPS Jump Instructions

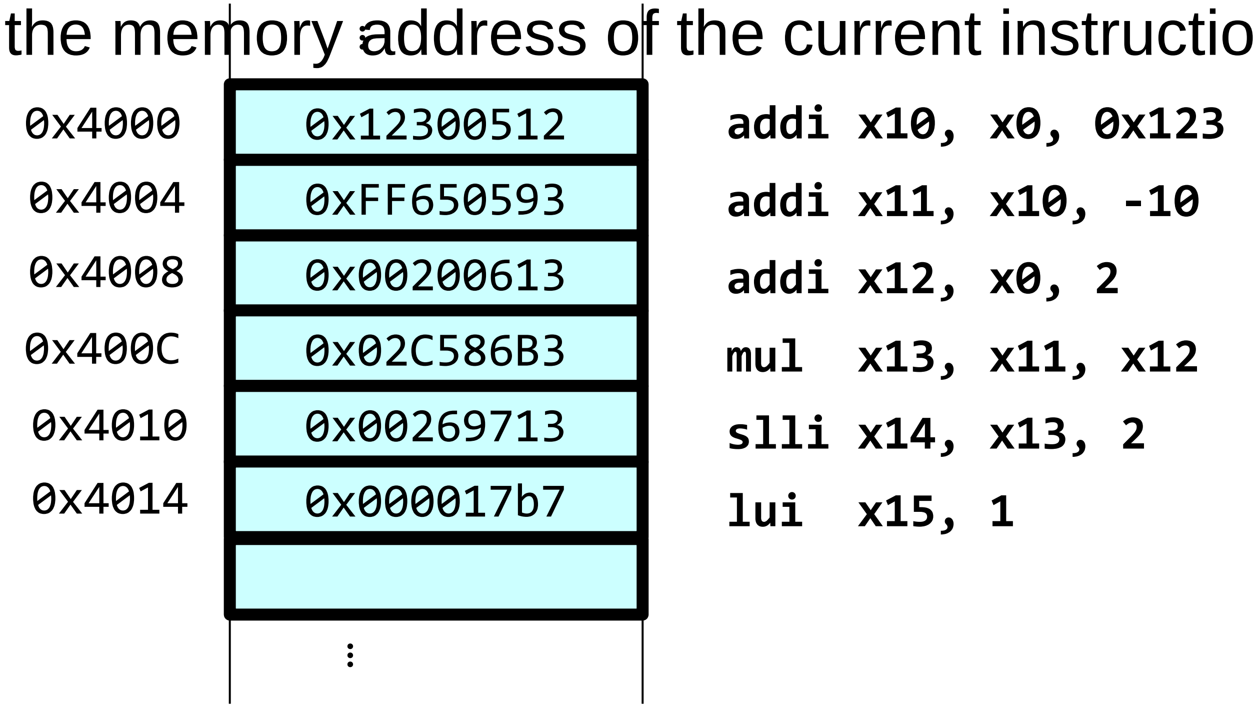
- **`jal x1, L1 :`** jump-and-link
  - Jump to L1
  - Address of the following instruction (PC+4) → x1

- **`jalr x1, Offset(x2) :`** jump register
  - Jump to offset + address in x2
  - Address of the following instruction (PC+4) → x1

- **`j L1 :`** jump
  - Pseudo-instruction
  - `jal x0, L1`

Program Counter

- Program Counter (PC)
  - Indicates the memory address of the current instruction

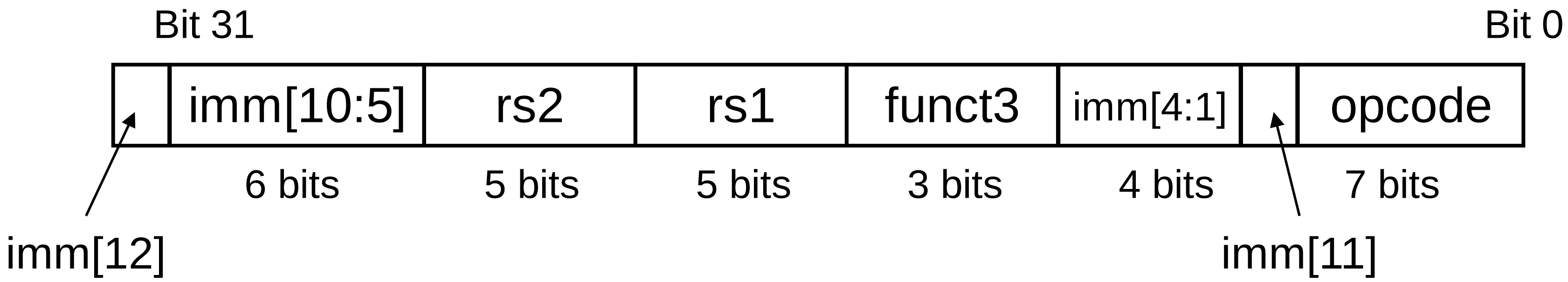


0x4000	0x12300512	**addi x10, x0, 0x123**
0x4004	0xFF650593	**addi x11, x10, -10**
0x4008	0x00200613	**addi x12, x0, 2**
0x400C	0x02C586B3	**mul x13, x11, x12**
0x4010	0x00269713	**slli x14, x13, 2**
0x4014	0x000017b7	**lui x15, 1**

Branch Addressing

- Branch instructions specify
  - Opcode, two registers, branch target
- Most branch targets are near the branch instruction
  - Immediate is a 13-bit signed value, and imm[0] is assumed to be zero.
  - Forward or backward

SB format



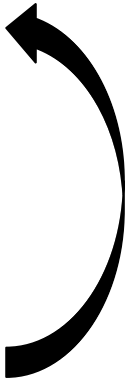
Bit 31							Bit 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
	6 bits	5 bits	5 bits	3 bits	4 bits		7 bits

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

Branch Encoding Example

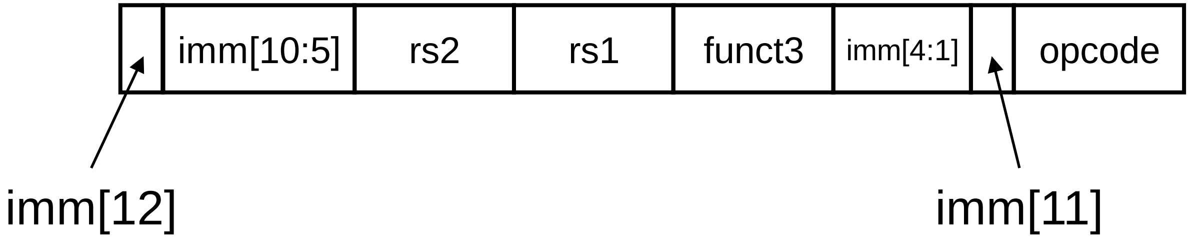
Address: 0x1000

```
L1: add x1, x2, x3

…

…

…

beq x1, x2, L1
```



-0x28 = -40

Address: 0x1028

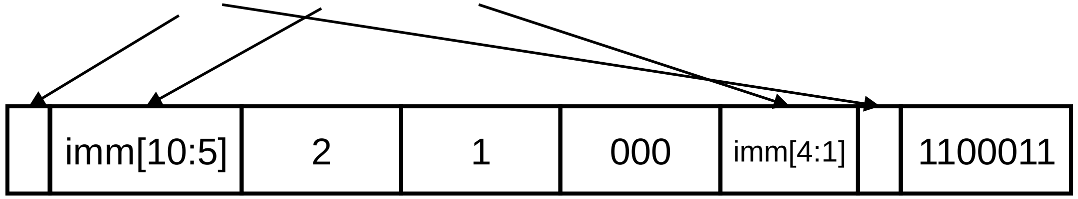


1111111011000$_2$: -40$_{10}$

1111111011000 :imm [12:0]

111111101100 :imm [12:1]

1 1 111110 1100



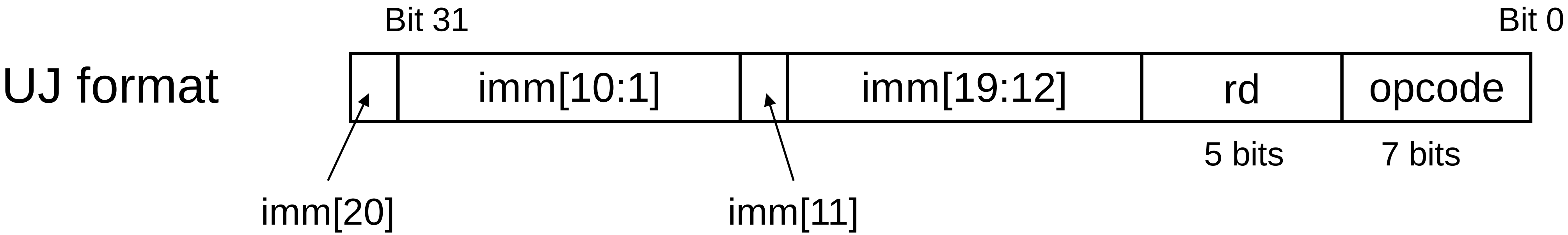
1 111110 00010 00001 000 1100 1 1100011$_2$

1111 1100 0010 0000 1000 1100 1110 0011$_2$

= 0xFC208CE3

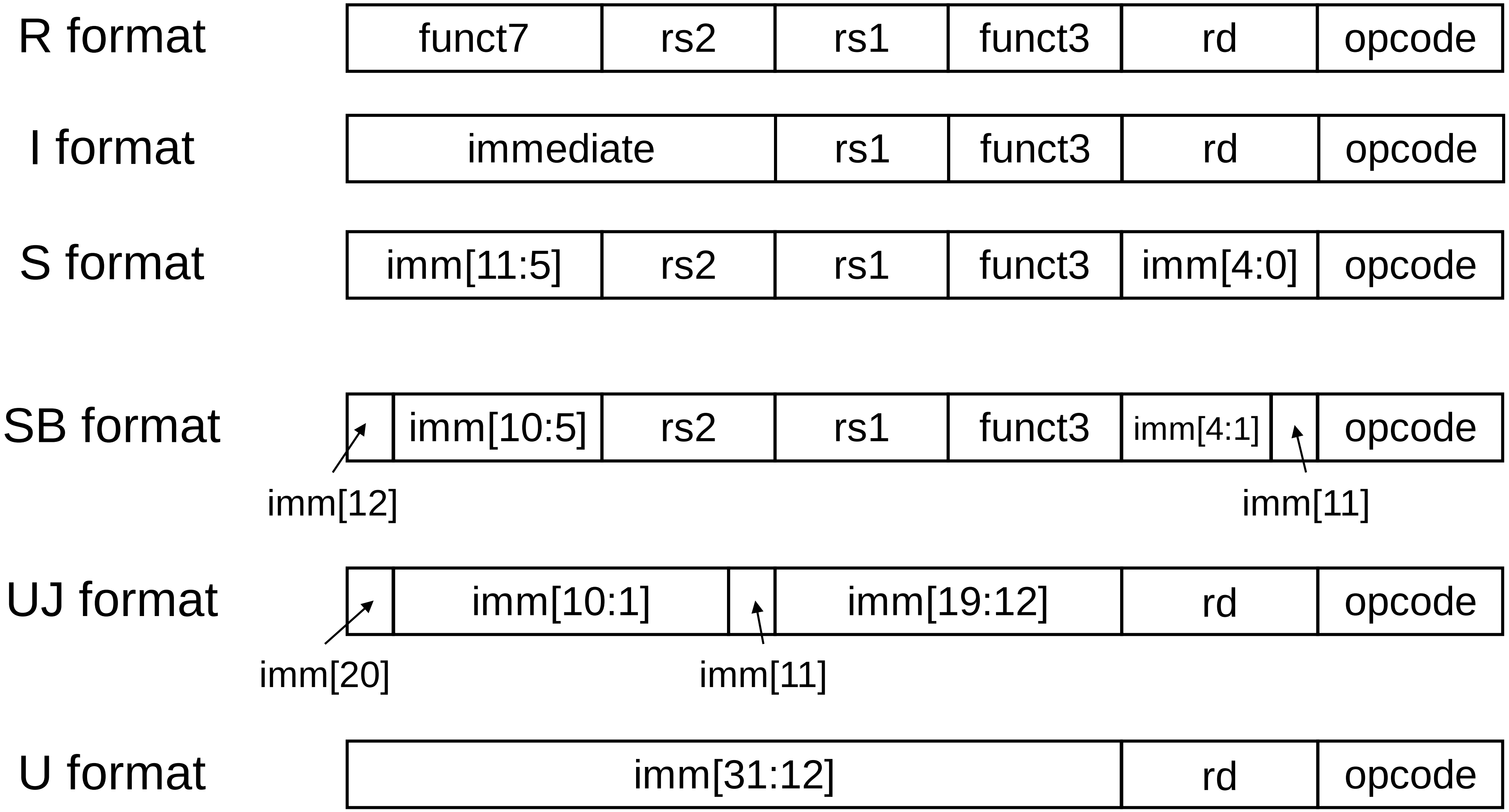
Jump Addressing

- Jump and link (`jal`) target uses a 20-bit immediate for a larger range



- For long jumps (e.g., to 32-bit absolute address),
  - `lui`: load address[31:12] to a temporary register
  - `jalr`: add address offset [11:0] to the temporary register and jump to target

RISC-V Encoding Summary



R format

funct7	rs2	rs1	funct3	rd	opcode

I format

immediate	rs1	funct3	rd	opcode

S format

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode

SB format

	imm[10:5]	rs2	rs1	funct3	imm[4:1]		opcode

imm[12]

imm[11]

UJ format

	imm[10:1]		imm[19:12]	rd	opcode

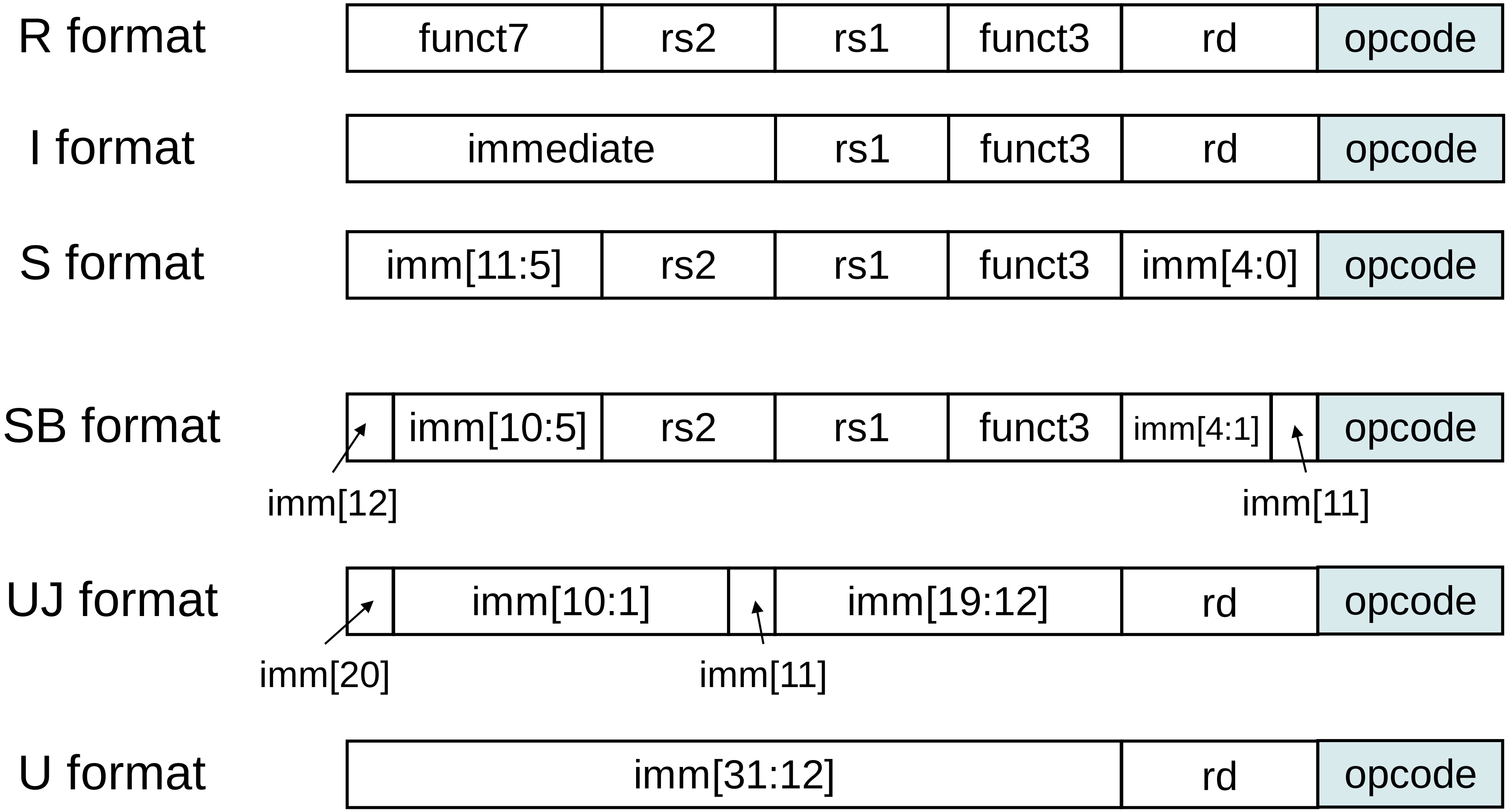
imm[20]

imm[11]

U format

imm[31:12]	rd	opcode

RISC-V Encoding Summary



R format

funct7	rs2	rs1	funct3	rd	opcode

I format

immediate	rs1	funct3	rd	opcode

S format

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode

SB format

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode

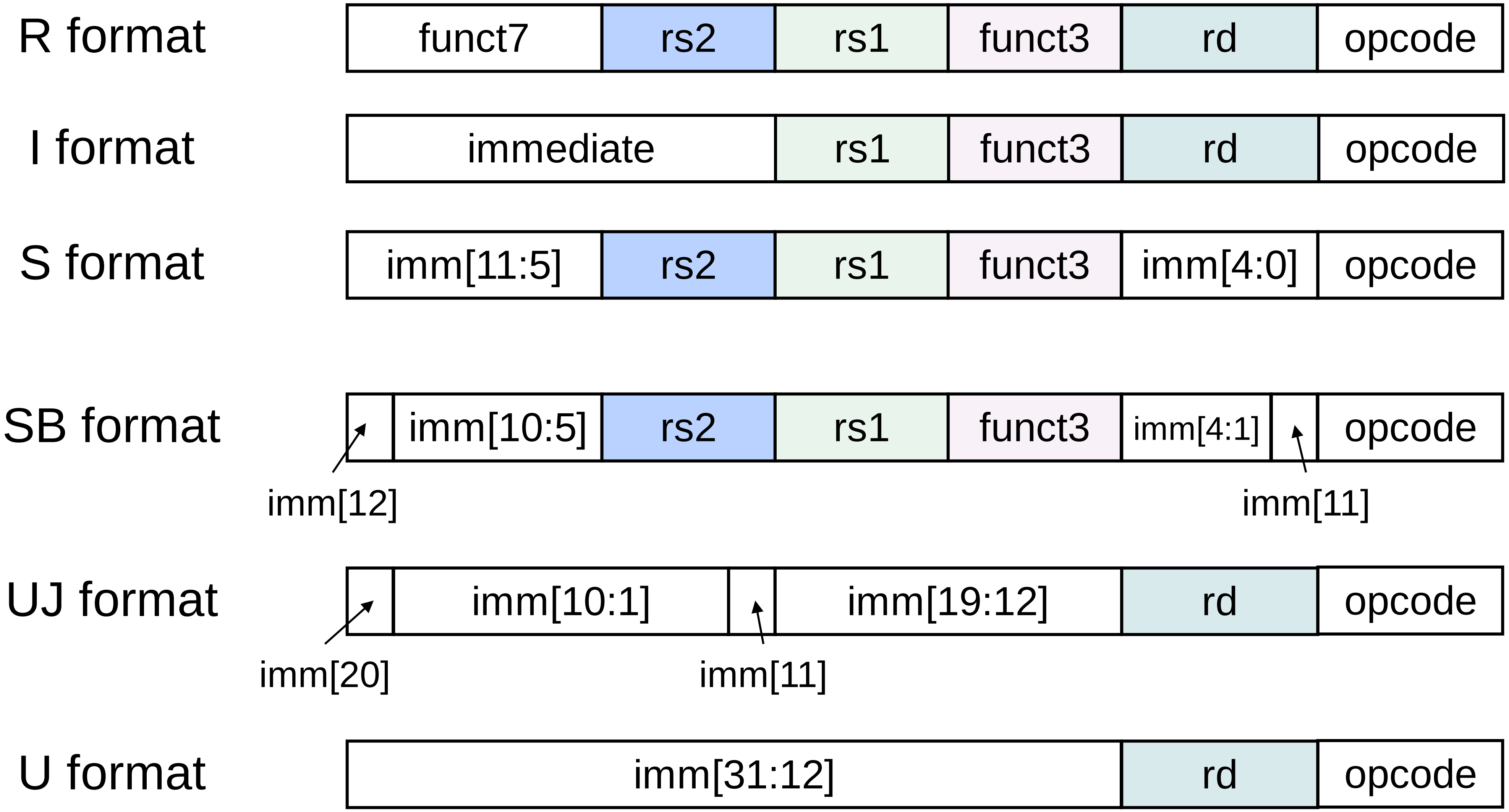
UJ format

imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode

U format

imm[31:12]	rd	opcode

RISC-V Encoding Summary



R format: funct7 | rs2 | rs1 | funct3 | rd | opcode

I format: immediate | rs1 | funct3 | rd | opcode

S format: imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode

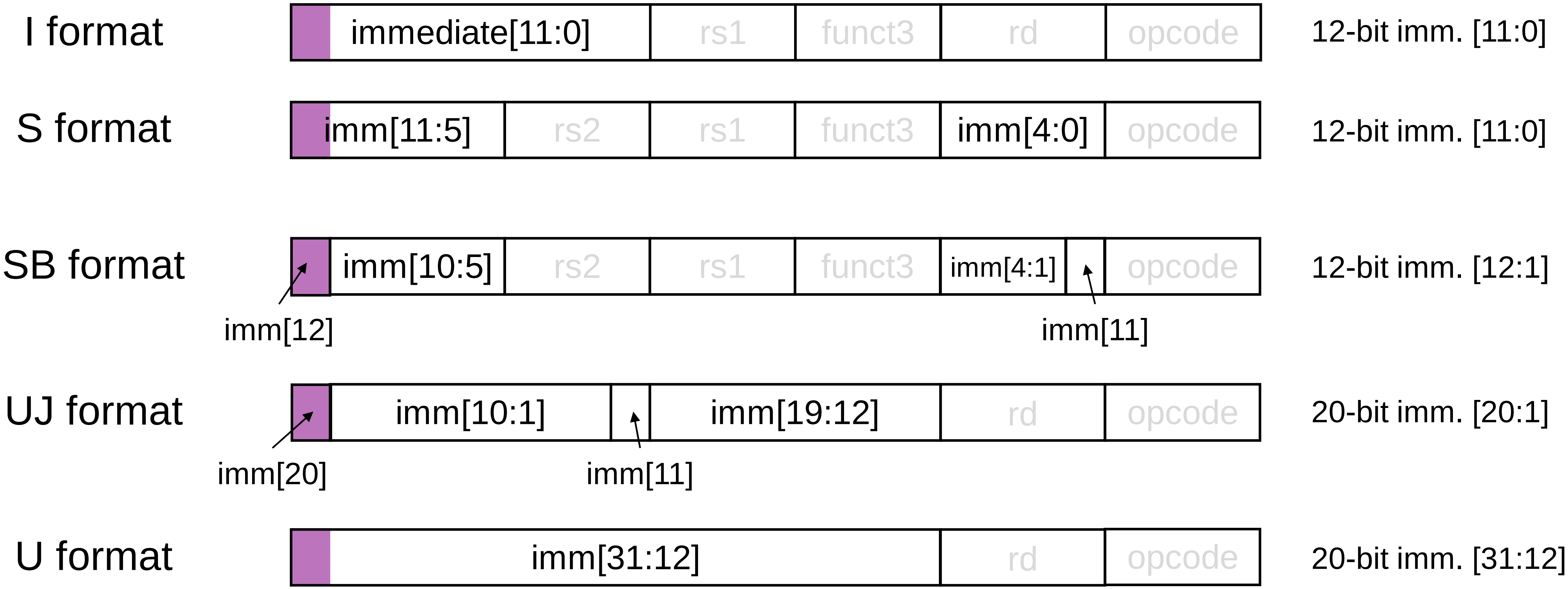
SB format: imm[12] | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | imm[11] | opcode

UJ format: imm[20] | imm[10:1] | imm[11] | imm[19:12] | rd | opcode

U format: imm[31:12] | rd | opcode

RISC-V Encoding Summary

🟪 :Instruction’s MSB is the MSB of the immediate value (for sign extension)



I format: immediate[11:0] | rs1 | funct3 | rd | opcode — 12-bit imm. [11:0]

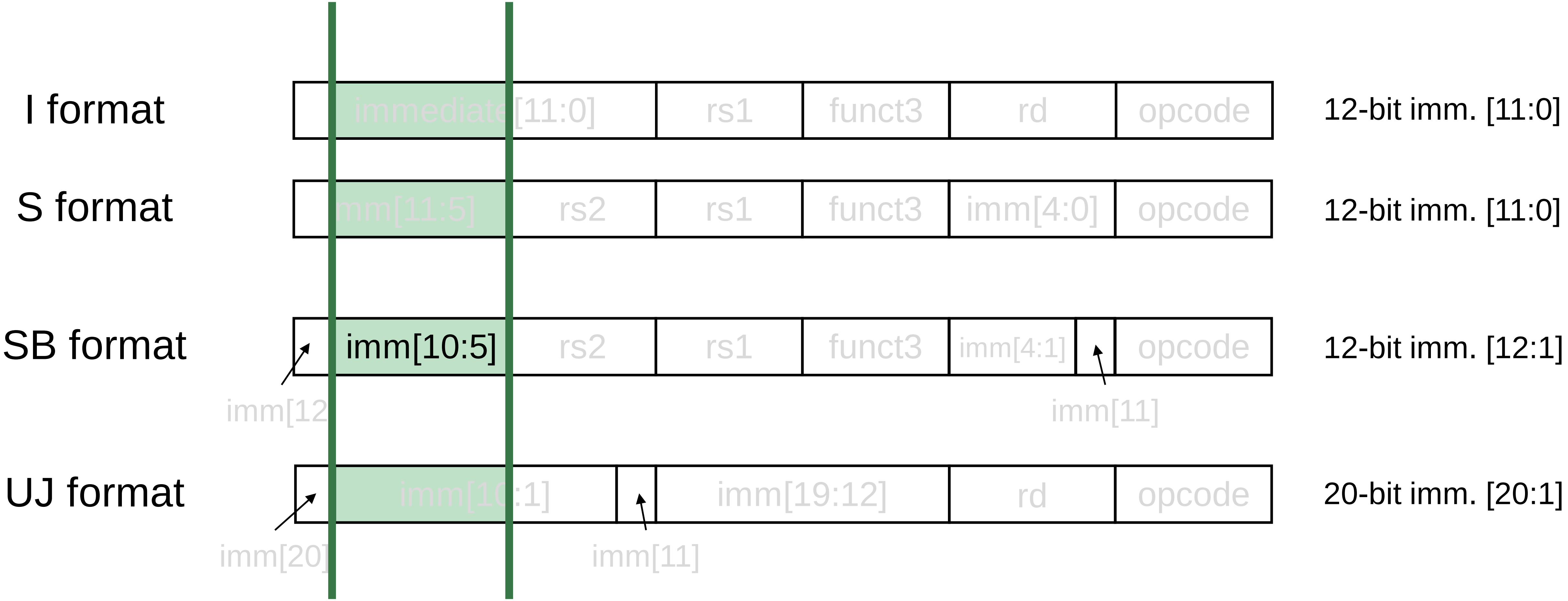
S format: imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode — 12-bit imm. [11:0]

SB format: imm[12] | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | imm[11] | opcode — 12-bit imm. [12:1]

UJ format: imm[20] | imm[10:1] | imm[11] | imm[19:12] | rd | opcode — 20-bit imm. [20:1]

U format: imm[31:12] | rd | opcode — 20-bit imm. [31:12]

RISC-V Encoding Summary



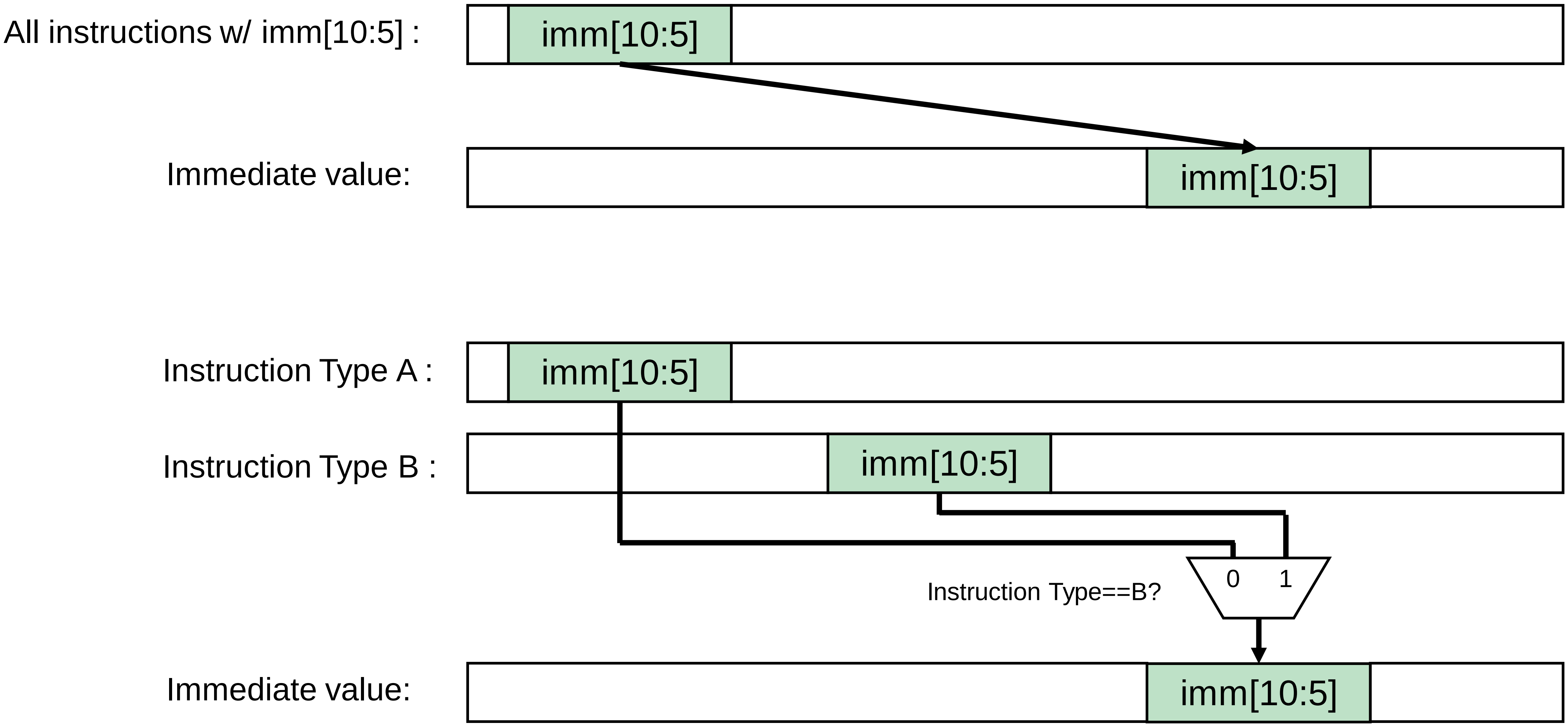
I format | immediate[11:0] | rs1 | funct3 | rd | opcode | 12-bit imm. [11:0]

S format | imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode | 12-bit imm. [11:0]

SB format | imm[12] | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | imm[11] | opcode | 12-bit imm. [12:1]

UJ format | imm[20] | imm[10:1] | imm[11] | imm[19:12] | rd | opcode | 20-bit imm. [20:1]

Aligned vs. Non-aligned Immediate Values


All instructions w/ imm[10:5] :
imm[10:5]
Immediate value:
imm[10:5]
Instruction Type A :
imm[10:5]
Instruction Type B :
imm[10:5]
0
1
Instruction Type==B?
Immediate value:
imm[10:5]