Encoding of instructions in RISC-V (S/B)

Sparsh Mittal

Courtesy for some slides: Dr Garcia and Dr Nikolic

S-format instruction encoding

RISC-V S-format Instructions

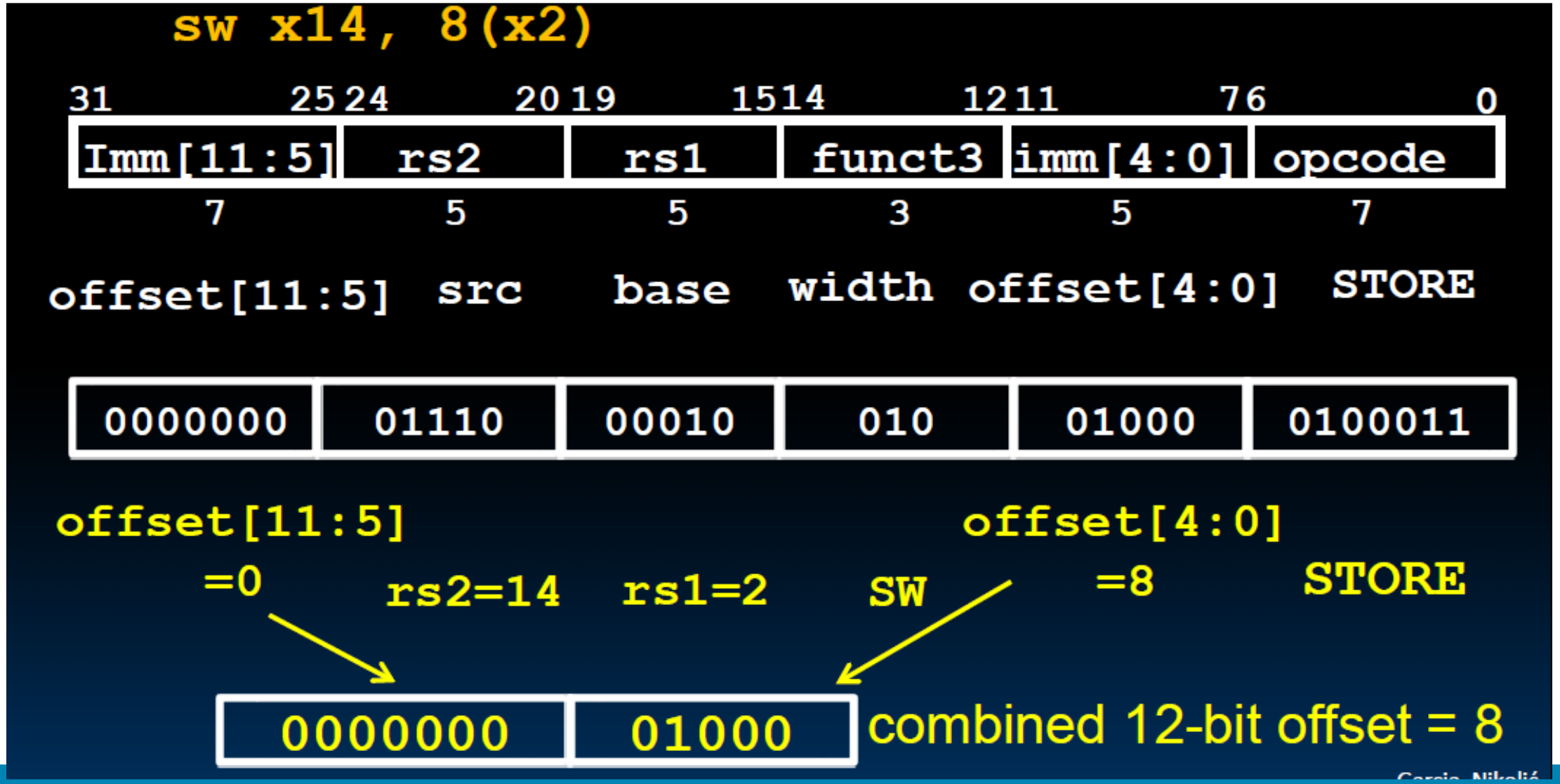
- Different immediate format for store instructions
  - rs1: base address register number
  - rs2: source operand register number
  - immediate: offset added to base address
    - Split so that rs1 and rs2 fields always in the same place

- Keeping the instruction formats as similar as possible reduces hardware complexity
- RISC-V design decision is to move low 5 bits of immediate to where **rd** field was in other instructions –keep **rs1/rs2** fields in same place
- Register names more critical than immediate bits in hardware design

sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm
bge	Branch ≥	B	1100011	0x5		if(rs1 >= rs2) PC += imm
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm

For sd, funct3 is 011, rest is same as sw.

Store instruction example



Example

- sd x9, 240(x10)

Decimal

immediate[11:5]	rs2	rs1	funct3	immediate[4:0]	opcode
7	9	10	3	16	35

240 is 000011110000
Lower 5 bits are 10000, which is 16
Upper 7 bits are 0000111 which is 7

Binary

immediate[11:5]	rs2	rs1	funct3	immediate[4:0]	opcode
0000111	01001	01010	011	10000	0100011

Example

R-type Instructions	funct7	rs2	rs1	funct3	rd	opcode	Example
add (add)	0000000	00011	00010	000	00001	0110011	add x1, x2, x3
sub (sub)	0100000	00011	00010	000	00001	0110011	sub x1, x2, x3
I-type Instructions	**immediate**		**rs1**	**funct3**	**rd**	**opcode**	**Example**
addi (add immediate)	001111101000		00010	000	00001	0010011	addi x1, x2, 1000
ld (load doubleword)	001111101000		00010	011	00001	0000011	ld x1, 1000 (x2)
S-type Instructions	**immed -iate**	**rs2**	**rs1**	**funct3**	**immed -iate**	**opcode**	**Example**
sd (store doubleword)	0011111	00001	00010	011	01000	0100011	sd x1, 1000(x2)

B-format instruction encoding

Branch instructions

- E.g., **`beq x1, x2, Label`**
- Branches read two registers but don't write to a register (similar to stores)
- How to encode label, i.e., where to branch to?

- Branches typically used for loops and conditional statement (**if-else**, **while**, **for**)
  - Loops are generally small (< 50 instructions)
  - Function calls and unconditional jumps handled with jump instructions (J-Format)

PC-relative addressing

Use the **`immediate`** field as a two's-complement offset to PC

- Branches generally change the PC by a small amount
- Can specify $\pm 2^{11}$ 'unit' addresses from the PC
- (We will see in a bit that we can encode 12-bit offsets as immediate)

- Why not use byte as a unit of offset from PC?
  - Because instructions are 32-bits (4-bytes)
  - We don't branch into middle of instruction

Support for 16-bit instructions

- Extensions to RISC-V base ISA support 16-bit compressed instructions and also variable-length instructions that are multiples of 16-bits in length
- To enable this, RISC-V scales the branch offset by 2 bytes even when there are no 16-bit instructions

- RISC-V architects wanted to support the possibility of instructions that are only 2 bytes long,
- so the branch instructions represent the number of *halfwords* between the branch and the branch target.

Branch calculation

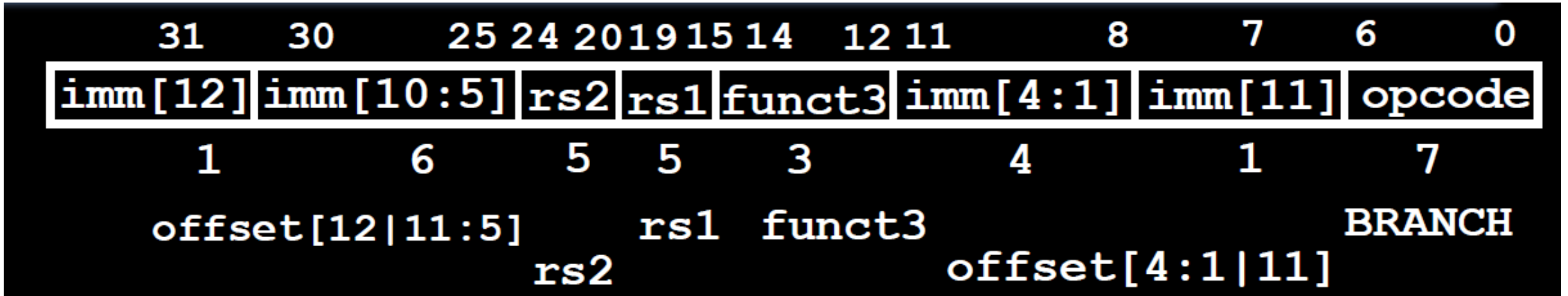
- If we don't take the branch:

  `PC = PC + 4` (i.e., next instruction)

- If we do take the branch:

  `PC = PC + immediate*2`

- Observations:
  - **`immediate`** is number of instructions to jump either forward (+) or backwards (–)



- B-format is mostly same as S-Format, with two register sources (**`rs1`**/**`rs2`**) and a 12-bit immediate **`imm[12:1]`**
- But now immediate represents values -4096 to +4094 in 2-byte increments
- The 12 immediate bits encode *even* 13-bit signed byte offsets (lowest bit of offset is always zero, so no need to store it)

B-type

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

- bne x10, x11, 2000

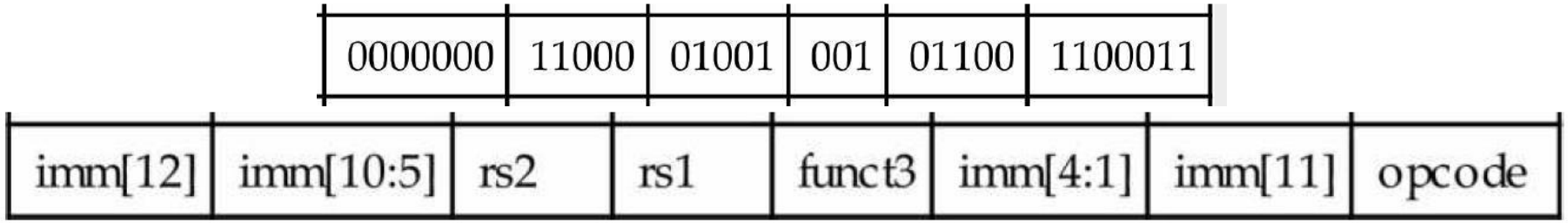
0	111110	01011	01010	001	1000	0	1100011
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode

0 0 111110 1000 in binary is 1000 in decimal. Multiplying by 2 gives us 2000.

- Loop:slli x10, x22, 3
- add x10, x10, x25
- ld x9, 0(x10)
- bne x9, x24, Exit
- addi x22, x22, 1
- beq x0, x0, Loop
- Exit:

Distance between them is 12B, so immediate should be 6

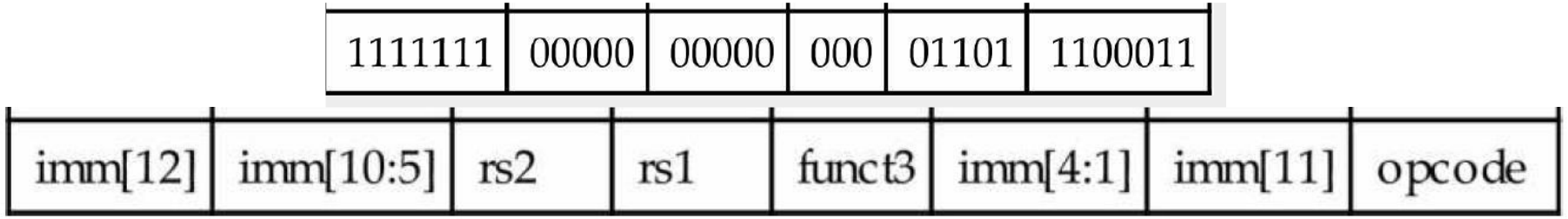
- bne x9, x24, Exit



- Loop:slli x10, x22, 3
- add x10, x10, x25
- ld x9, 0(x10)
- bne x9, x24, Exit
- addi x22, x22, 1
- beq x0, x0, Loop
- Exit:

Distance between them is -20B, so immediate should be -10

- beq x0, x0, Loop



In 2's complement, 12-bit notation for -10 is 111111110110

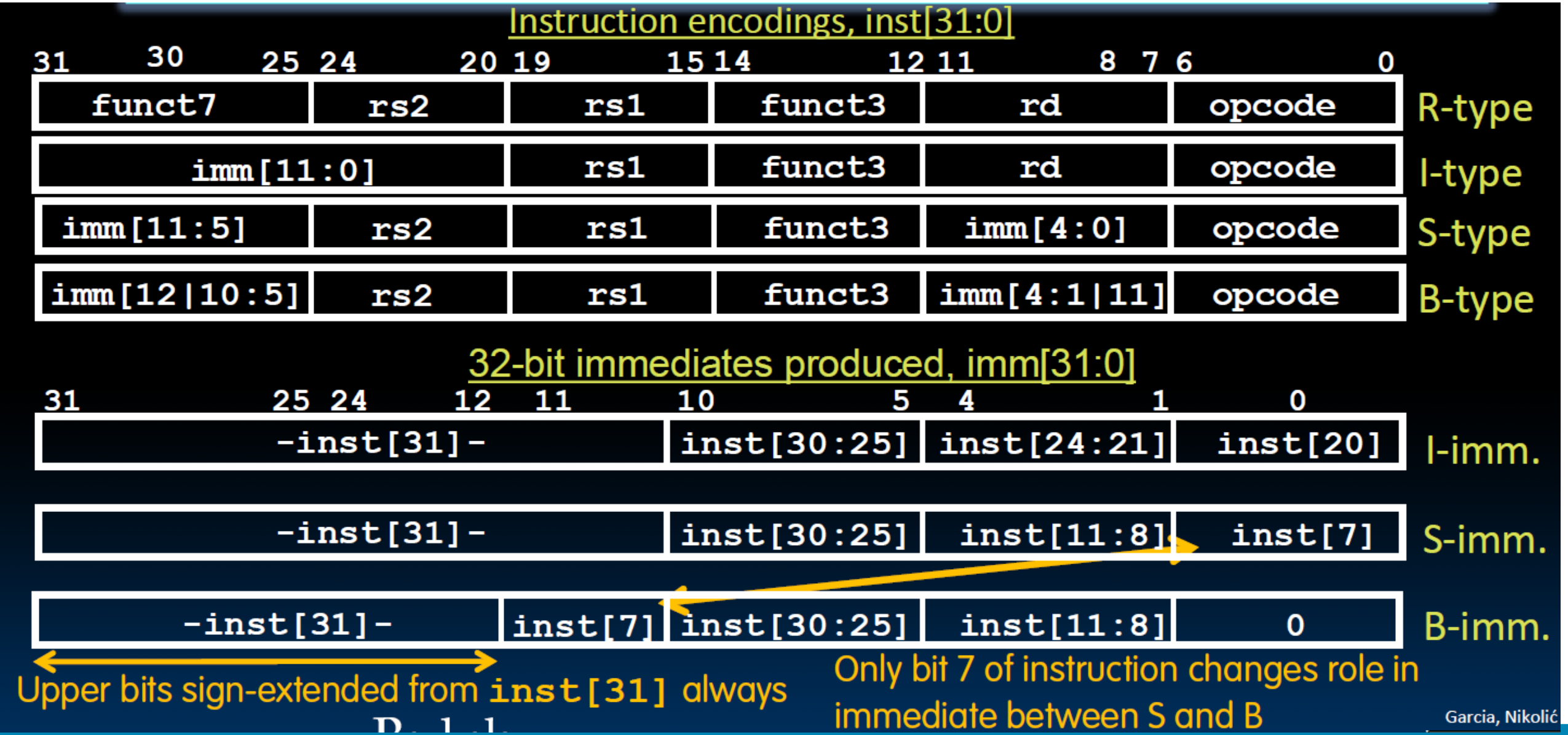
Do RISC-V encoding of instruction #4 (bne) in below sequence.

```
Loop: slli x11, x21, 3 #This is at address = 40000
add x11, x11, x25
ld x8, 0(x10)
bne x8, x24, Exit
sd x7, 120(x9)
beq x1, x5, Loop
Exit:
```

Format of bne is

- imm[12] |imm[10:5] | rs2 | rs1 | 001 | imm[4:1]| imm[11] | 1100011
- rs1 =x8 = 01000, rs2 = x24 = 11000
- Address Difference Between Exit and bne instruction is 12B, so immediate is 6, or 0000 0000 0110.
- Finally, we get
- 0 000000 11000 01000 001 0110 0 1100011

Encoding of Immediate



Branch encoding example